

Developer's Guide

This guide provides introductory conceptual material and how-to explanations for routine tasks for developers using Spread Windows Forms. It describes how an application developer would use the properties and methods in Spread to create spreadsheets on Windows Forms, bind to databases, and otherwise create a grid on data-intensive applications for the .NET platform.

- **Understanding the Product**
- **Working with the Component**
- **Spreadsheet Objects**
- **Ribbon Control**
- **Sheets**
- **Rows and Columns**
- **Headers**
- **Cells**
- **Cell Types**
- **Data Binding**
- **Customizing the Sheet Appearance**
- **Customizing Interaction in Cells**
- **Creating Tables**
- **Understanding the Underlying Models**
- **Customizing Row or Column Interaction**
- **Formulas in Cells**
- **Keyboard Interaction**
- **File Operations**
- **Printing**
- **Chart Control**
- **Customizing Drawing**
- **Touch Support with the Component**

For more information, be sure to look at the additional helpful resources:

For sample information, refer to **Working with the Component**.

For complete API reference information, refer to the **Assembly Reference (on-line documentation)**.

For a complete list of documentation, refer to the **Spread Windows Forms Documentation (on-line documentation)**.

Table of Contents

Developer's Guide	0
Getting Started	22
Handling Installation	22
Installing the Product	22
Activating the Product	23-30
End-User License Agreement	30
Creating a Runtime License	30
Handling Redistribution	30-31
Product Requirements	31-33
Using Windows Regional Settings or Options	33
Using Satellite Assemblies for Languages	33
Understanding the Spread Wizard	33
Starting the Spread Wizard	33-34
Using the Spread Wizard	34-35
Getting More Practice	35
Finding the Documentation	35-37
Starting the Spread Wizard	37-38
Tutorial: Creating a Checkbook Register	38
Adding Spread to the Checkbook Project	38-39
Setting Up the Rows and Columns of the Register	39-41
Setting the Cell Types of the Register	41-43
Adding Formulas to Calculate Balances	43-44
Understanding the Product	45
Product Overview	45-46
Shortcut Objects	46-49
Underlying Models	49
Formatted versus Unformatted Data	49-50
Cell Types	50-52

Key Features	52-54
Working with the Component	55
Adding NuGet Package in Spread Windows Forms	55-60
Migrate .NET Framework Project to .NET 6 Platform	60-65
Adding a Component to a Visual Studio 2019 Project	65-67
Adding a Component to a Visual Studio 2017 Project	67-68
Setting the Properties	68-69
Using Verbs in the Properties Window	69-71
Using Smart Tags Drop-Down	71-73
Working with Collection Editors	73-74
Adding Support for High DPI Settings	74-76
Spreadsheet Objects	77
Understanding Parts of the Component	77-78
Object Parentage	78-79
Resetting Parts of the Interface	79-82
Rich Text Editing	82-84
Improving Performance by Suspending the Layout	84-86
Ways to Improve Performance	86-87
Allowing User Functionality	87-88
Allowing the User to Zoom the Display of the Component	88-89
Working with Scroll Bars	89
Customizing the Scroll Bars	89-93
Customizing the Position in the Display	93-94
Customizing Scroll Bar Tips	94-95
Adding a Status Bar	95-96
Customizing Viewports	97-100
Customizing Split Boxes	100-102
Working With Slicers	102-108
Working with BuiltIn Dialogs	108-117

Ribbon Control	118-119
Sheets	120
Working with the Active Sheet	120-121
Working with Multiple Sheets	121-122
Customizing the Sheet Name Tabs	122-126
Navigating Sheet Tabs	126-128
Adding a Sheet	128-129
Adding ChartSheet	129-132
Removing a Sheet	132-133
Showing or Hiding a Sheet	133-135
Moving a Sheet	135-136
Selecting Multiple Sheets	136-138
Copying and Inserting a Sheet	138-141
Protecting a Worksheet	141-146
Form Controls	146-151
Adding a Title and Subtitle to a Sheet	151-153
Placing Child Controls on a Sheet	153-154
Displaying a Footer for Columns or Groups	154-162
Adding a Tag to a Sheet	162
Working with 1-Based Indexing	162
Customizing Clipboard Operation Options	163-170
Rows and Columns	171
Customizing the Number of Rows or Columns	171-172
Adding a Row or Column	173-174
Removing a Row or Column	174-175
Showing or Hiding a Row or Column	175-176
Setting the Row Height or Column Width	176-178
Resizing the Row or Column to Fit the Data	178-180
Allowing the User to Resize Rows or Columns	180-182

Using Auto Row Height	182-183
Setting Fixed (Frozen) Rows or Columns	183-187
Moving Rows or Columns	187-189
Creating Alternating Rows	189-191
Setting up Preview Rows	191-193
Input Data in Rows or Columns	193-194
Rows or Columns That Have Data	194-196
Adding a Tag to a Row or Column	196
Headers	197
Understanding Headers	197-198
Showing or Hiding Headers	198-199
Setting the Height or Width of Header	199-200
Setting the Header Text	200-201
Customizing Header Label Text	201-202
Customizing the Default Header Labels	202-204
Wrapping the Header Text	204-205
Customizing the Appearance of Headers	205-206
Customizing the Style of Header Cells	206-208
Customizing the Header Grid Lines	208-211
Adding a Gradient to Header Cells	211-212
Auto Expand Row Headers	212-213
Creating a Header with Multiple Rows or Columns	213-215
Creating a Span in a Header	215-218
Customizing the Sheet Corner Appearance	218
General Style of the Sheet Corner	218-221
Text Display in the Sheet Corner	221-222
Table Display in the Sheet Corner	222-224
Customizable Cell in the Sheet Corner	224-225
Cell Spans in the Sheet Corner	225-226

Header Count Synchronization in the Sheet Corner	226-227
Drawing (Rendering) Style	227-228
Cells	229-230
Working with the Active Cell	230-231
Adding Hyperlink in a Cell	231-235
Creating a Range of Cells	235
Working with Cell Format Strings	235-242
Managing Data on a Sheet	242
Placing and Retrieving Data	242-243
Handling Data Using Sheet Methods	243-245
Handling Data Using Cell Properties	245-246
Moving Data on a Sheet	246-247
Copying Data on a Sheet	247
Repeatedly Filling a Range of Cells with Copied Cells	247-249
Swapping Data on a Sheet	249-250
Removing Data from a Sheet	250-251
Remove Duplicates from Range	251-258
Text to Columns	258-267
Creating Data Type for Custom Objects	267-272
Displaying Cell Data	272-273
Resizing the Data to Fit the Cell	273-274
Allowing Cell Data to Overflow	274-276
Aligning Cell Contents	276-279
Resizing a Cell to Fit the Data	279-280
Creating a Span of Cells	280-282
Allowing Cells to Merge Automatically	282-284
Adding a Note to a Cell	284-288
Adding a Tag to a Cell	288-289
Displaying Text Tips in a Cell	289-290

Working with Cell Format Strings	290-297
Working with Pattern and Gradient Fill Effects	297-298
Inserting Cells	298-307
Setting Rich Text in a Cell	307-310
Adding a Comment to a Cell	310-317
Adding Image in a Cell	317-323
Formatting a Cell Value	323-324
Cell Types	325
Understanding Cell Type Basics	325-326
Understanding How Cell Types Work	326-327
Understanding How Cell Types Display and Format Data	327-330
Understanding How Cell Type Affects Model Data	330-331
Working with Editable Cell Types	331-332
Setting a General Cell	332-333
Setting a Text Cell	333-334
Setting a Date-Time Cell	334-336
Customizing the Pop-Up Date-Time Control	336-338
Setting a GcCharMask Cell	338-341
Setting a GcDateTime Cell	341-342
Setting a GcMask Cell	342-345
Setting a GcNumber Cell	345-347
Setting a GcTextBox Cell	347-349
Setting a GcTimeSpan Cell	349-351
Setting a Number Cell	351-357
Setting a Currency Cell	357-358
Setting a Mask Cell	358-360
Setting a Percent Cell	360-361
Setting a Regular Expression Cell	361-362
Setting a Rich Text Cell	362-366

Working with Graphical Cell Types	366
Setting a Barcode Cell	367-372
Setting a Button Cell	372-376
Setting a Check Box Cell	376-379
Setting a Color Picker Cell	379-383
Setting a Combo Box Cell	383-386
Allowing a Combo Box Cell to Handle a Double Click	386-388
Setting a Hyperlink Cell	388-391
Setting an Image Cell	391-394
Setting a List Box Cell	394-396
Setting a Multiple-Column Combo Box Cell	396-399
Setting a Multiple Option Cell	399-401
Setting a Progress Indicator Cell	401-404
Setting a Slider Cell	404-407
Understanding Additional Features of Cell Types	407
Allowing the Display of Buttons in a Cell	407-408
Displaying Spin Buttons	408-409
Limiting Values for a Numeric Cell	409-411
Customizing the Pop-Up Calculator Control	411-412
Customizing Automatic Completion (Type Ahead)	412-413
Working with a SubEditor	413-414
Creating a Custom Cell Type	414-418
Data Binding	419
Binding to Data	419
Binding Spread to an External Data Set	419-420
Binding a Cell Range in Spread to an External Data Source	420-422
Binding a Cell Range in Spread as a Data Source to an External Control	422-424
Binding a Combo Box to a DataReader	424-425
Customizing Data Binding	425

Adding an Unbound Column to a Bound Sheet	426
Customizing Column and Field Binding	426-429
Customizing Cell Types for Bound Sheets	429-431
Customizing Column Headers for Bound Sheets	431-433
Adding to Bound Data	433
Adding a Row to a Bound Sheet	433-434
Adding an Unbound Row to a Bound Sheet	434-436
Working with Hierarchical Data Display	436-441
Creating a Hierarchical Display Manually	441-442
Creating Custom Hierarchy Icons	442
Tutorial: Binding to a Corporate Database (Visual Studio 2013 or later)	442-452
Customizing the Sheet Appearance	453
Customizing the Dimensions of the Component	453-454
Customizing the Individual Sheet Appearance	454-455
Setting the Background Colors for a Sheet	455-456
Coloring a Cell	456-459
Setting a Background Image for a Sheet	459-460
Setting a Background Image to a Cell	460-461
Customizing the Appearance of a Cell	461-462
Customizing the Outline of the Component	462-463
Displaying Grid Lines on a Sheet	463-466
Creating and Customizing Cell Borders	466-470
Creating a Complex Border with Multiple Lines	470-471
Customizing the Overall Component Appearance	471-472
Setting the Component to the Original Appearance	472-473
Applying a Skin to the Component	473-474
Creating a Custom Skin for a Component	474-477
Applying a Skin to a Sheet	477-479
Creating a Custom Skin for a Sheet	479-481

Saving and Loading a Skin	481-482
Saving a Skin	482
Loading a Skin	482-483
Creating and Applying a Style for Cells	483-486
Using Conditional Formatting of Cells	486
Creating Conditional Formatting with Rules	486-487
Color Scale Rules	487-488
Data Bar Rule	488-490
Icon Set Rule	490-491
Highlighting Rules	491-493
Top, Bottom, or Average Rules	493-494
Setting up Conditional Formatting of a Cell	494-496
Customizing the Display of the Pointer	496-498
Customizing the User Interface Images	498-499
Using XP Themes with the Component	499-500
Customizing the Renderers	500-504
Handling Right-to-Left Layouts	504
Customizing Painting of Parts of the Component	504-505
Text Rendering with GDI	505-506
Applying Theme to Customize the Appearance	506-507
Customizing Interaction in Cells	508
Using Edit Mode and Focus	508
Understanding Edit Mode in a Cell	508-509
Locking a Cell	509-511
Customizing User Selection and Deselection of Data	511-512
Specifying What the User Can Select	512-515
Customizing the Selection Appearance	515-517
Working with Selections	517-519
Hiding the Selection When Focus is Lost	519-520

Working with Deselections	520-522
Using Drag Operations to Fill Cells	522
Filling Cells with Drag and Drop	522-523
Filling Cells with Drag and Fill	523-527
Filling Cells with Drag and Move	527
Using Validation	527-528
Using Validation in Cells	528-529
Using a Cell Comparison Validator	529-530
Using a Character Format Validator	530-531
Using a String Comparison Validator	531-532
Using a Value Comparison Validator	532-533
Using an Encoding Validator	533-534
Using the Exclude List Validator	534
Using the Include List Validator	534-535
Using a Pair Validator	535-536
Using the Range Validator	536-537
Using a Regular Expression Validator	537-538
Using a Required Field Validator	538-539
Using a Required Type Validator	539-540
Using a Surrogate Character Validator	540
Using a Text Length Validator	540-541
Validating User Input	541-545
Customizing the User Error Messages	545
Displaying Error Icons in Cells or Rows	545-546
Using Visible Indicators in the Cell	546
Locating the Pointer Using HitTest	546-547
Returning Information for a Clicked Cell	547
Preventing a Cell from Receiving Focus	547-548
Customizing the Focus Indicator for a Cell	548-550

Customizing Undo and Redo Actions	550-551
Customizing Interaction Based on Events	551-552
Adding a Context Menu to a Component	552-553
Tables	554
Adding a Table	554-555
Using Table Filters	555-558
Resizing a Table	558-559
Sorting a Table	559-561
Setting Table Styles	561-563
Binding a Table	563-567
Adding a Table Formula	567-569
Understanding Structured References	569
Using Operators and Special Items	569-570
Understanding Structured Reference Syntax Rules	570-571
Using Structured References	571-572
Understanding the Underlying Models	573
Finding More Details on the Sheet Models	573
Understanding the Sheet Model Classes and Interfaces	573-575
Understanding the Data Model	575-577
Understanding the Axis Model	577-578
Understanding the Selection Model	578-579
Understanding the Span Model	579
Understanding the Style Model	579-582
Creating a Custom Sheet Model	582-584
Understanding the Optional Interfaces	584-585
Customizing Row or Column Interaction	586
Managing Sorting of Rows of User Data	586-587
Allowing the User to Automatically Sort Rows	587-589
Using Automatic Sorting	589-590

Sorting Rows, Columns, or Ranges	590-591
Setting the Appearance of Sort Indicators	591-593
Managing Cell Range Sorting	593-594
Managing Filtering of Rows of User Data	594
Allowing the User to Filter Rows	595-597
Setting the Appearance of Filtered Rows	597-600
Customizing Simple Filtering	600
Understanding Simple Row Filtering	601-602
Customizing the Filter List	602
Defining the Contents of the Filter Item List	602-605
Defining the Order of the Items in the Filter Item List	605-606
Setting the Appearance of the Display of the Filter Item List	606-608
Creating a Custom Filter	608-613
Customizing Enhanced Filtering	613-616
Adding a Custom Sort Dialog	616-617
Customizing the Filter Bar	617-619
Setting the Appearance of Filter Indicators	619
Using Custom Filter Indicator Images	619-620
Showing or Hiding Filter Indicators	620-621
Determining Which Header Row Displays the Indicators	621
Managing Cell Range Filtering	621-623
Managing Grouping of Rows of User Data	623
Allowing the User to Group Rows	623-624
Using Grouping	624-625
Setting the Appearance of Grouped Rows	625-627
Customizing the Group Bar	627
Creating a Custom Group	627-631
Interoperability of Grouping with Other Features	631
Managing Outlines (Range Groups) of Rows and Columns	631-632

Using an Outline (Range Group) of Rows or Columns	632-634
Customizing the Appearance of an Outline (Range Group)	634-636
Interoperability of Outlines with Other Features	636
Customizing User Searching of Data	636-637
Allowing the User to Perform a Standard Search	637-638
Allowing the User to Perform an Advanced Search	638
Searching for Data with Code	638-639
Formulas in Cells	640
Placing a Formula in Cells	640-643
Displaying Formula in Cells	643-644
Specifying a Cell Reference in a Formula	644-646
Specifying a Sheet Reference in a Formula	646-647
Specifying an External Reference in a Formula	647
Using a Circular Reference in a Formula	647-649
Using DataTable Formula	649-653
Nesting Functions in a Formula	653-654
Recalculating and Updating Formulas Automatically	654-655
Finding a Value Using GoalSeek	655
Allowing the User to Enter Formulas	655-657
Creating and Using a Custom Name	657-658
Creating and Using a Custom Function	658-660
Creating and Using a Visual Function	660-662
Creating and Using External Variable	662-663
Using External Variables with Text Box Control	663-665
Using the Array Formula	665-666
Working With Dynamic Array Formulas	666-679
Working with the Formula Text Box	679-682
Setting up the Formula Provider	682-684
Setting up the Name Box	684-685

Using Language Package	685
Available Language Packages for WinForms	685-686
Creating and Using a Custom Language Package	686-688
Accessing Data from Header or Footer	688-690
Auto Format Formulas	690-692
Managing External Reference	692-695
Precedents and Dependents	695-703
Sparklines	704
Add Sparklines Using Methods	704-706
Specifying Horizontal and Vertical Axes	706-708
Customizing Markers and Points	708-710
Working with Sparklines	710-713
Add Sparklines using Formulas	713-714
Column, Line, and Winloss Sparkline	714
Area Sparkline	714-717
BoxPlot Sparkline	717-719
Bullet Sparkline	720-721
Cascade Sparkline	721-723
Gauge KPI Sparkline	723-724
Hbar and Vbar Sparkline	724-726
Histogram Sparkline	726-730
Image Sparkline	730-733
Month and Year Sparkline	734-737
Pareto Sparkline	737-739
Pie Sparkline	739-741
Scatter Sparkline	741-742
Spread Sparkline	742-745
Stacked Sparkline	745-747
Vari Sparkline	747-750

Keyboard Interaction	751
Underlying Keystroke Processing	751-752
Factors of Keyboard Map Usage	752-754
Default Keyboard Navigation	754-761
Default Keyboard Maps	761-762
Default Map for Normal and WhenFocused	762
Default Map for Normal and WhenAncestorOfFocused	762-764
Default Map for ReadOnly and WhenFocused	764
Default Map for ReadOnly and WhenAncestorOfFocused	764
Default Map for ReadOnly and WhenAncestorOfFocused	764-765
Default Map for RowMode and WhenAncestorOfFocused	765-766
Default Map for SingleSelect and WhenFocused	766
Default Map for SingleSelect and WhenAncestorOfFocused	766
Default Map for MultiSelect and WhenFocused	766-767
Default Map for MultiSelect and WhenAncestorOfFocused	767
Default Map for ExtendedSelect and WhenFocused	767
Default Map for ExtendedSelect and WhenAncestorOfFocused	767-768
Deactivating the Default Keyboard Map	768-769
Changing the Default Keyboard Map	769-770
Using Input Maps with Action Maps	770-774
Customizing the Input Maps	774-776
Changing an Input Map for a Child View	776-778
Saving and Loading Map Files	778-779
Using Excel-compatible Keyboard Shortcuts	779-781
Using the Excel Compatibility Input Maps	781-782
Default Map for Excel Compatibility	782-784
Events from User Actions	785
Clicking Actions	785-788
Selecting Actions	788-789

Entering Data Actions	789
Sheet-Level Actions	789-790
Interactivity Actions	790
Shape Actions	790
Print Actions	790-791
File Operations	792
Saving Data to a File	792
Saving to a Spread XML File	792-793
Saving to an Excel File	793-795
Saving to a Text File	795-796
Saving to an Image File	796-797
Storing Excel Summary and View	798-799
Saving to an HTML Table	799-800
Saving Spreadsheet Data to Simple XML	800-801
Opening Existing Files	801
Opening a Spread XML File	801-802
Opening an Excel File	802-803
Opening a Custom Text File	803-804
Opening a Spread COM File	804-805
Using Serialization	805-806
Implementing a Serializer Class	806-811
Parsing Formulas in Custom XML Deserialization	811-812
Storing Excel Summary and View	812-813
Printing	814
Specifying What to Print	814
Printing an Entire Sheet	814-816
Printing to PDF	816-817
Printing a Child View of a Hierarchical Display	817-818
Printing Particular Pages	818-820

Printing the Portion of the Sheet with Data	820
Printing a Range of Cells on a Sheet	820-822
Printing an Area of the Sheet	823
Printing a Sheet with Cell Notes	823-825
Printing a Sheet with Shapes	825-826
Printing in Duplex Mode	826
Customizing the Appearance of the Printing	826
Understanding the Printing Options	826-831
Customizing the Print Job Settings	831-834
Customizing the Printed Page Layout	834-835
Customizing the Printed Page Header or Footer	835-845
Customizing the Print Preview Dialog	845-847
Repeating Rows or Columns on Printed Pages	847
Adding a Page Break	847-848
Adding a Watermark to a Printed Page	848-849
Optimizing the Printing	850
Optimizing the Printing Using Rules	850-853
Optimizing the Printing Using Size	853
Displaying Dialogs for Users	853
Displaying a Print Dialog for the User	853-854
Displaying an Abort Message for the User	854
Providing a Preview of the Printing	854-856
Chart Control	857-858
Understanding Charts	858-859
Chart User Interface Elements	859-861
Chart Object Model	861-862
Chart Types and Views	862-864
Plot Types	864
Y Plot Types	864-865

Bar Charts	865-868
Area Charts	868-869
Box Whisker Charts	869-872
Funnel Charts	872-874
Histogram Charts	874-876
Line Charts	876-878
Market Data (High-Low) Charts	878-879
Pareto Charts	879
Point Charts	879-881
Stripe Charts	881
Waterfall Charts	881-883
XY Plot Types	883
XY Bubble Charts	883-884
XY Point Charts	884
XY Line Charts	884-885
XY Stripe Charts	885
XYZ Plot Types	885-886
XYZ Point Charts	886-887
XYZ Line Charts	887-888
XYZ Surface Charts	888-889
XYZ Stripe Charts	889
Pie Plot Types	889
Doughnut Charts	889-890
Pie Charts	890-891
Polar Plot Types	891
Polar Point Charts	891-892
Polar Line Charts	892-893
Polar Area Charts	893-894
Polar Stripe Charts	894

Radar Plot Types	894-895
Radar Point Charts	895-896
Radar Line Charts	896-897
Radar Area Charts	897
Radar Stripe Charts	897-898
Data Plot Types	898
Plot area	898-899
Series	899-900
Walls	900-901
Axis	901-902
Chart Line Style	902-904
Elevation and Rotation	904-905
Lighting, Shapes, and Borders	905-908
Labels	908-912
Legends	912-913
Creating Charts	913-914
Using the Chart Control on sheet	914-919
Allowing the User to Change the Chart	919-920
Saving or Loading a Chart	920-921
Using the Chart Control	921-923
Save/Load Chart Control	923-924
Creating Plot Types	924
Creating a Y Plot	924-927
Creating an XY Plot	927-929
Creating an XYZ Plot	929-933
Creating a Pie Plot	933-937
Creating a Polar Plot	937-940
Creating a Radar Plot	940-943
Creating a Sunburst Chart	943-946

Creating a Treemap Chart	946-950
Combining different types of plots	951-952
Connecting to Data	952-953
Using a Bound Data Source	953-954
Using an UnBound Data Source	954-955
Binding with cell range	955-958
Advanced chart settings	958
Fill Effects	958-960
View Type	960-961
Customizing Drawing	962
Working with Shapes in Code	962-964
Creating Camera Shapes	964-966
Allowing the User to Draw with a Tablet PC	966-967
Working With Shapes (Enhanced Shape Engine)	967-980
Creating Enhanced Camera Shape	980-999
Working with Images	999-1002
Touch Support with the Component	1003
Understanding Touch Messages	1003
Using a Touch Keyboard	1003- 1004
Using the Touch Menu Bar	1004- 1008
Using Touch Support	1008- 1009
Using Touch Support with AutoFit	1009
Using Touch Support with Cell Notes	1009
Using Touch Support with Charts	1009- 1010
Using Touch Support with Clipboard Operations	1010
Using Touch Support with Drag and Fill	1010- 1012

<u>Using Touch Support with Drop-Down Elements</u>	1012- 1013
<u>Using Touch Support with Editable Cells</u>	1013- 1014
<u>Using Touch Support with InputMan Cells</u>	1014- 1016
<u>Using Touch Support with Filtering</u>	1016- 1017
<u>Using Touch Support with Range Grouping</u>	1017- 1018
<u>Using Touch Support with Grouping</u>	1018- 1019
<u>Using Touch Support when Moving Columns or Rows</u>	1019- 1020
<u>Using Touch Support when Resizing Columns or Rows</u>	1020- 1021
<u>Using Touch Support with Scrolling</u>	1021- 1023
<u>Using Touch Support with Selections</u>	1023- 1024
<u>Using Touch Support with Shapes</u>	1024
<u>Using Touch Support when Sorting</u>	1025
<u>Using Touch Support with Viewports</u>	1025
<u>Using Touch Support with the Tab Strip</u>	1025- 1026
<u>Using Touch Support with Zooming</u>	1026- 1027
1. <u>Index</u>	1028- 1043

Getting Started

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

- **Handling Installation**
 - **Installing the Product**
 - **Activating the Product**
 - **End-User License Agreement**
 - **Creating a Runtime License**
 - **Handling Redistribution**
 - **Product Requirements**
 - **Using Windows Regional Settings or Options**
 - **Using Satellite Assemblies for Languages**
- **Understanding the Spread Wizard**
 - **Starting the Spread Wizard**
 - **Using the Spread Wizard**
- **Getting More Practice**
 - **Finding the Documentation**
 - **Getting Technical Support (on-line documentation)**
- **Tutorial: Creating a Checkbook Register**
 - **Adding Spread to the Checkbook Project**
 - **Setting Up the Rows and Columns of the Register**
 - **Setting the Cell Types of the Register**
 - **Adding Formulas to Calculate Balances**

Handling Installation

The following tasks involve installing and redistributing the product.

- **Installing the Product**
- **Activating the Product**
- **End-User License Agreement**
- **Creating a Runtime License**
- **Handling Redistribution**
- **Product Requirements**
- **Using Windows Regional Settings or Options**
- **Using Satellite Assemblies for Languages**

Installing the Product

Installation instructions and a list of installed files for Spread Windows Forms is provided in the Read Me file that accompanies this product. To view the Read Me file, do one of the following:

1. From the **Start** menu choose **Program Files -> MESCIUS -> Spread.NET [version] -> Spread Windows Forms-> SpreadWindowsFormsReadMe**. Select the Read Me under the Mescius name on the **Start** screen with Microsoft Windows 8, 8.1, or 10.
2. If you performed a default installation, in Windows Explorer browse to **Program Files (x86)\MESCIUS\Spread.NET [version]\Docs\readme.chm** file.

You can also access the [Read Me](#) on the web site.

Activating the Product

You can activate Spread for Winforms using online and offline methods listed below:

- Online Activation
 - Using MESCIUS License Manager
 - Using Visual Studio
 - Using Command Line Interface
- Offline Activation
 - Using MESCIUS License Manager
 - Using Command Line Interface

 **Note:** Contact sales at ["us.sales@mescius.com"](mailto:us.sales@mescius.com) to inquire about special license options for Azure DevOps or other similar build pipelines.

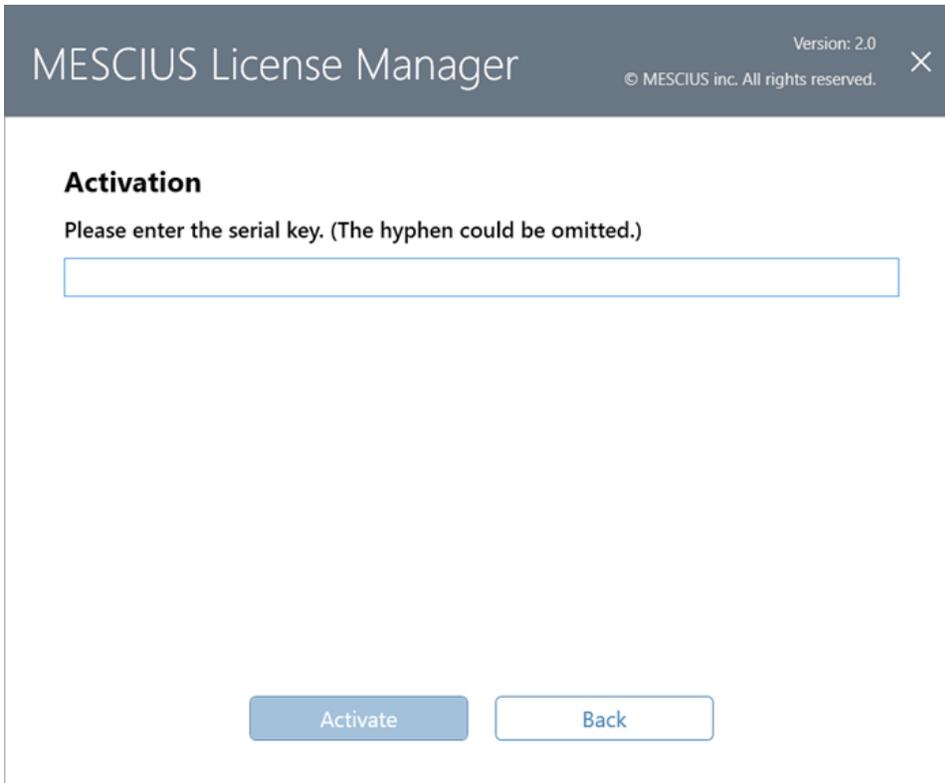
Online Activation

Using MESCIUS License Manager

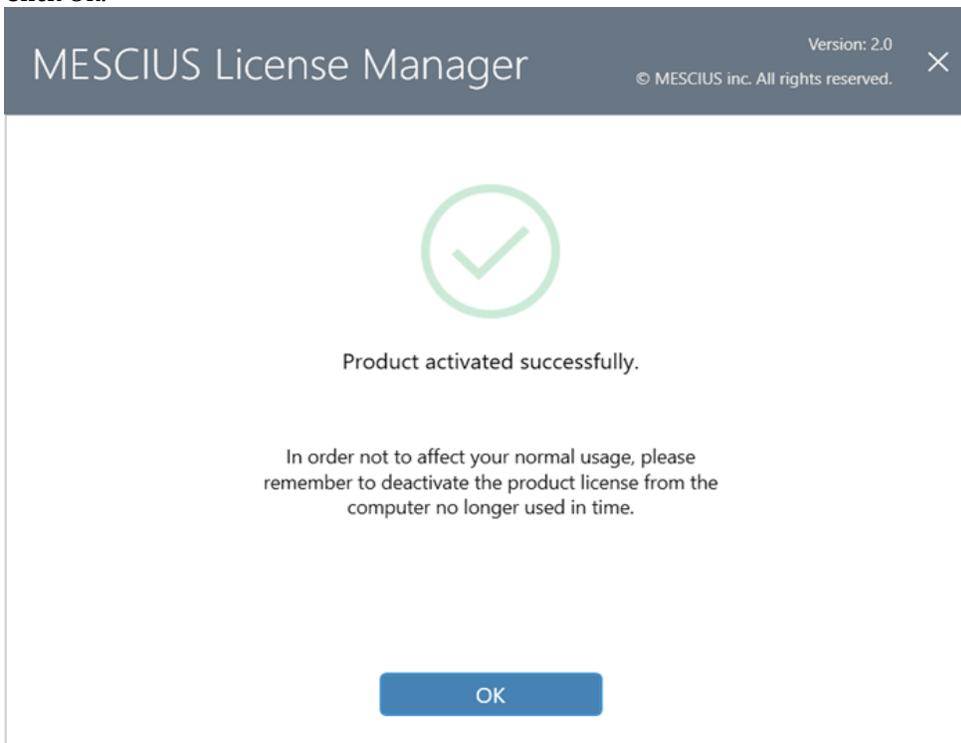
1. After installation, navigate to C:\ProgramData\MESCIUS\gclm on your machine and double click gclm.exe. Click 'Activate'.



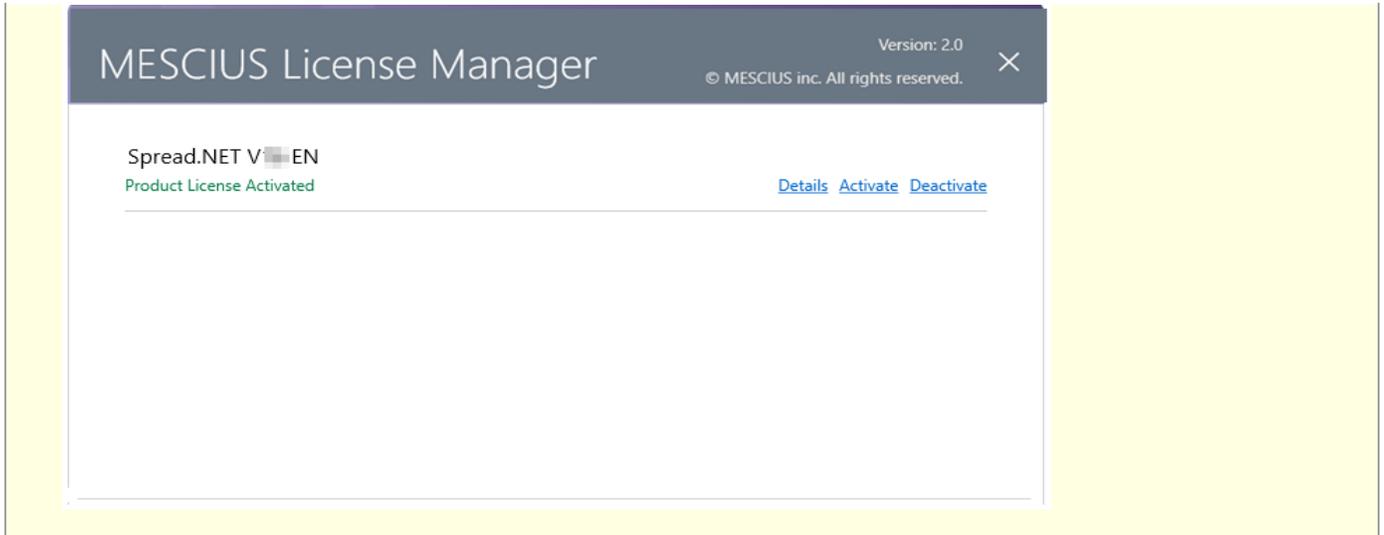
2. Enter the license key and click 'Activate'.



3. Click Ok.



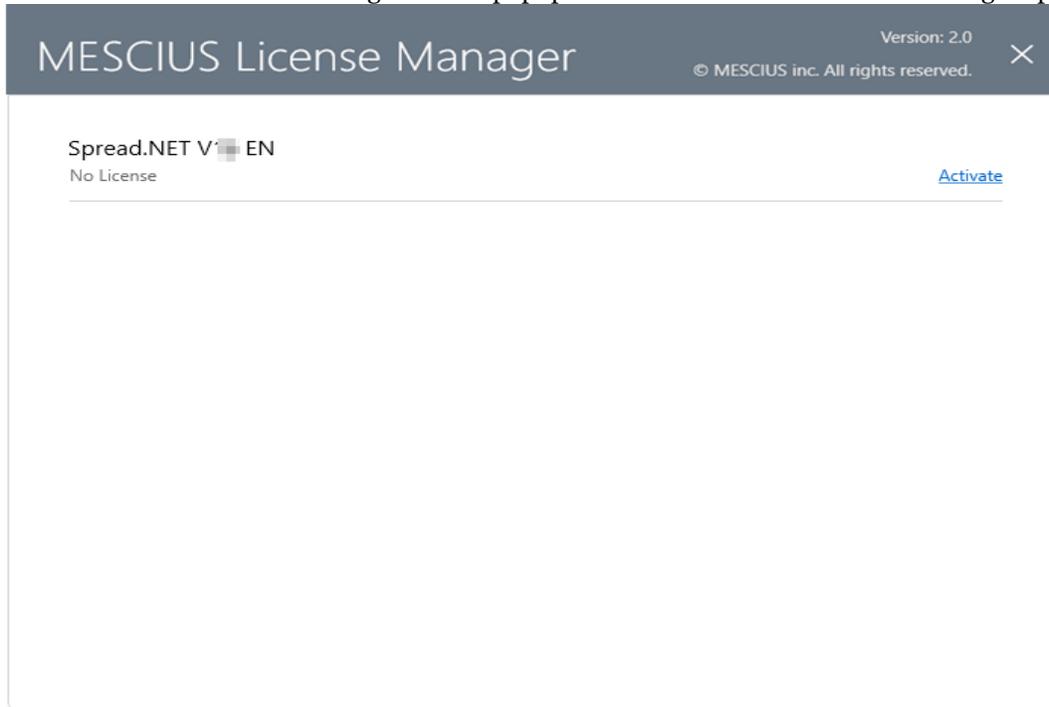
 **Note:** Use glm to deactivate the license before moving the license to another machine. Open glm located in C:\ProgramData\MESCIUS\glm and click 'Deactivate'.



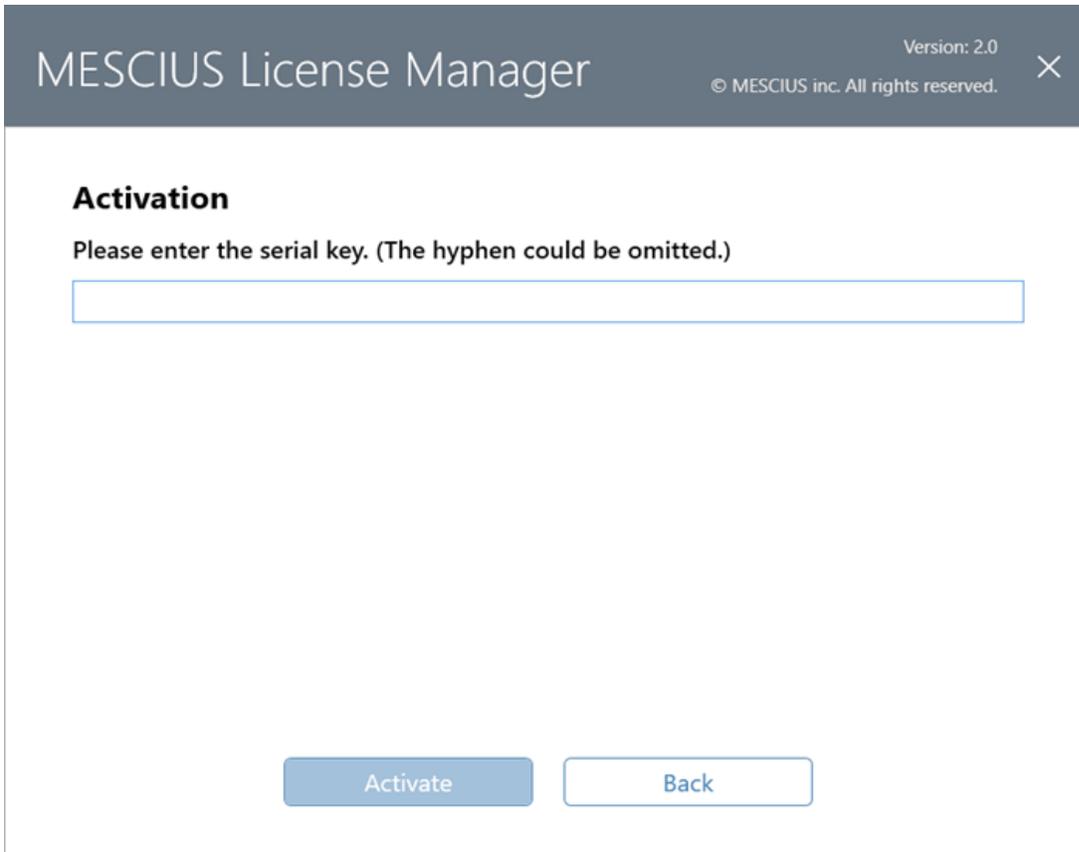
Using Visual Studio

If you are trying to activate the product package installed from the NuGet Package Manager in Visual Studio, follow the steps mentioned below:

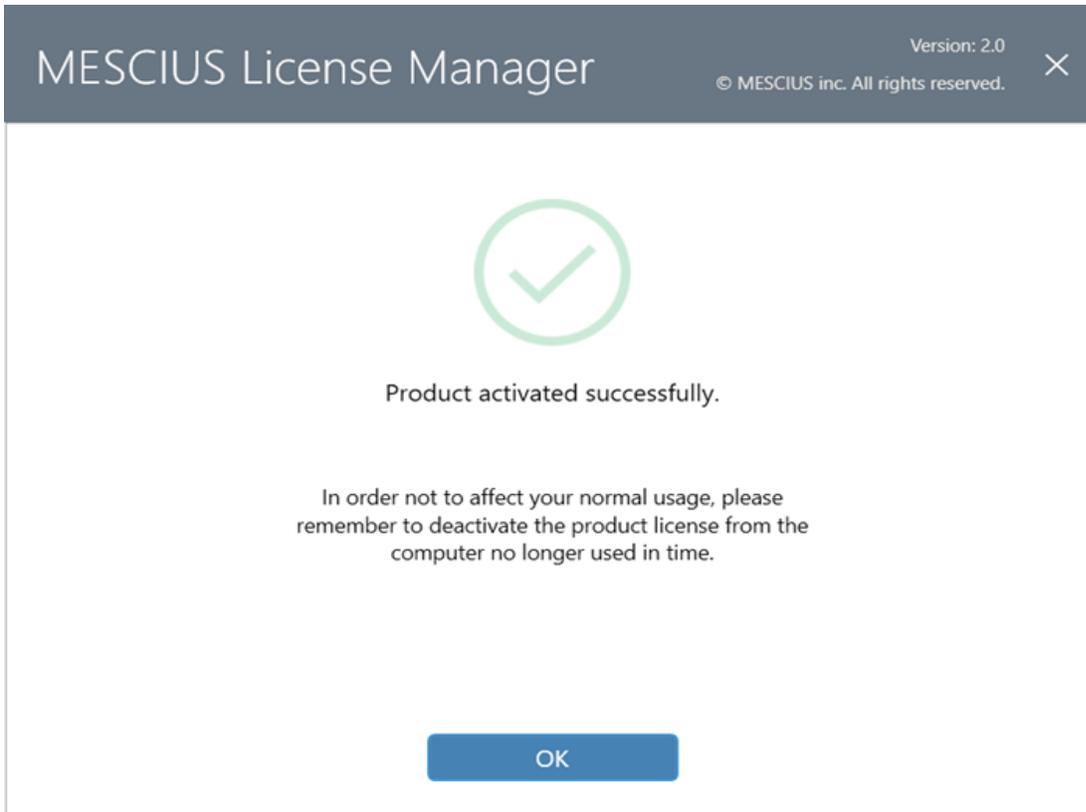
1. Create a project in Visual Studio.
2. Open NuGet Package Manager and install the product package from nuget.org
3. Build the project in Visual Studio. You will see a popup window during build.
4. Click on "Run the License Manager" in the popup window to launch the License Manager app.



5. Click the "Activate" link for the product in the window, and type your serial key.



6. After clicking the "Activate" button, License Manager App will connect to our website. If successful, you will see the result:



7. After closing the License Manager window, Visual Studio will continue the build.

Using Command Line Interface

You can activate the product using the command line interface if the graphics UI of the License Manager does not work normally. This is useful, especially for a build machine.

1. After installation, navigate to “C:\ProgramData\MESCIUS\gclm” in the command prompt.
2. Type the following command:

```
gclm.exe "product-id" -a "serial-key" > result.txt
```

3. Verify the activation in the generated “result.txt” file.

The following command line options are available for MESCIUS License Manager:

Command Option

gclm [product-guid]

gclm [product-guid] -a [serial key]

gclm [product-guid] -d

gclm [product-guid] -r

**gclm [product-guid] -g [serial key]
[output path]**

gclm [product-guid] -i [path of .license]

gclm [product-guid] -dx [output path]

Description

Output license status for the product

Activate license for the product

Deactivate license for the product

Remove license for the product without deactivation. For invalid license only.

Generate a .key file for offline activation

Import an offline activated license file

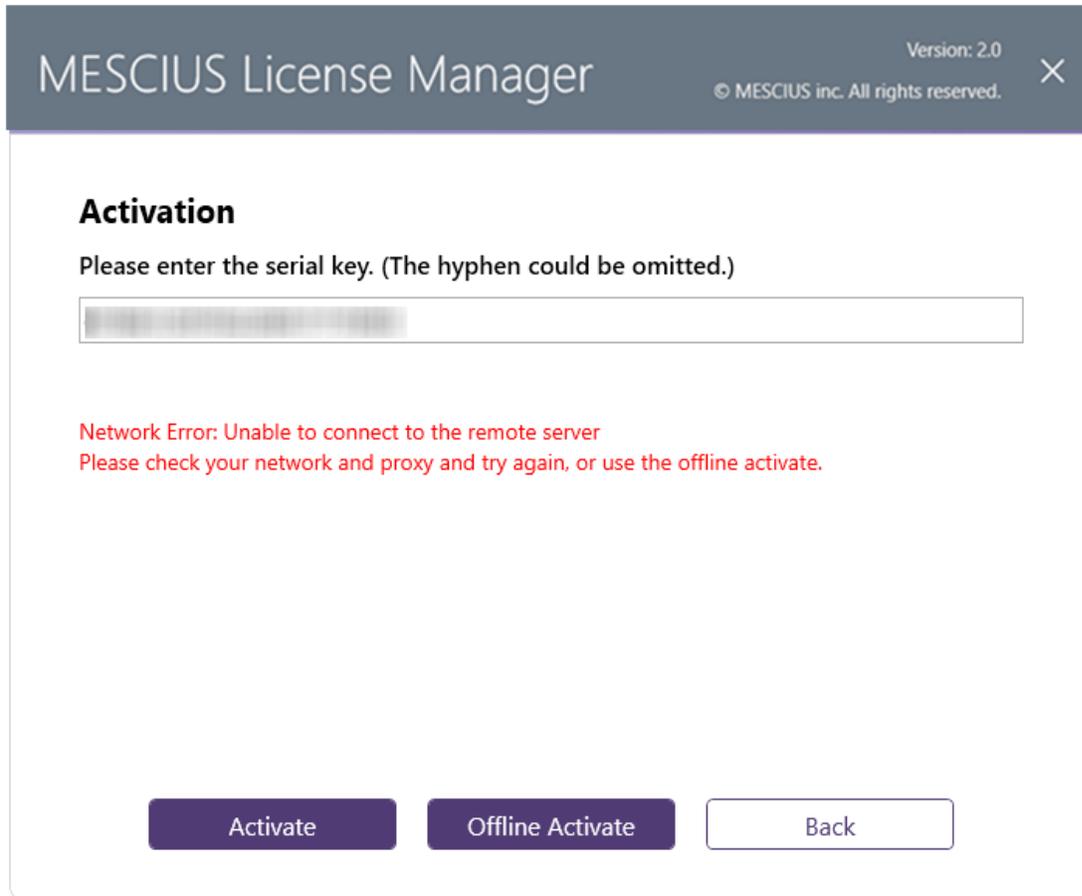
Remove the license and generate a .key file for offline deactivation.

Offline Activation

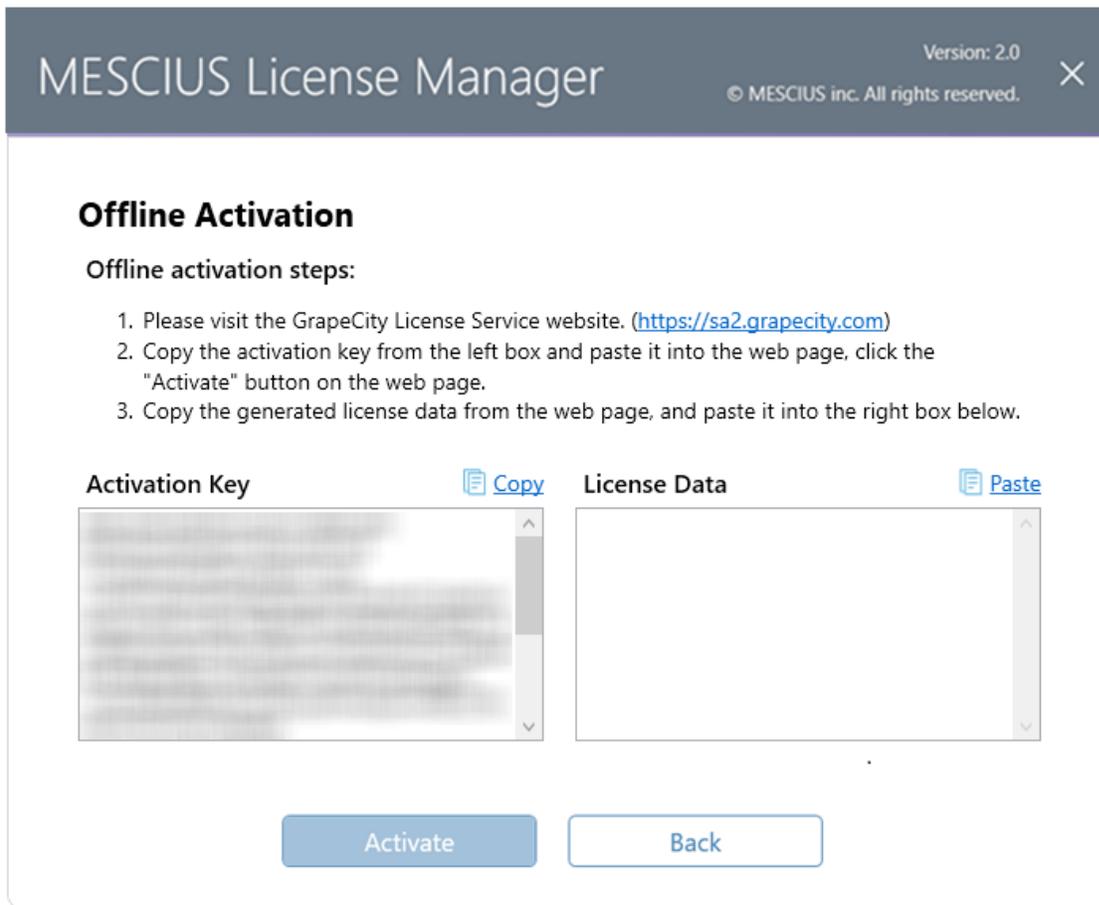
Suppose the target machine cannot access the internet, but there is another machine that can access the sa2 website via web browser. In that case, you can activate a normal developer license in offline mode.

Using MESCIUS License Manager

1. Run the gclm.exe from "C:\ProgramData\MESCIUS\gclm" and try to activate a serial key without network access. Our tool will display the following error on the window.



2. Click on the "Offline Activate" button to access the offline activation window.



3. Copy the contents of "Activation Key" to another machine which could access the sa2 site.
4. On that machine, access <https://sa2.grapecity.com/activate> and paste the copied Activation key.
5. Click "Activate" button to generate the license data.
6. Save the license data from the webpage, and copy them back to the original machine.
7. Paste the license data into the offline activation page of the gclm.exe, then the license is activated.

Using Command Line Interface

1. Navigate to "C:\ProgramData\MESCIUS\gclm" in the command prompt.
2. Type the following command:

```
gclm.exe "product-id" -g "serial-key" "output-path"
```

It generates a text file that contains an activation key.

3. Copy the contents of the activation key text file to another machine that could access the sa2 site.
4. On that machine, access <https://sa2.grapecity.com/activate> and paste the copied Activation key.
5. Click the "Activation" button to generate the license data.
6. Copy the license data to another text file.
7. Type the following command:

```
gclm.exe "product-id" -i "license-file-name"
```

It imports the offline activated license.

 **Note:** For both offline methods, the deactivation steps are similar to activation. So, you need to copy the "Deactivation Key" to the sa2 site to release the usage of the activation key.

End-User License Agreement

The Mescius licensing information, including the Mescius end-user license agreements, frequently asked licensing questions, and the Mescius licensing model, is available online at <https://developer.mescius.com/spread/licensing> and <https://developer.mescius.com/legal/eula>.

Creating a Runtime License

When applications are built on a properly activated development or build machine, the Mescius license file generates automatically and embeds in your application.

In some scenarios, you may need to manually generate this file for an application, such as a user control or class library. And every application requires a unique runtime license file (.gclix).

To create a runtime license for an application, follow these steps by the command line.

1. Navigate to "C:\ProgramData\GrapeCity\gclm" directory in the command prompt.
2. Type the following command:

```
gclm.exe 4e51d0df-dea4-48be-920b-a02502bd9882 -lc ../gclix app-name  
where "app-name" refers to the application name.
```

The .gclix file is generated in the "gclm" folder. Add the generated .gclix file in your project and set its **BuildAction** to **EmbeddedResource**.

Handling Redistribution

When you deploy applications that you have developed using Spread Windows Forms, your users' systems must meet the following requirements and you must distribute the files listed in the following sections:

System Requirements

Your users' systems must meet the following requirements:

Operating System

Must be one of the following:

- Microsoft Windows Server 2008
- Microsoft Windows Server 2008 R2
- Microsoft Windows Server 2012
- Microsoft Windows Server 2012 R2
- Microsoft Windows Server 2016
- Microsoft Windows Server 2019
- Microsoft Windows Server 2022
- Microsoft Windows 7
- Microsoft Windows 8.1
- Microsoft Windows 10
- Microsoft Windows 11

Software

You must have the Microsoft .NET Framework 4.6.2 or later installed.

Files to Distribute

You must distribute the following files to your users' systems:

- The following assemblies that come with Spread Windows Forms:
 - GrapeCity.CalcEngine.dll
 - GrapeCity.Spreadsheet.dll
 - GrapeCity.SpreadSheet.Win.dll
 - FarPoint.CalcEngine.dll
 - FarPoint.Excel.dll
 - FarPoint.PluginCalendar.WinForms.dll
 - FarPoint.Win.dll
 - FarPoint.Win.Spread.dll

Installation for your application must copy these DLLs from the Spread Windows Forms directory to the directory where the application's executable file resides or install them in the global assembly cache (GAC). For more information on the GAC, refer to the Microsoft Visual Studio .NET and .NET Framework documentation.

- The .NET Framework redistributable package, if the users do not have the .NET Framework on their systems. For more information on this package, refer to the .NET Framework documentation.
- If you use the ink notation feature in your project then you will also need to distribute the FarPoint.Win.Ink.dll. This DLL would need to be installed to the directory where the application's executable file resides or be installed in the global assembly cache (GAC). This also requires the runtime components of the Microsoft Tablet PC SDK. The FarPoint.Win.Ink assembly is currently built with version 1.7 of the Microsoft Tablet PC SDK.
- If you use the text renderer feature in your project then you will also need to distribute the FarPoint.Win.TextRenderer.dll. This DLL would need to be installed to the directory where the application's executable file resides.
- If you use the export to PDF feature in your project then you will also need to distribute the FarPoint.PDF.dll.
- If you use the export to HTML feature in your project then you will also need to distribute the FarPoint.Win.Spread.Html.dll and the System.Web.dll.
- If you use a Spread designer dialog at run time then you also need to distribute the FarPoint.Win.Spread.Design.dll.
- If you use the chart control in your project then you need to distribute the FarPoint.Win.Chart.dll.

Hosting the Control on a Web Page

If you are hosting the Spread Windows Forms control as a user control on a Web page in Microsoft Internet Explorer (IE), make these security permission adjustments:

1. In IE, select **Tools->Internet Options->Security and select Trusted Sites**. Click the **Sites** button and add the Web site where your user control resides (for example, <http://localhost>).
2. In Windows, select **Start->Settings->Control Panel** and select **Administrative Tools**. Select **Microsoft .NET Framework Configuration**. In the .NET Framework Configuration window, select **Runtime Security Policy** and click **Adjust Zone Security**. In the **Adjust Zone Security Wizard**, answer the first screen (which computer it applies to) and in the next screen, click **Trusted Sites** and slide the indicator to give that zone **Full Trust**. Finish the wizard by clicking **Next**.

Product Requirements

Developing applications with the .NET 6 version of Spread Windows Forms

For developing applications with the .NET 6 version of Spread Windows Forms, you must have the following system and software specifications:

Operating System for Installing .NET 6 on Windows

One of the following:

- Microsoft Windows Server 2012 R2+
- Microsoft Windows Server 2016
- Microsoft Windows Server 2019
- Microsoft Windows Server 2022
- Microsoft Windows 7 SP1+
- Microsoft Windows 8.1
- Microsoft Windows 10 version 1607+
- Microsoft Windows 11

Software

Release version of the Microsoft .NET 6

These are minimum requirements to run the product.

.NET 6 do not support Ink capabilities.

Developing applications with the .NET 4.6.2 version of Spread Windows Forms

For developing applications with the .NET 4.6.2 version of Spread Windows Forms, you must have the following system and software specifications:

Operating System

One of the following:

- Microsoft Windows Server 2008
- Microsoft Windows Server 2008 R2
- Microsoft Windows Server 2012
- Microsoft Windows Server 2012 R2
- Microsoft Windows Server 2016
- Microsoft Windows Server 2019
- Microsoft Windows Server 2022
- Microsoft Windows 7
- Microsoft Windows 8.1
- Microsoft Windows 10
- Microsoft Windows 11

Software

The release version of the Microsoft .NET 4.6.2 Framework or later.

These are the minimum requirements to run the product.

If you want to take advantage of the ink capabilities of Spread Windows Forms, you will need to install the runtime components of the Microsoft Tablet PC SDK. The FarPoint.Win.Ink assembly is currently built with version 1.7 of the

Microsoft Tablet PC SDK.

Using Windows Regional Settings or Options

The Spread component reads the Windows regional settings or options, which are set by the user through the Control Panel, but due to variations in how Windows handles those settings, your user might experience unexpected results.

In general in Windows operating systems, the Spread component does not recognize changes made to the Windows regional settings until you restart your development environment or your application or perform any operation that unloads and reloads the current assembly and dependent assemblies. This is because handling the regional settings is very processor intensive. To optimize performance these settings are not checked each time a simple operation is performed.

In most Windows operating systems, the regional options are read from the system registry. In certain situations, Windows does not clear previous regional options when reading changes from the system registry. Be aware of this when working with regional settings.

Using Satellite Assemblies for Languages

You can place resources for different languages using satellite assemblies. The assembly is then loaded in memory if the user views the application in that language. The resources must be placed in specific locations so they can be located and used. If the resource cannot be found, the default resource is used.

Use the following steps to add a language resource:

1. Find the resources in the localization folder under the installed bin folder (for example, ko-KR or zh-CN).
2. Copy the folder to the bin folder of the application or install to the GAC.
3. Set the current UI culture to the language (for example, Korean) using the following code.

```
System.Threading.Thread.CurrentThread.CurrentUICulture = new CultureInfo("ko-KR")
```

If you wish to use the resource at design time, install the satellite assemblies to the GAC and select the language in Visual Studio.NET.



The stand-alone designer uses the resources in the GAC if the operating system and the GAC resources use the same language. If the language is different, the default resource is applied (English).

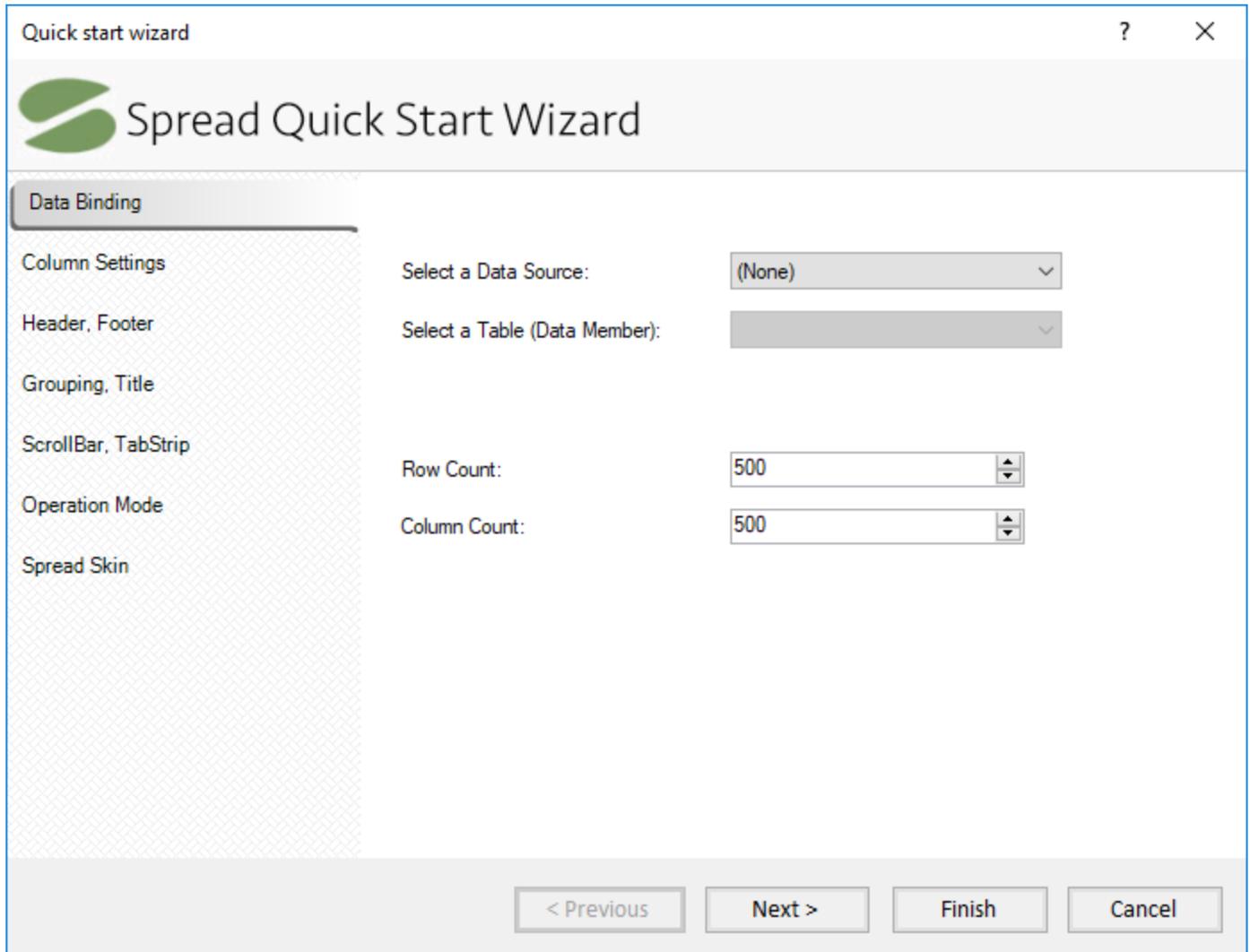
Understanding the Spread Wizard

You can use the Spread Wizard to quickly and easily bind data, set up the column structure, and customize the appearance of a spreadsheet. See the following topics for more information:

- **Starting the Spread Wizard**
- **Using the Spread Wizard**

Starting the Spread Wizard

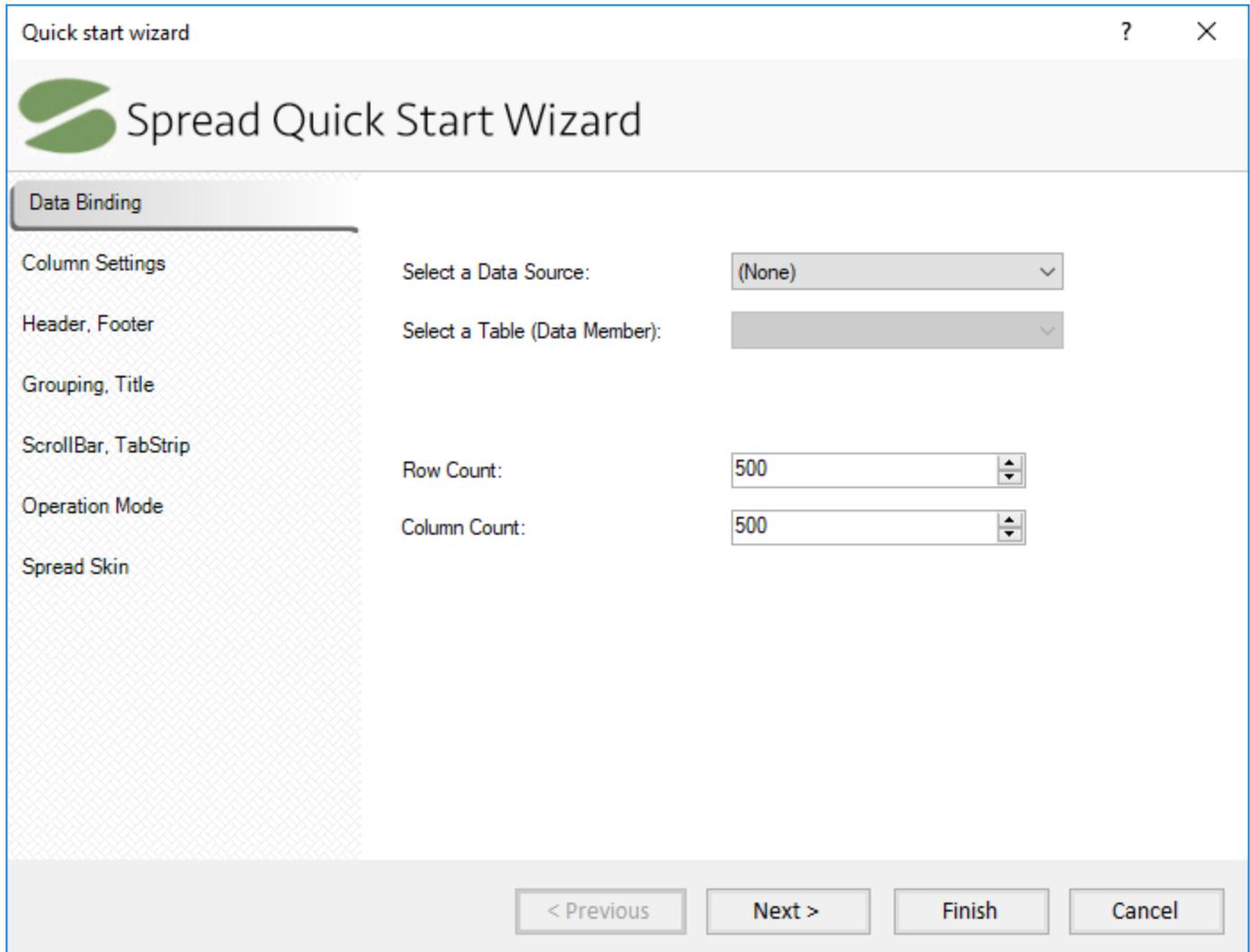
You can launch the Spread Wizard from the Smart Tags on the FpSpread component on the form in Visual Studio as shown in this figure. The smart tag is the arrow icon at the top, right edge of the control.



Using the Spread Wizard

You can use the Spread Wizard to bind to a data source, set column properties, set the operation mode, specify titles, select a skin, and perform many other tasks.

Select the menu option of the feature you wish to customize, located on the left side of the dialog. Select the various options for that feature and then click **Next** to go to the next step. When you are finished, click **Finish**.



Getting More Practice

If you need more tips about customizing Spread and taking advantage of its many features, we have provided these additional sources of information to help you get started.

- **Finding the Documentation**
- **Getting Technical Support (on-line documentation)**

Finding the Documentation

There are several different ways to accomplish the same result when creating a Windows Forms page with a Spread component. In this documentation, the procedures often describe more than one way, including using the **Properties** window in Visual Studio .NET, writing code including using shortcut objects, and using the Spread Designer. The Spread Designer sets properties and calls methods for the Spread component, including properties not available at design time through Visual Studio .NET, without producing any editable code.

Each of these has its advantages and disadvantages. Using shortcut objects is the shortest, quickest way of adding code using dot notation and setting a property of a shortcut object. Using code without using shortcut objects generally means declaring objects and setting properties for them.

Documentation Provided

The Spread Windows Forms documentation provides introductory information about the product, conceptual information, how-to topics, and a detailed assembly and formula function reference in a help file and in PDF files. Additional information is provided in the Read me file.

Accessing the Help

You can access the help through F1 support provided in Visual Studio .NET. While the Spread component or one of its members has focus, press F1 to display the Spread Windows Forms help.

You can also access the help file in a stand-alone window by choosing **Programs->Mescius->...->Product Name** and then selecting the help.

Documentation Conventions

The format of the help is similar to the help provided for Visual Studio .NET. Reference material for members provides multiple language reference for the member. You can change which language's syntax is displayed by clicking the Languages button in the title of the topic.

List of How-To's

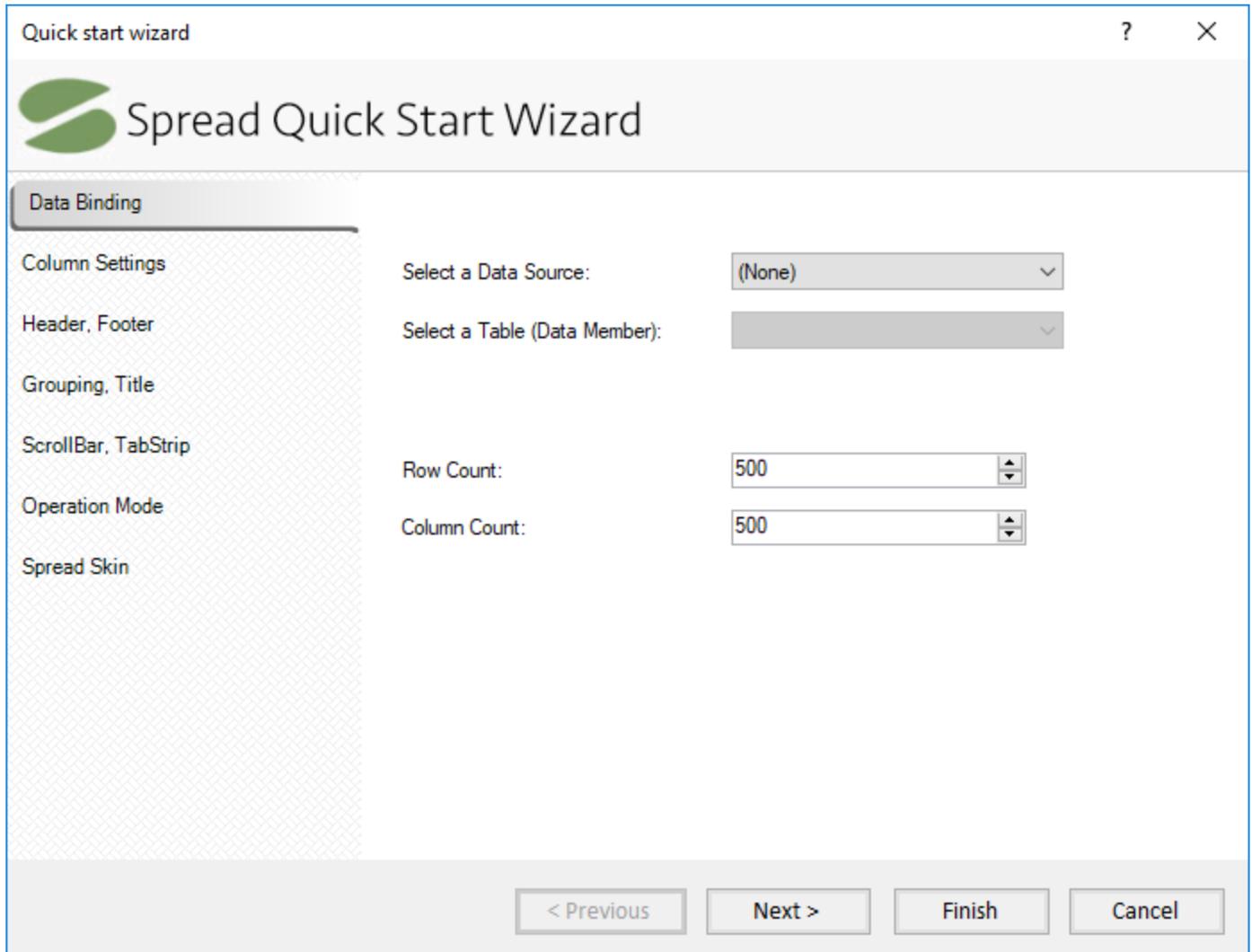
Here is a list of the How To's:

- **Adding a Note to a Cell**
- **Adding a Row or Column**
- **Adding a Sheet**
- **Allowing the User to Enter Formulas**
- **Allowing the User to Automatically Sort Rows**
- **Allowing the User to Perform a Standard Search**
- **Applying a Skin to a Sheet**
- **Creating a Custom Skin for a Sheet**
- **Creating and Using a Custom Function**
- **Creating and Using a Custom Name**
- **Creating Alternating Rows**
- **Customizing the Outline of the Component**
- **Customizing the Input Maps**
- **Customizing the Scroll Bars**
- **Customizing Split Boxes**
- **Customizing the Dimensions of the Component**
- **Customizing the Number of Rows or Columns**
- **Customizing Viewports**
- **Customizing the Selection Appearance**
- **Customizing the Sheet Corner Appearance**
- **Displaying Grid Lines on a Sheet**
- **Working with Multiple Sheets**
- **Displaying Text Tips in a Cell**
- **Locking a Cell**
- **Nesting Functions in a Formula**
- **Opening Existing Files**

- **Optimizing the Printing Using Rules**
- **Using Automatic Sorting**
- **Placing a Formula in Cells**
- **Placing Child Controls on a Sheet**
- **Printing an Entire Sheet**
- **Printing Particular Pages**
- **Printing a Range of Cells on a Sheet**
- **Printing an Entire Sheet**
- **Providing a Preview of the Printing**
- **Removing a Row or Column**
- **Removing a Sheet**
- **Saving Data to a File**
- **Searching for Data with Code**
- **Setting the Background Colors for a Sheet**
- **Setting the Row Height or Column Width**
- **Sorting Rows, Columns, or Ranges**
- **Specifying a Cell Reference in a Formula**
- **Specifying What the User Can Select**
- **Using a Circular Reference in a Formula**
- **Using Drag Operations to Fill Cells**
- **Working with Editable Cell Types**
- **Working with Graphical Cell Types**
- **Working with Selections**

Starting the Spread Wizard

You can launch the Spread Wizard from the Smart Tags on the FpSpread component on the form in Visual Studio as shown in this figure. The smart tag is the arrow icon at the top, right edge of the control.



Tutorial: Creating a Checkbook Register

The following tutorial walks you through creating a project in Visual Studio .NET using the Spread Windows Forms component. By creating a checkbook register, you will learn how to modify the appearance of a spreadsheet, work with cell types, and add some formulas for performing calculations.

In this tutorial, the major steps are:

- **Adding Spread to the Checkbook Project**
- **Setting Up the Rows and Columns of the Register**
- **Setting the Cell Types of the Register**
- **Adding Formulas to Calculate Balances**

Adding Spread to the Checkbook Project

Use the following steps to add Spread to the project.

1. Start a new Visual Studio .NET project.

2. Name the project **checkbook**. Name the form in the project **register**.
3. Add the FpSpread component to your project, and then place the control on the form.

If you do not know how to add the FpSpread component to the project, complete the steps in **Adding a Component to a Project**.

Go to **Setting Up the Rows and Columns of the Register** to continue the tutorial.

Setting Up the Rows and Columns of the Register

The Spread control on your form already has a sheet, ready for you to configure. In this step, you are going to set up the columns and cells in the sheet to resemble a checkbook register.

Using Code

1. Double-click on the form in your project to open the code window.
2. In the Form Load event, type the following code:
This code sets up the control to be 300 pixels high and 763 pixels wide, and the sheet to have 8 columns and 100 rows.

Example

C#

```
// Set up control and rows and columns in sheet.  
fpSpread1.Height = 330;  
fpSpread1.Width = 765;  
fpSpread1.Sheets[0].ColumnCount = 8;  
fpSpread1.Sheets[0].RowCount = 100;
```

VB

```
' Set up control and rows and columns in sheet.  
fpSpread1.Height = 330  
fpSpread1.Width = 765  
fpSpread1.Sheets(0).ColumnCount = 8  
fpSpread1.Sheets(0).RowCount = 100
```

3. Next set up the columns to add custom headings. Add the following code below the code you added in Step 2:

Example

C#

```
// Add text to column heading.  
fpSpread1.Sheets[0].ColumnHeader.Cells[0, 0].Text = "Check #";  
fpSpread1.Sheets[0].ColumnHeader.Cells[0, 1].Text = "Date";  
fpSpread1.Sheets[0].ColumnHeader.Cells[0, 2].Text = "Description";  
fpSpread1.Sheets[0].ColumnHeader.Cells[0, 3].Text = "Tax?";  
fpSpread1.Sheets[0].ColumnHeader.Cells[0, 4].Text = "Cleared?";  
fpSpread1.Sheets[0].ColumnHeader.Cells[0, 5].Text = "Debit";  
fpSpread1.Sheets[0].ColumnHeader.Cells[0, 6].Text = "Credit";  
fpSpread1.Sheets[0].ColumnHeader.Cells[0, 7].Text = "Balance";
```

VB

```
' Add text to column heading.
fpSpread1.Sheets(0).ColumnHeader.Cells(0, 0).Text = "Check #"
fpSpread1.Sheets(0).ColumnHeader.Cells(0, 1).Text = "Date"
fpSpread1.Sheets(0).ColumnHeader.Cells(0, 2).Text = "Description"
fpSpread1.Sheets(0).ColumnHeader.Cells(0, 3).Text = "Tax?"
fpSpread1.Sheets(0).ColumnHeader.Cells(0, 4).Text = "Cleared?"
fpSpread1.Sheets(0).ColumnHeader.Cells(0, 5).Text = "Debit"
fpSpread1.Sheets(0).ColumnHeader.Cells(0, 6).Text = "Credit"
fpSpread1.Sheets(0).ColumnHeader.Cells(0, 7).Text = "Balance"
```

- Now set up the column widths to properly display the headings and the data you will add. Add the following code below the code you added in Step 3:

Example

C#

```
// Set column widths.
fpSpread1.Sheets[0].Columns[0].Width = 50;
fpSpread1.Sheets[0].Columns[1].Width = 50;
fpSpread1.Sheets[0].Columns[2].Width = 175;
fpSpread1.Sheets[0].Columns[3].Width = 40;
fpSpread1.Sheets[0].Columns[4].Width = 65;
fpSpread1.Sheets[0].Columns[5].Width = 100;
fpSpread1.Sheets[0].Columns[6].Width = 100;
fpSpread1.Sheets[0].Columns[7].Width = 125;
```

VB

```
' Set column widths.
fpSpread1.Sheets(0).Columns(0).Width = 50
fpSpread1.Sheets(0).Columns(1).Width = 50
fpSpread1.Sheets(0).Columns(2).Width = 175
fpSpread1.Sheets(0).Columns(3).Width = 40
fpSpread1.Sheets(0).Columns(4).Width = 65
fpSpread1.Sheets(0).Columns(5).Width = 100
fpSpread1.Sheets(0).Columns(6).Width = 100
fpSpread1.Sheets(0).Columns(7).Width = 125
```

- Save your project, then click the **Start** button in the toolbar to run your project.
- Your form should look similar to the following picture.

	Check #	Date	Description	Tax?	Cleared?	Debit	Credit	Balance
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								
11								
12								
13								
14								
15								
16								
17								

Go to **Setting the Cell Types of the Register** to continue the tutorial.

Setting the Cell Types of the Register

To set cell types, for each custom cell type, you have to create a cell type object, set the properties for it, and then assign that object to the **CellType** ('CellType Property' in the on-line documentation) property for a cell or range of cells.

1. Set the cell type for the Check # column by adding the following code below the code you have already added:

Example

C#

```
// Create Check # column of number cells.
FarPoint.Win.Spread.CellType.NumberCellType objNumCell = new
FarPoint.Win.Spread.CellType.NumberCellType();
objNumCell.DecimalPlaces = 0;
objNumCell.MinimumValue = 1;
objNumCell.MaximumValue = 9999;
objNumCell.ShowSeparator = false;
fpSpread1.Sheets[0].Columns[0].CellType = objNumCell;
```

VB

```
' Create Check # column of number cells.
Dim objNumCell As New FarPoint.Win.Spread.CellType.NumberCellType()
objNumCell.DecimalPlaces = 0
objNumCell.MinimumValue = 1
objNumCell.MaximumValue = 9999
objNumCell.ShowSeparator = False
fpSpread1.Sheets(0).Columns(0).CellType = objNumCell
```

2. Set the cell type for the Date column by adding the following code below the code you have already added:

Example

C#

```
// Create Date column of date-time cells.
FarPoint.Win.Spread.CellType.DateTimeCellType objDateCell = new
FarPoint.Win.Spread.CellType.DateTimeCellType();
objDateCell.DateTimeFormat =
FarPoint.Win.Spread.CellType.DateTimeFormat.ShortDate;
fpSpread1.Sheets[0].Columns[1].CellType = objDateCell;
```

VB

```
' Create Date column of date-time cells.
Dim objDateCell As New FarPoint.Win.Spread.CellType.DateTimeCellType()
objDateCell.DateTimeFormat = FarPoint.Win.Spread.CellType.DateTimeFormat.ShortDate
fpSpread1.Sheets(0).Columns(1).CellType = objDateCell
```

- Set the cell type for the Description column by adding the following code below the code you have already added:

Example**C#**

```
// Create Description column of text cells.
FarPoint.Win.Spread.CellType.TextCellType objTextCell = new
FarPoint.Win.Spread.CellType.TextCellType();
objTextCell.MaxLength = 100;
fpSpread1.Sheets[0].Columns[2].CellType = objTextCell;
```

VB

```
' Create Description column of text cells.
Dim objTextCell As New FarPoint.Win.Spread.CellType.TextCellType()
objTextCell.MaxLength = 100
fpSpread1.Sheets(0).Columns(2).CellType = objTextCell
```

- Set the cell type for the Tax? and Cleared? columns by adding the following code below the code you have already added:

Example**C#**

```
/// Create Tax? and Cleared? columns of check box cells.
FarPoint.Win.Spread.CellType.CheckBoxCellType objCheckCell = new
FarPoint.Win.Spread.CellType.CheckBoxCellType();
objCheckCell.ThreeState = false;
fpSpread1.Sheets[0].Columns[3].CellType = objCheckCell;
fpSpread1.Sheets[0].Columns[4].CellType = objCheckCell;
```

VB

```
' Create Tax? and Cleared? columns of check box cells.
Dim objCheckCell As New FarPoint.Win.Spread.CellType.CheckBoxCellType()
objCheckCell.ThreeState = False
fpSpread1.Sheets(0).Columns(3).CellType = objCheckCell
fpSpread1.Sheets(0).Columns(4).CellType = objCheckCell
```

- Set the cell type for the Debit, Credit, and Balance columns by adding the following code below the code you have already added:

Example

C#

```
// Create the Debit, Credit, and Balance columns of currency cells.
FarPoint.Win.Spread.CellType.CurrencyCellType objCurrCell = new
FarPoint.Win.Spread.CellType.CurrencyCellType();
objCurrCell.LeadingZero = FarPoint.Win.Spread.CellType.LeadingZero.Yes;
objCurrCell.NegativeRed = true;
objCurrCell.FixedPoint = true;
fpSpread1.Sheets[0].Columns[5].CellType = objCurrCell;
fpSpread1.Sheets[0].Columns[6].CellType = objCurrCell;
fpSpread1.Sheets[0].Columns[7].CellType = objCurrCell;
```

VB

```
' Create the Debit, Credit, and Balance columns of currency cells.
Dim objCurrCell As New FarPoint.Win.Spread.CellType.CurrencyCellType()
objCurrCell.LeadingZero = FarPoint.Win.Spread.CellType.LeadingZero.Yes
objCurrCell.NegativeRed = True
objCurrCell.FixedPoint = True
fpSpread1.Sheets(0).Columns(5).CellType = objCurrCell
fpSpread1.Sheets(0).Columns(6).CellType = objCurrCell
fpSpread1.Sheets(0).Columns(7).CellType = objCurrCell
```

- Save your project, then click the **Start** button in the toolbar to run your project.
- Your form should look similar to the following picture.

Check #	Date	Description	Tax?	Cleared?	Debit	Credit	Balance
1			<input type="checkbox"/>	<input type="checkbox"/>			
2			<input type="checkbox"/>	<input type="checkbox"/>			
3			<input type="checkbox"/>	<input type="checkbox"/>			
4			<input type="checkbox"/>	<input type="checkbox"/>			
5			<input type="checkbox"/>	<input type="checkbox"/>			
6			<input type="checkbox"/>	<input type="checkbox"/>			
7			<input type="checkbox"/>	<input type="checkbox"/>			
8			<input type="checkbox"/>	<input type="checkbox"/>			
9			<input type="checkbox"/>	<input type="checkbox"/>			
10			<input type="checkbox"/>	<input type="checkbox"/>			
11			<input type="checkbox"/>	<input type="checkbox"/>			
12			<input type="checkbox"/>	<input type="checkbox"/>			
13			<input type="checkbox"/>	<input type="checkbox"/>			
14			<input type="checkbox"/>	<input type="checkbox"/>			
15			<input type="checkbox"/>	<input type="checkbox"/>			

Go to **Adding Formulas to Calculate Balances** to continue the tutorial.

Adding Formulas to Calculate Balances

Your checkbook register is now set up to look like a checkbook register; however, it does not balance the currency figures you enter in the register. This step sets up the formula for balancing the figures.

1. Provide the following code in the Form Load event after the code you have already added:

Example

C#

```
// Set formula for calculating balance.
fpSpread1.Sheets[0].ReferenceStyle = FarPoint.Win.Spread.Model.ReferenceStyle.R1C1;
int i;
for (i = 0; i <= fpSpread1.ActiveSheet.RowCount - 1; i++)
{
    if (i == 0)
        fpSpread1.Sheets[0].Cells[i, 7].Formula = "RC[-1] - RC[-2]" ;
    else
        fpSpread1.Sheets[0].Cells[i, 7].Formula = "RC[-1] - RC[-2] + R[-1]C";
}
```

VB

```
' Set formula for calculating balance.
fpSpread1.Sheets(0).ReferenceStyle = FarPoint.Win.Spread.Model.ReferenceStyle.R1C1
Dim i As Integer
For i = 0 To fpSpread1.ActiveSheet.RowCount - 1
    If i = 0 Then
        fpSpread1.Sheets(0).Cells(i, 7).Formula = "RC[-1] - RC[-2]"
    Else
        fpSpread1.Sheets(0).Cells(i, 7).Formula = "RC[-1]-RC[-2]+R[-1]C"
    End If
Next
```

2. Save your project, then click the **Start** button in the toolbar to run your project.
3. Your form should look similar to the following picture. Type data into your checkbook register to test it and see how it operates.

Check #	Date	Description	Tax?	Cleared?	Debit	Credit	Balance
1		10/30/2014	<input type="checkbox"/>	<input type="checkbox"/>	\$55.00	\$100.00	\$45.00
2		11/01/2014	<input type="checkbox"/>	<input type="checkbox"/>	\$55.00	\$100.00	\$90.00
3			<input type="checkbox"/>	<input type="checkbox"/>			\$90.00
4			<input type="checkbox"/>	<input type="checkbox"/>			\$90.00
5			<input type="checkbox"/>	<input type="checkbox"/>			\$90.00
6			<input type="checkbox"/>	<input type="checkbox"/>			\$90.00
7			<input type="checkbox"/>	<input type="checkbox"/>			\$90.00
8			<input type="checkbox"/>	<input type="checkbox"/>			\$90.00
9			<input type="checkbox"/>	<input type="checkbox"/>			\$90.00
10			<input type="checkbox"/>	<input type="checkbox"/>			\$90.00
11			<input type="checkbox"/>	<input type="checkbox"/>			\$90.00
12			<input type="checkbox"/>	<input type="checkbox"/>			\$90.00
13			<input type="checkbox"/>	<input type="checkbox"/>			\$90.00
14			<input type="checkbox"/>	<input type="checkbox"/>			\$90.00
15			<input type="checkbox"/>	<input type="checkbox"/>			\$90.00

4. You have created a checkbook register using Spread. You have completed this tutorial.

Review the list of steps for **Tutorial: Creating a Checkbook Register**.

Understanding the Product

The following sections explain some of the underlying concepts for this unique and powerful product.

- **Product Overview**
 - **Shortcut Objects**
 - **Underlying Models**
 - **Formatted versus Unformatted Data**
 - **Cell Types**
- **Key Features**

Product Overview

Spread Windows Forms is a comprehensive spreadsheet component for Windows Forms applications that combines grid capabilities, spreadsheet functionality, and includes the ability to bind to data sources. One component can handle upto 2 billion sheets and one sheet can handle upto 2 billion rows and columns. Cross-sheet referencing allows calculations to make use of data and formulas on a variety of sheets.

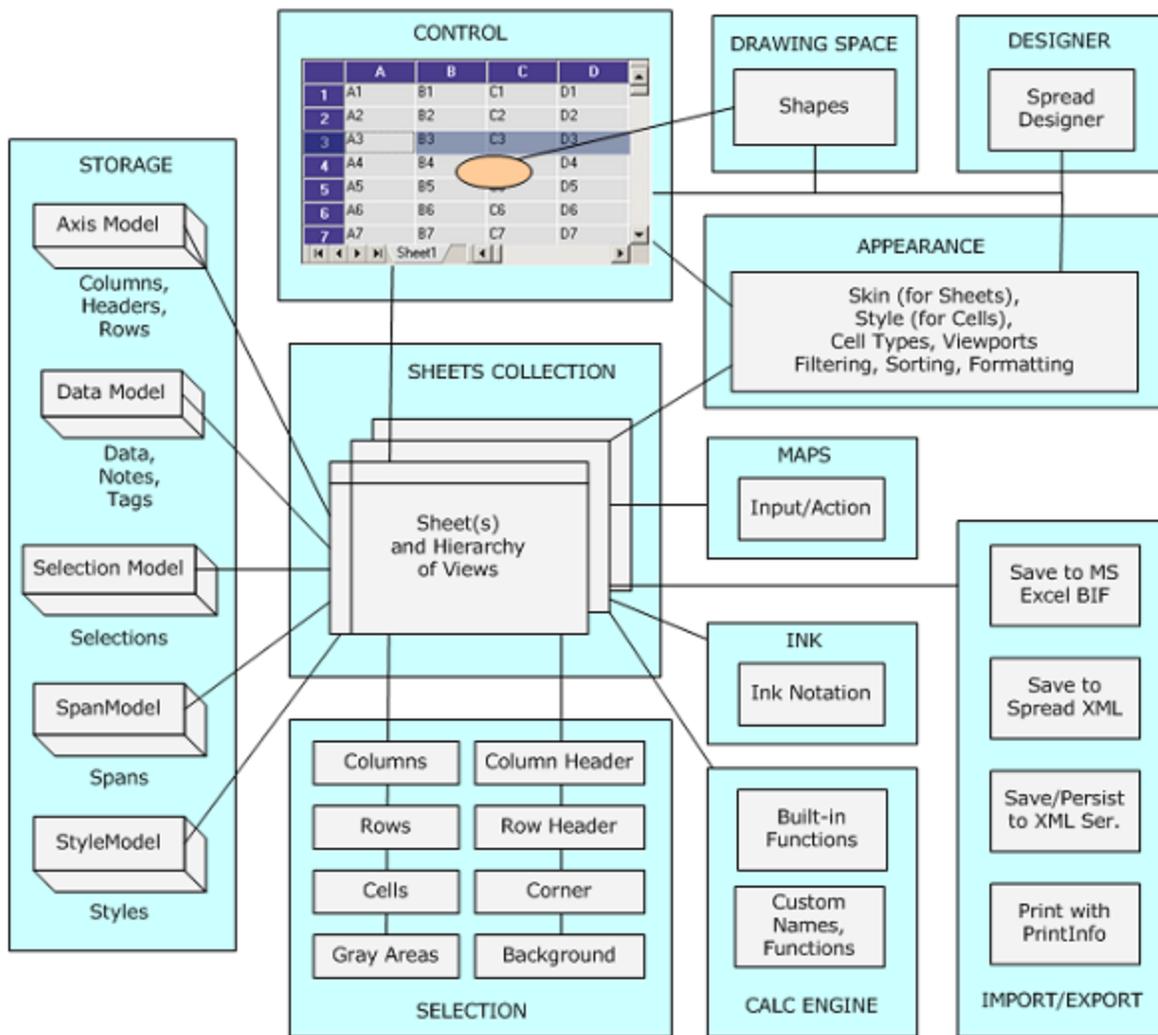
The Spread component includes toolbar and navigation feature with sheet collection. Sheets are formed from row, column, cell, and header objects.

You can use the included GUI designer to design grids, and intuitively set various appearance styles such as cell format, borders, background color, fonts, etc. Styles for cells and skins for sheets provide ways to save customized appearances that can be applied to other groups of cells or an entire sheet.

Spread Windows Forms can handle data from comma-delimited text files as well as multiple spreadsheets from Microsoft Excel files. The contents of a sheet may be saved as a BIFF8 or text file compatible with Microsoft Excel or as a Spread XML file.

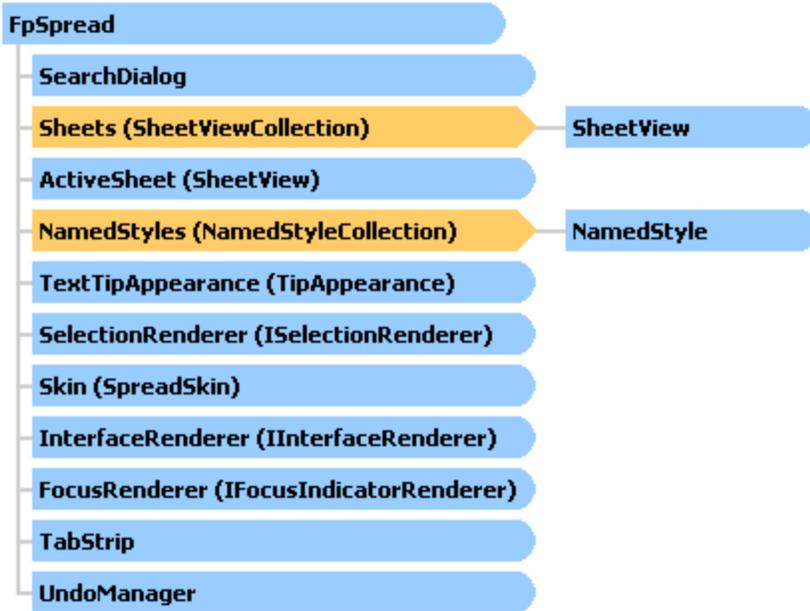
In Spread, you can use the default models or extend them through inheritance. For more information on models, refer to **Underlying Models**

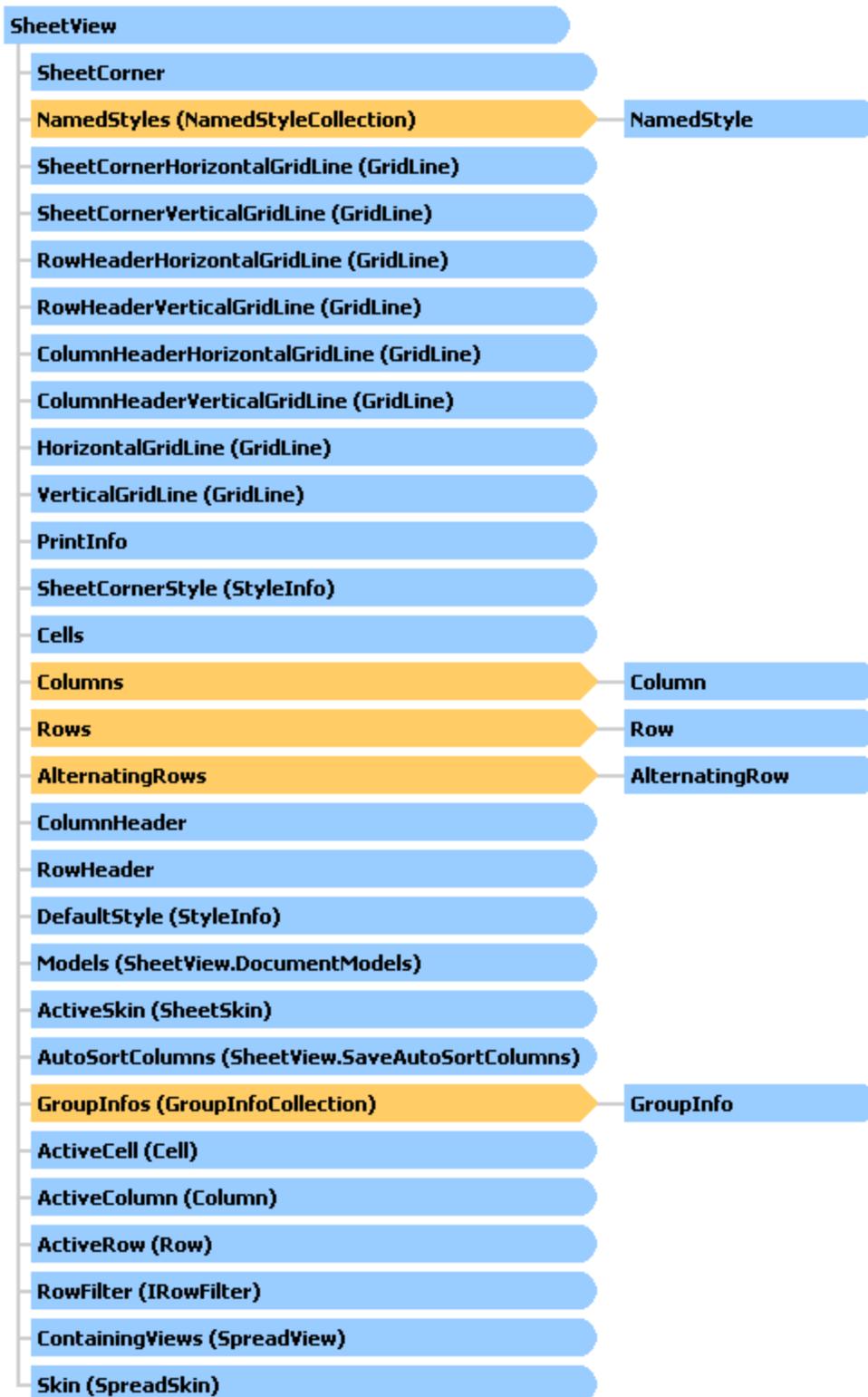
The following figure provides a conceptual overview of Spread Windows Forms.



Shortcut Objects

The spreadsheet objects in the FarPoint Spread namespace, which represent various parts of the spreadsheet, can be accessed through a built-in set of shortcut objects. Cells, rows, columns and others are wrappers to other objects, and make customization easier by allowing you to manipulate them. The shortcut objects represent parts of a visible spreadsheet, such as columns, rows, and cells; and there are conceptual representations of underlying pieces of the spreadsheet which are implemented in the underlying models. To understand more about the objects in Spread, look at the simplified object model diagrams for the **FpSpread ('FpSpread Class' in the on-line documentation)** class and the **SheetView ('SheetView Class' in the on-line documentation)** class as shown here.





The Spread Windows Forms component provides the following shortcut objects in the **FpSpread** ('FpSpread Class' **in the on-line documentation**) class:

Shortcut Corresponding Classes

Object

cell	Cell ('Cell Class' in the on-line documentation)	Cells ('Cells Class' in the on-line documentation)
column	Column ('Column Class' in the on-line documentation)	Columns ('Columns Class' in the on-line documentation)
header	ColumnHeader ('ColumnHeader Class' in the on-line documentation)	RowHeader ('RowHeader Class' in the on-line documentation)
row	Row ('Row Class' in the on-line documentation)	Rows ('Rows Class' in the on-line documentation)
alternating row	AlternatingRow ('AlternatingRow Class' in the on-line documentation)	AlternatingRows ('AlternatingRows Class' in the on-line documentation)
sheet	SheetView ('SheetView Class' in the on-line documentation)	SheetViewCollection ('SheetViewCollection Class' in the on-line documentation)

To use the shortcut objects, set their properties or call their methods. Many of the objects provide indexes for specifying the sheet, row, column, or cell with which you want to work.

Use the shortcut objects for their ease of use. The shortcut objects are fairly self-documenting; however, in Visual Studio .NET, the Intellisense feature provides additional information that help you use these objects.

Underlying Models

The Spread component provides the models that provide a basis for much of the customization that is possible with the component. The models are the underlying template from which the more commonly used shortcut objects are derived.

The shortcut objects access the underlying models. When you work with shortcut objects, you are actually working with the models in the component. For example, if you change the number of columns in a sheet using the Sheets shortcut object, the model for this (the default sheet axis model) is updated with that information.

To provide different features or customize the behavior or appearance of your application, you can extend the models to create new classes. For example you may do this to create a template component for all the developers in your organization. By creating your own class based on one of the models, you can create a customized class and provide it to all the developers to use.

Use the object models for the following benefits:

- For better performance: if you are setting several properties, for example, your application will be faster if you set the properties for an object, and then assign that object to Spread.
- For specialized features: if you want to create your own customized features, such as extending the data model to bring in a tab-delimited file, you can extend the BaseSheetDataModel to do so. If you want to create your own cell type or customize the behavior of how users select cells, you can do that through the models.
- For consistency in development: if you are a development team that would like to have consistency in some custom style and custom behavior, make the changes in the models and the entire team can benefit.
- For more complete understanding of the product: if you are using many of the features of the component, the most efficient way to customize the component is by first understanding the workings of the models upon which the objects are based.

For more information about the underlying models and how to use them, refer to **Understanding the Underlying Models**.

Formatted versus Unformatted Data

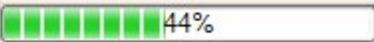
The Spread Windows Forms component provides both text (formatted data) and value (unformatted data) properties for a cell. For example, in a currency cell, the formatted data could be \$1,432.56, but the value would be 1432.56. The Text

property of the cell would return the entire formatted string with currency symbol and thousand separator. The Value property of the cell could be used in formulas or other calculations. Every cell has both properties. Depending on the cell type, the data in a cell may be handled differently. For cell types that have buttons or check boxes, the distinction is important.

For more detailed information on the difference between formatted and unformatted data, and a summary of the results for specific cell types, refer to **Handling Data Using Sheet Methods**.

Cell Types

There are several different types of cells that can be set in a sheet to customize how the user interacts with the information in that cell. You can specify the cell type for individual cells, columns, rows, a range of cells, or an entire sheet. For each cell type there are properties of a cell that can be set. In general, working with cell types includes defining the cell type, setting the properties, and applying that cell type to cells.

	Cell Type	Example
1	BarCode	
2	Button	
3	CheckBox	<input checked="" type="checkbox"/>
4	ColorPicker	
5	ComboBox	Green 
6	Currency	\$123.45
7	DateTime	1/17/2008 12:00:00 AM
8	Empty	Can't be edited.
9	General	Hold's all types of data
10	HyperLink	FarPoint home page
11	Image	
12	ListBox	Red Green Blue
13	Mask	123-45-6789
14	MultiColumnComboBox	Red 
15	MultiOption	<input type="radio"/> Red <input checked="" type="radio"/> Green <input type="radio"/> Blue
16	Number	123.45
17	Percent	50 %
18	Progress	 44%
19	RegularExpression	987-12-3456
20	RichText	RichText
21	Slider	
22	Text	Text box area

In general the cell types are grouped into two broad categories: editable cell types (such as text, currency, and number) and graphical or control cell types (such as button, progress, and slider). For a list of cell types and details about using them, refer to **Cell Types**.

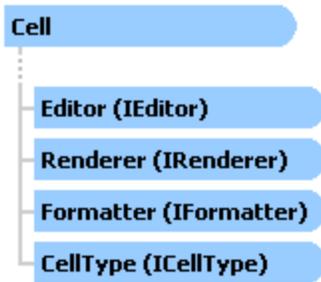
Header Cells

While you can assign a cell type to the cells in the row header or column header, the cell type is only used for painting purposes. It is rare that you would set the cell types of header cells.

Details

In Spread, a cell has both an editor, which determines how the user interacts with the value in the cell, a formatter, which determines how the value is displayed, and a renderer which does the painting of the cell. The editor is an actual

control instance that Spread creates and places in the location of the cell when you go into edit mode. The formatter decides how the displayed text appears. The renderer is simply code that paints that control inside the cell rectangle when the editor is not there.



For more detailed information on these objects, refer to the individual interfaces in the Assembly Reference. For more general information about cell types and applying them to cells, columns, rows, or whole sheets, refer to **Cell Types**.

Key Features

Spread Windows Forms introduces some powerful features which are described in the following table.

Area	Overview	Reference
Designer	Visually design components and create prototype	Spread Designer Guide (on-line documentation)
Spreadsheet Objects	Status Bar	Adding a Status Bar
	Create multiple viewports (pane)	Customizing Viewports
Sheet	Use multiple sheets	Working with Multiple Sheets
	Change sheet name tab	Customizing the Sheet Name Tabs
	Add a title and subtitle	Adding a Title and Subtitle to a Sheet
	Customize the top left cell	Customizing the Sheet Corner Appearance
	Set a style using skin	Customizing the Overall Component Appearance
	Add a footer to the bottom of sheet	Displaying a Footer for Columns or Groups
	Add a table	Tables
Row and Column	Change background color of odd row only	Creating Alternating Rows
	Display multiple columns and rows in header	Creating a Header with Multiple Rows or Columns
	Specify a non-scrollable area	Setting Fixed (Frozen) Rows or Columns
	Display grouped data	Managing Grouping of Rows of User Data
	Filtering the data	Managing Filtering of Rows of User Data
	Display supplement information in a row	Setting up Preview Rows
	Sort data in ascending or descending order	Managing Sorting of Rows of User Data
	Create a row range or column range that can be expanded and collapsed	Managing Outlines (Range Groups) of Rows and Columns

Cell	<ul style="list-style-type: none"> Change display character of column header cell Use a cell type that matches the format or data type Search for a specific cell Merge multiple cells Display string in cell's tooltip Determine cell format according to the result of conditional operation 	<ul style="list-style-type: none"> Cell Types Customizing User Searching of Data Creating a Span of Cells Adding a Note to a Cell Using Conditional Formatting of Cells
Shape	<ul style="list-style-type: none"> Add a shape to a sheet Capture content in any cell range Support for adding and customizing shapes and group shapes 	<ul style="list-style-type: none"> Customizing Drawing Creating Camera Shapes Working With Shapes (Enhanced Shape Engine)
Chart	<ul style="list-style-type: none"> Add a chart to a sheet 	<ul style="list-style-type: none"> Chart Control
Sparklines	<ul style="list-style-type: none"> Display a small graph in a cell Add Sparklines Using Methods Add Sparklines Using Formulas 	<ul style="list-style-type: none"> Sparklines Add Sparklines Using Methods Add Sparklines using Formulas
Edit	<ul style="list-style-type: none"> Allow edit mode remains on for any cell Validate user input Set initial value to a cell Perform automatic calculation based on input value Display formula text box (formula bar) Display name box Perform input by drag and drop Define external variable 	<ul style="list-style-type: none"> Understanding Edit Mode in a Cell Using Validation Placing and Retrieving Data Formulas in Cells Working with the Formula Text Box Setting up the Name Box Filling Cells with Drag and Drop Creating and Using External Variable Opening an Excel File Saving to an Excel File Printing Printing to PDF Binding to Data Working with Hierarchical Data Display Specifying What the User Can Select Customizing Clipboard Operation Options Locating the Pointer Using HitTest Customizing Undo and Redo Actions Keyboard Interaction Allowing the User to Automatically Sort Rows Touch Support with the Component
Excel	<ul style="list-style-type: none"> Load an Excel file Save to Excel file 	
Print	<ul style="list-style-type: none"> Print using printing option Print in PDF format 	
Data Binding	<ul style="list-style-type: none"> Bind a data source Display relational data hierarchically 	
User Operation	<ul style="list-style-type: none"> Allow multiple row selection Set clipboard data by copy and paste operation Get cursor position Customize the behavior of undo and redo Customize keyboard operation Sort a row by clicking column header Use on a tablet 	

Render

Make it look like a visual style
Text rendering using GDI

Using XP Themes with the Component
Text Rendering with GDI

Working with the Component

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

- **Adding NuGet Package in Spread Windows Forms**
- **Migrate .NET Framework Project to .NET 6 Platform**
- **Adding a Component to a Visual Studio 2019 Project**
- **Adding a Component to a Visual Studio 2017 Project**
- **Setting the Properties**
 - **Using Verbs in the Properties Window**
 - **Using Smart Tags Drop-Down**
 - **Working with Collection Editors**
- **Adding Support for High DPI Settings**

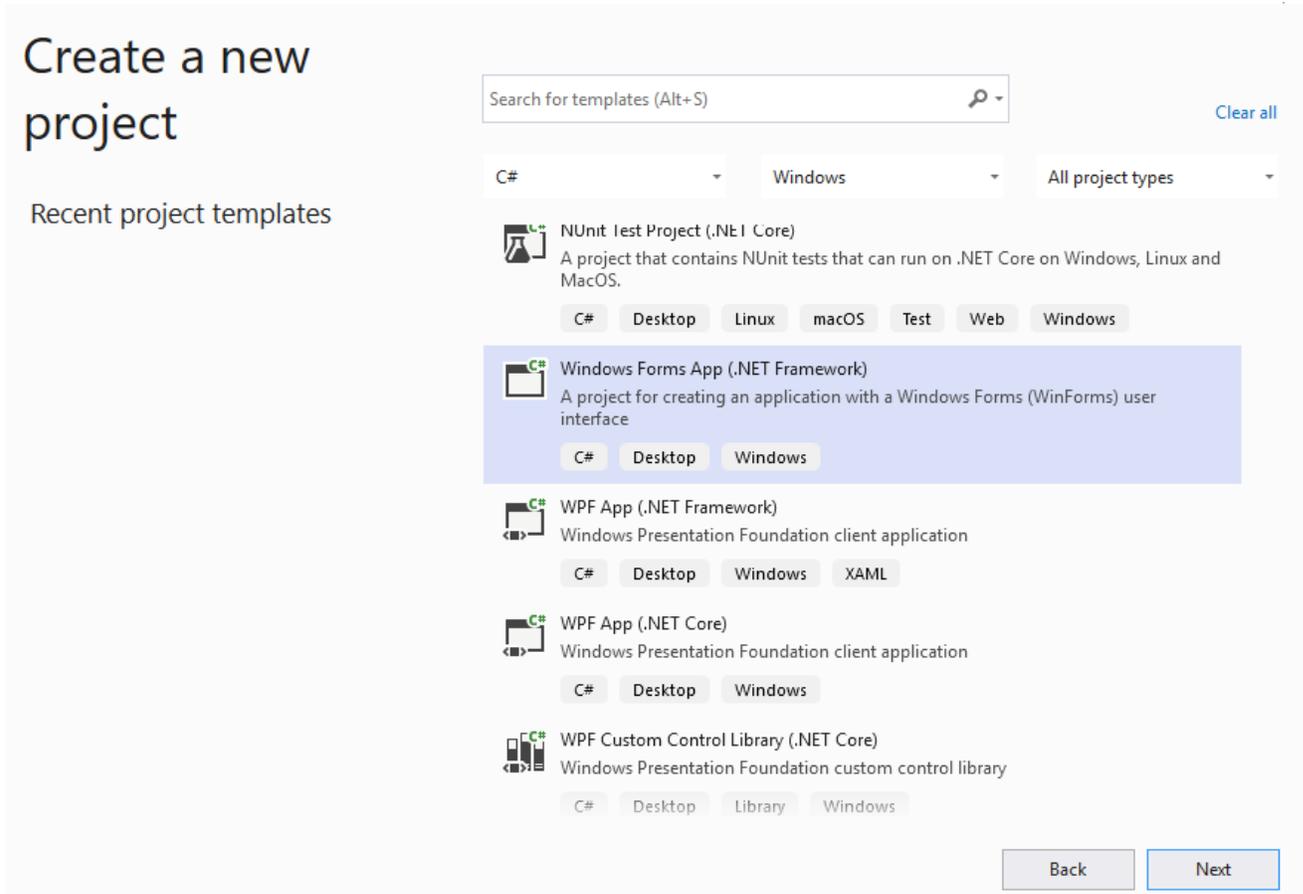
Adding NuGet Package in Spread Windows Forms

The following section explains how to use Spread Windows Forms in a .NET Framework and .NET 6 platform by using GrapeCity.Spread.WinForms NuGet package.

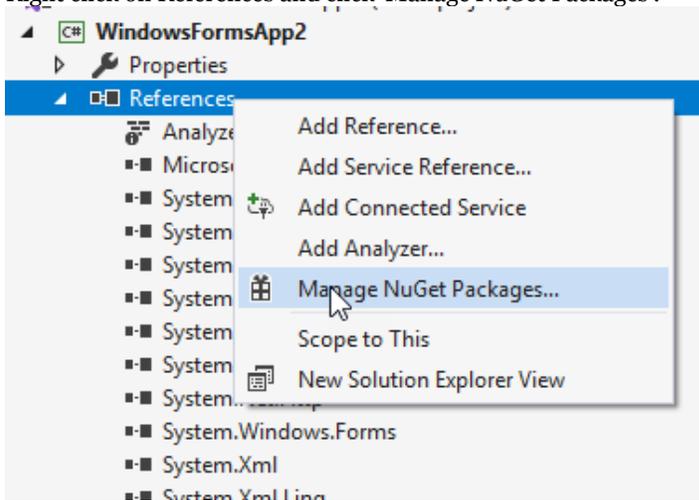
.NET Framework

Follow the below steps to create a Windows Forms application in .NET Framework platform:

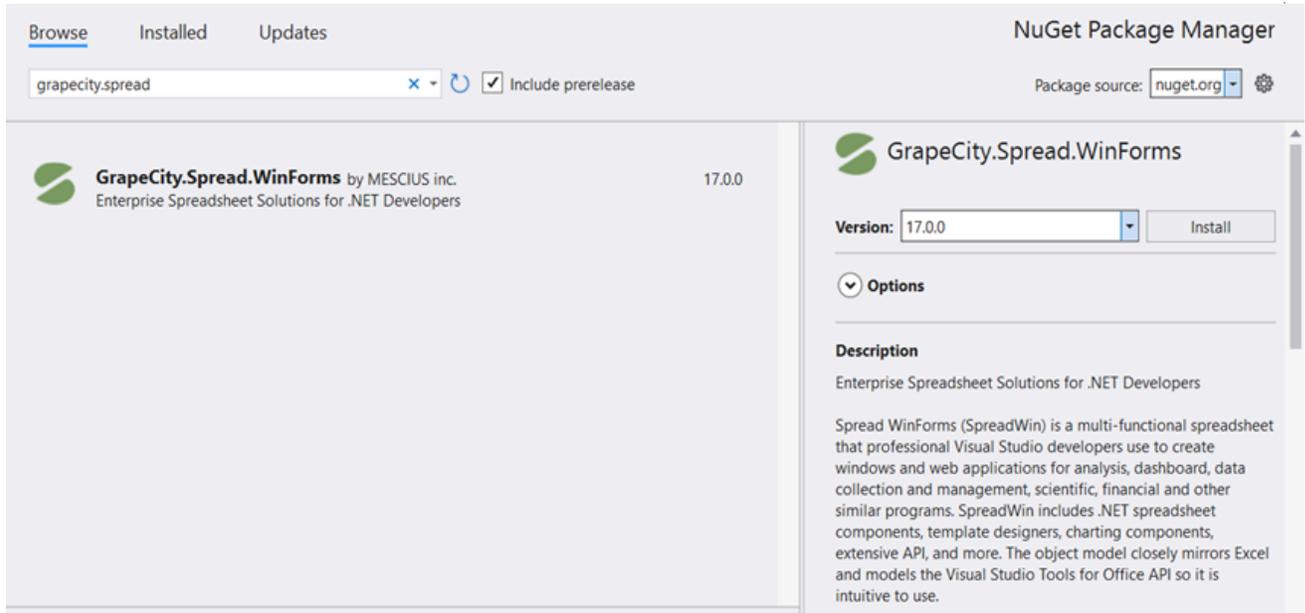
1. Click on 'Create a new project' and select 'Windows Forms App (.NET Framework)'.



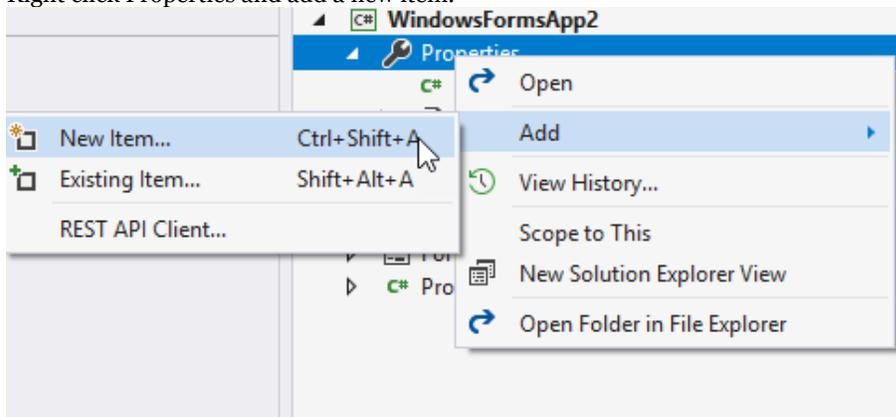
2. Name the project and click Create.
3. Right click on References and click 'Manage NuGet Packages'.



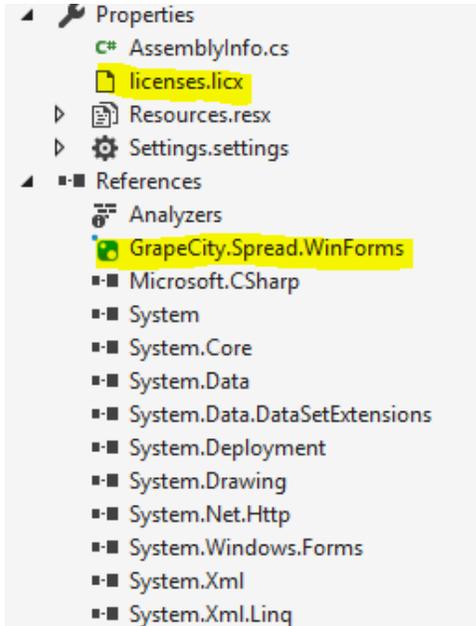
4. Search 'GrapeCity.Spread' and install GrapeCity.Spread.WinForms NuGet package in your solution.



5. Right click Properties and add a new item.



6. Add a text file from 'Add new item' dialog and name it as licenses.licx.



7. Add Spread's license string as follows:

```
FarPoint.Win.Spread.FpSpread, FarPoint.Win.Spread
```

8. Switch to the designer, and drag drop the fpSpread control on the form. Set the Name, Size, Sheet Count, and Cell Text properties from the Properties window.

OR

Double click Form1.cs and add the fpSpread control using the following code in its form load event.

Form1.cs

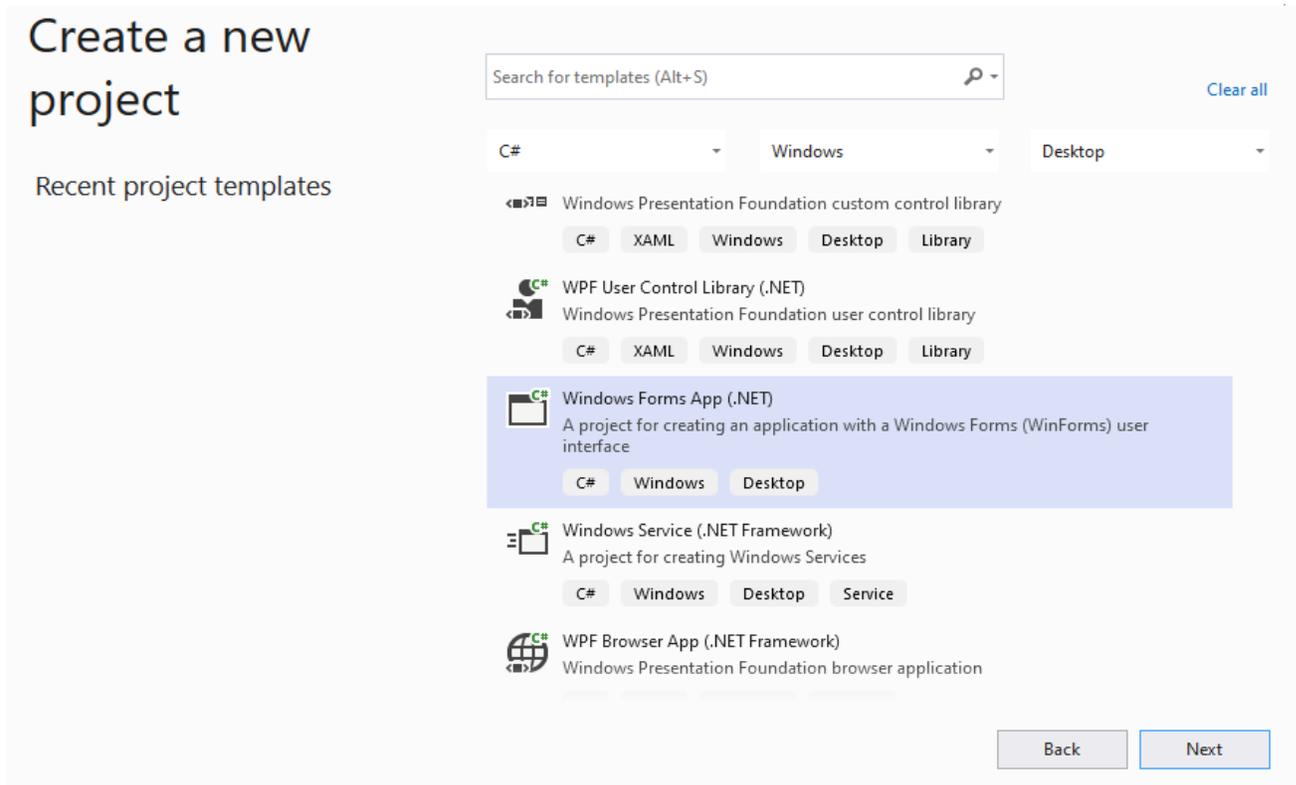
```
FpSpread fpSpread1 = new FpSpread();  
fpSpread1.Location = new System.Drawing.Point(52, 12);  
fpSpread1.Name = "fpSpread1";  
fpSpread1.Size = new System.Drawing.Size(653, 380);  
fpSpread1.Sheets.Count = 2;  
this.Controls.Add(fpSpread1);  
fpSpread1.ActiveSheet.Cells[0, 0].Text = "Welcome";
```

9. Build the project and run to view fpSpread control on to the Form.

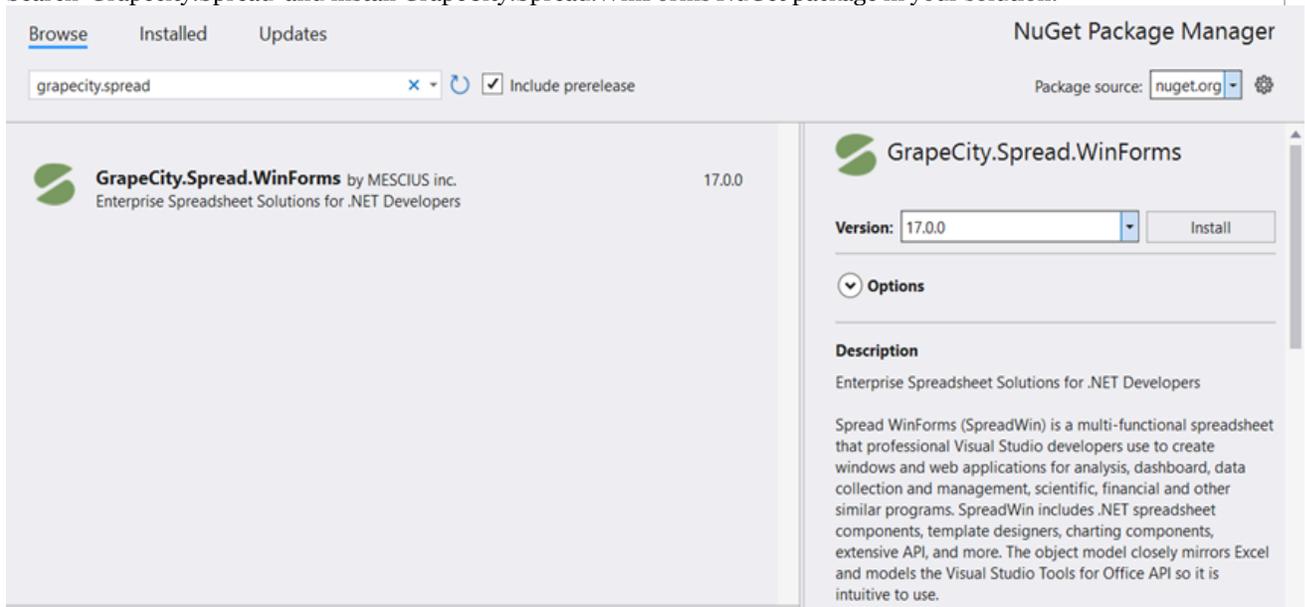
.NET 6

Follow the below steps to create a Windows Forms application in .NET 6/.NET Core 3.1 platforms:

1. Click on 'Create a new project' and select 'Windows Forms App (.NET)'.



2. Name the project and click Create.
3. Right click on References and click 'Manage NuGet Packages'.
4. Search 'Grapecity.Spread' and install GrapeCity.Spread.WinForms NuGet package in your solution.



5. Switch to the designer, and drag drop the fpSpread control on the form. Set the Name, Size, Sheet Count, and Cell Text properties from the Properties window.
OR
Double click Form1.cs and add the fpSpread control using the following code in its form load event.

Form1.cs

```
FpSpread fpSpread1 = new FpSpread();
fpSpread1.Location = new System.Drawing.Point(52, 12);
fpSpread1.Name = "fpSpread1";
fpSpread1.Size = new System.Drawing.Size(653, 380);
fpSpread1.Sheets.Count = 2;
this.Controls.Add(fpSpread1);
fpSpread1.ActiveSheet.Cells[0, 0].Text = "Welcome";
```

6. Build the project and run to view fpSpread control on to the Form.

 Licenses.licx file with FpSpread license string isn't required in .NET 6 application.

To work with the Spread Designer in your project using NuGet packages, refer to the topic **Using Spread Designer in Runtime (on-line documentation)**.

Migrate .NET Framework Project to .NET 6 Platform

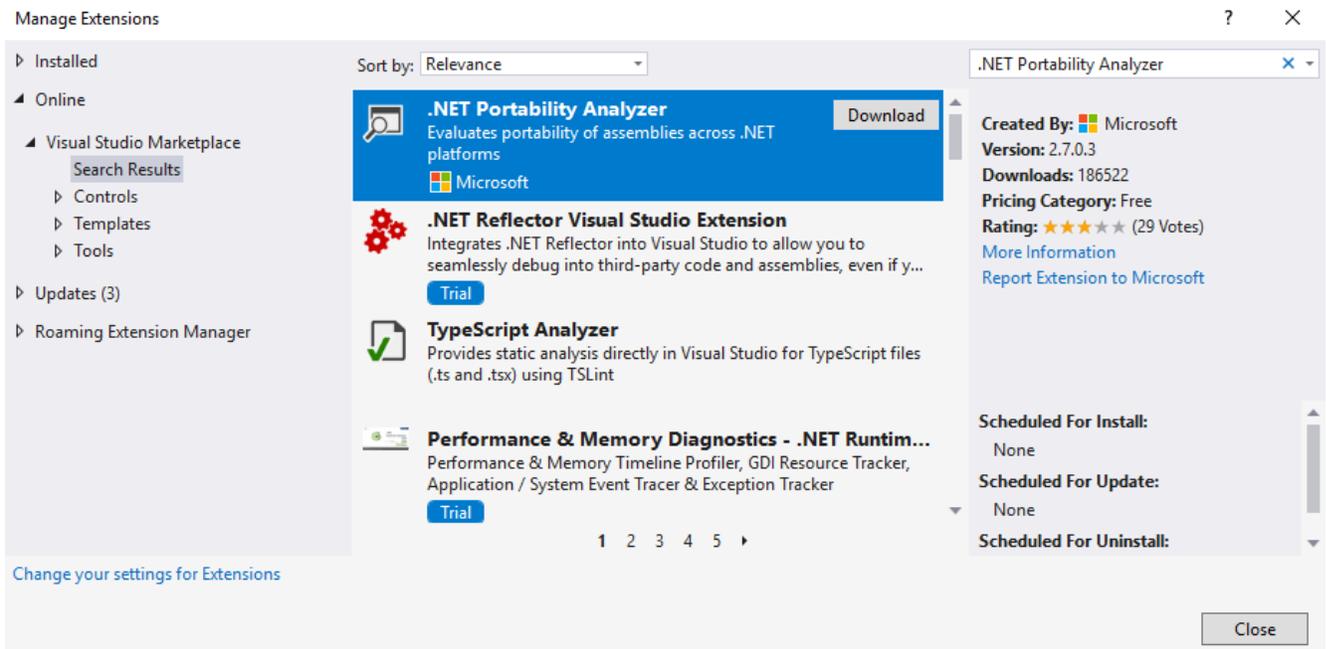
Spread Windows Forms supports .NET 6 platform. If you want to migrate a .NET Framework project to .NET 6 platform, go through the detailed steps mentioned in this topic.

The following steps consider 'GrapeCity.AgingReport' as a sample .NET Framework project which needs to be migrated to .NET 6 platform. You can also locate this project on your system at `C:\Program Files (x86)\Mescius\Spread.NET [version]\Windows Forms\[version]\Samples\C#\Spread.Examples\` after you install SpreadWin successfully.

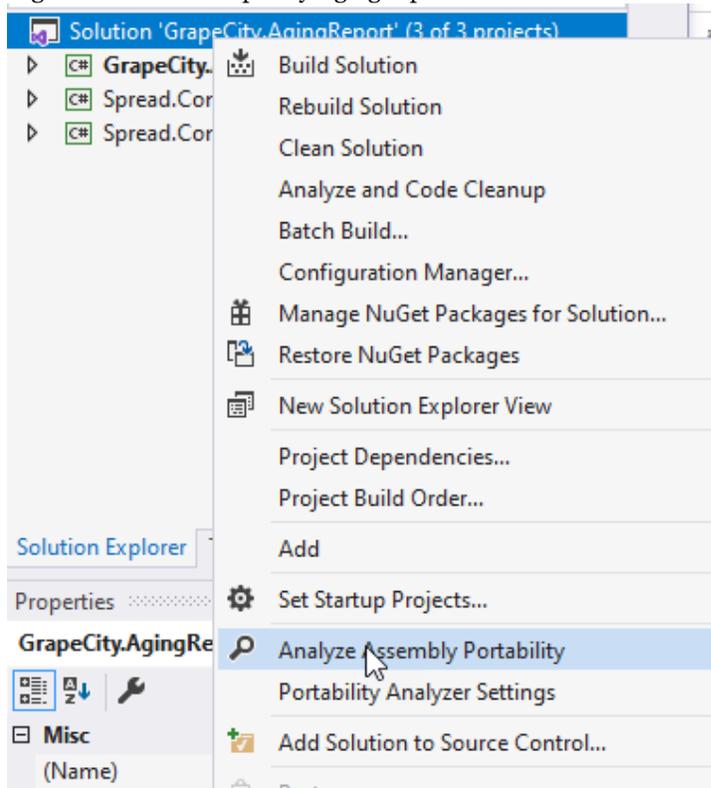
Step 1: Analyze Portability

(This section helps you to check if the code in your project is portable and supported on .NET 6 platform. You can ignore this section if you are already clear about the project portability.)

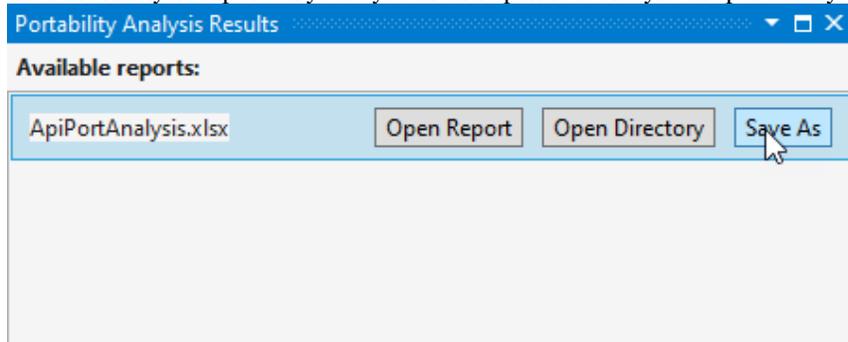
1. Open the `GrapeCity.AgingReport.sln` file in Visual Studio 2019 v16.8+.
2. Click on `Extensions > Manage Extensions` in Visual Studio 2019 v16.8+ tabs. In Manage Extensions dialog, search for '.NET Portability Analyzer' and download it.



3. Right click on the GrapeCity.AgingReport solution and click 'Analyze Assembly Portability'.



4. Save the Analysis report on your system and open it to analyze the portability details of your project.



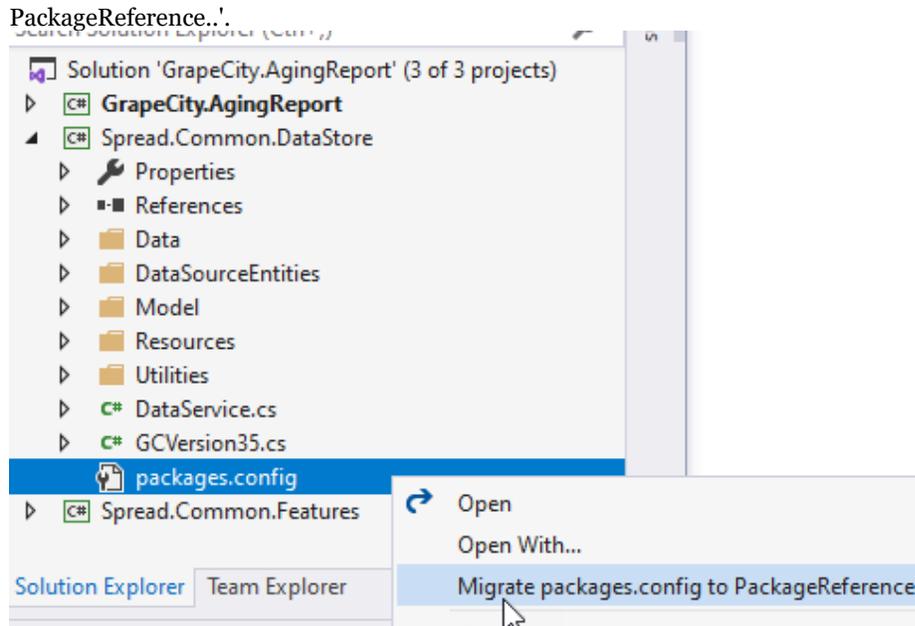
Note: In this example, the Analysis report shows that the Spread.Common.DataStore project is not 100% supported on .NET Core. You can click on the 'Details' tab of report to see that the issues are related to assemblies (Newtonsoft.Json, System.Data.OleDb). Hence, their latest packages will be installed from NuGet in next section.

Step 2: Migrate Projects to .NET 6 Platform

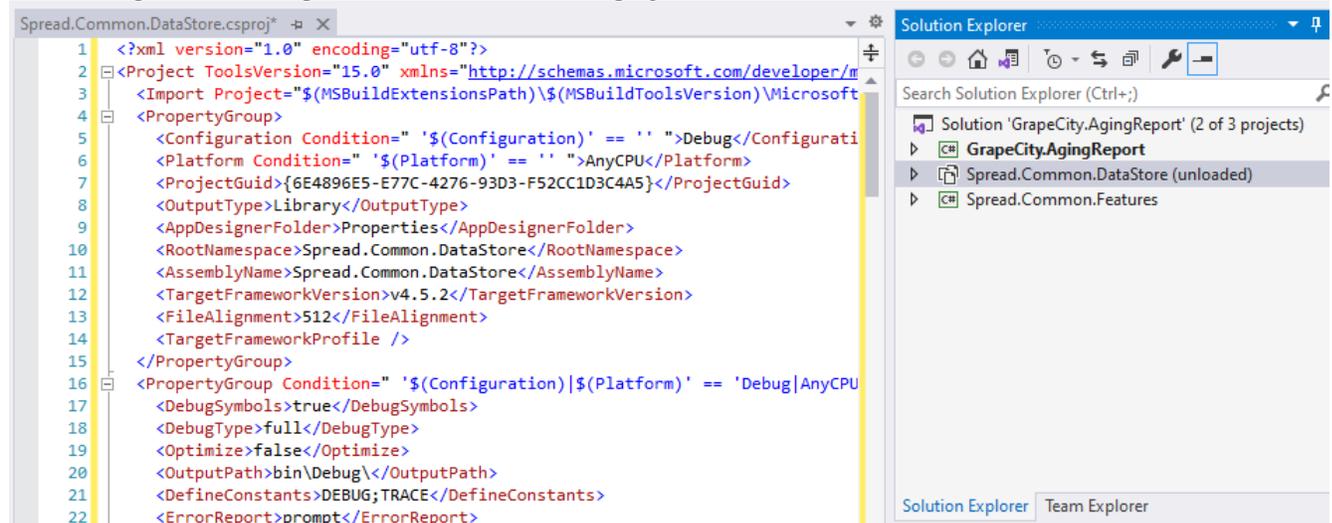
(In this example, there are 3 projects in the solution where GrapeCity.AgingReport project refers to Spread.Common.DataStore and Spread.Common.Feature project. Hence GrapeCity.AgingReport project will be migrated after migrating the other two projects.)

Migrate Spread.Common.DataStore Project

1. Right click packages.config in Spread.Common.DataStore project and click 'Migrate packages.config to



2. Click Ok in 'Migrate NuGet format to PackageReference - Spread.Common.DataStore' dialog box.
3. Right click Spread.Common.DataStore project and click 'Unload Project'. Double click Spread.Common.DataStore in Solution Explorer to view Spread.Common.DataStore.csproj file.



4. Remove the content of Spread.Common.DataStore.csproj file and copy it in a text file so that Spread.Common.DataStore.csproj file appears blank and you have its backup in a text file.
5. Add the following code to blank Spread.Common.DataStore.csproj file to change it to project SDK type. Note that the OutputType is "Library" because DataStore project is class library project.

Spread.Common.DataStore.csproj

```

<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>library</OutputType>
    <TargetFramework>net6.0-windows</TargetFramework>
  </PropertyGroup>
</Project>
    
```

6. Find the text "PackageReference" in the backup text file and copy the whole ItemGroup to Spread.Common.DataStore.csproj file.

Spread.Common.DataStore.csproj

```
<ItemGroup>
  <PackageReference Include="Newtonsoft.Json">
    <Version>11.0.2</Version>
  </PackageReference>
</ItemGroup>
```

7. Turn off 'auto generate assembly info' by adding below code to Spread.Common.DataStore.csproj file.

Spread.Common.DataStore.csproj

```
<PropertyGroup>
  <GenerateAssemblyInfo>>false</GenerateAssemblyInfo>
</PropertyGroup>
```

8. Install the latest versions of Newtonsoft.Json and System.Data.OleDb from NuGet to make sure that they support .NET 6.
9. Right click Spread.Common.DataStore in Solution Explorer and click on Reload project. The final content of Spread.Common.DataStore.csproj file will look like below:

Spread.Common.DataStore.csproj

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

  <PropertyGroup>
    <OutputType>library</OutputType>
    <TargetFramework>net6.0-windows</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Newtonsoft.Json">
      <Version>12.0.3</Version>
    </PackageReference>
    <PackageReference Include="System.Data.OleDb" Version="4.7.1" />
  </ItemGroup>
  <PropertyGroup>
    <GenerateAssemblyInfo>>false</GenerateAssemblyInfo>
  </PropertyGroup>
</Project>
```

10. Rebuild Spread.Common.DataStore project to observe that the build is successful and the project has been successfully migrated to .NET 6 platform.

Migrate Spread.Common.Features Project

You can migrate Spread.Common.Features project to .NET 6 platform in the same way as Spread.Common.DataStore project in above section (except the installation of System.Data.OleDb Nuget package in step 8). The final code of Spread.Common.Features.csproj will look like below:

Spread.Common.Features.csproj

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

  <PropertyGroup>
```

```

        <OutputType>library</OutputType>
        <TargetFramework>net6.0-windows</TargetFramework>
    </PropertyGroup>

    <ItemGroup>
        <PackageReference Include="Newtonsoft.Json">
            <Version>12.0.3</Version>
        </PackageReference>
    </ItemGroup>
    <PropertyGroup>
        <GenerateAssemblyInfo>>false</GenerateAssemblyInfo>
    </PropertyGroup>
</Project>

```

Rebuild Spread.Common.Features project to observe that the build is successful and the project has been successfully migrated to .NET 6 platform.

Migrate GrapeCity.AgingReport Project

You can migrate GrapeCity.AgingReport project to .NET 6 platform in the same way as Spread.Common.DataStore project in above section except the following:

- Skip migrating packages.config to PackageReference as GrapeCity.AgingReport does not have packages.config (done in Step 1).
- GrapeCity.AgingReport project refers Spread.Common.DataStore and Spread.Common.Features project. Hence, add the ProjectReference to GrapeCity.AgingReport.csproj file from the text backup file (done in Step 6).
- Skip installing the latest versions of Newtonsoft.Json and System.Data.OleDb from NuGet (done in Step 8).

The final code of GrapeCity.AgingReport.csproj will look like below:

GrapeCity.AgingReport.csproj

```

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

    <PropertyGroup>
        <OutputType>WinExe</OutputType>
        <TargetFramework>net6.0-windows</TargetFramework>
    </PropertyGroup>

    <ItemGroup>
        <ProjectReference
Include="..\..\Common\Spread.Common.DataStore\Spread.Common.DataStore.csproj">
            <Project>{6e4896e5-e77c-4276-93d3-f52cc1d3c4a5}</Project>
            <Name>Spread.Common.DataStore</Name>
        </ProjectReference>
        <ProjectReference
Include="..\..\Common\Spread.Common.Features\Spread.Common.Features.csproj">
            <Project>{08b57d63-d0ad-431a-819a-c3a00f479feb}</Project>
            <Name>Spread.Common.Features</Name>
        </ProjectReference>
    </ItemGroup>

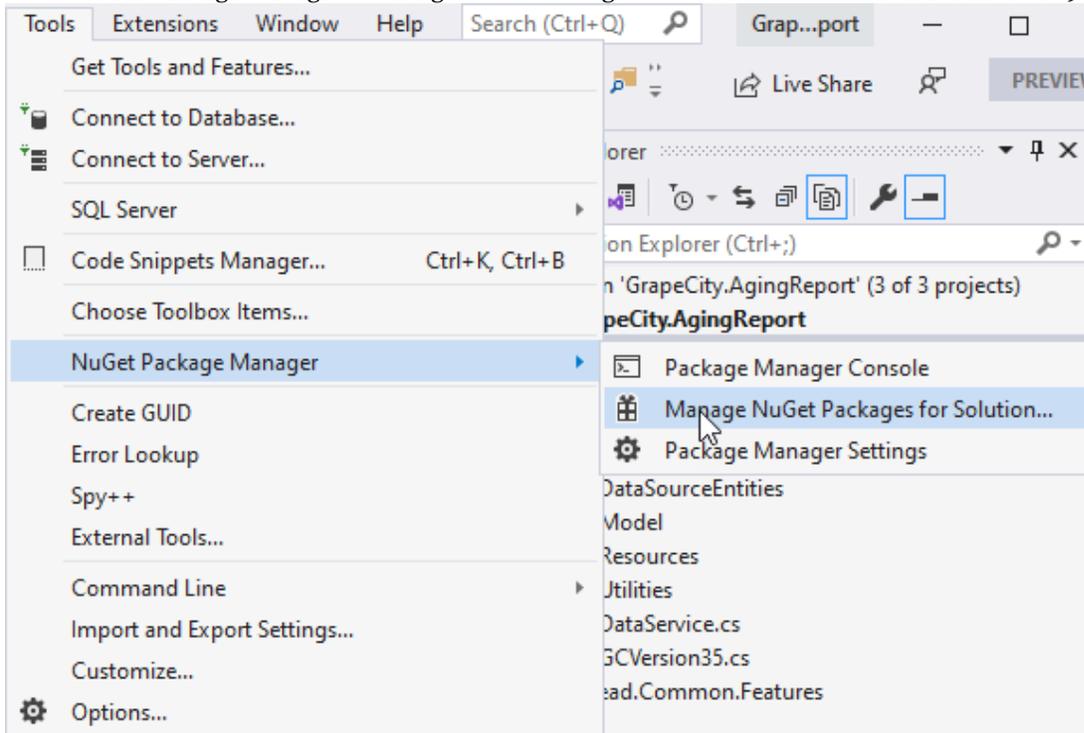
    <PropertyGroup>
        <GenerateAssemblyInfo>>false</GenerateAssemblyInfo>
    </PropertyGroup>
</Project>

```

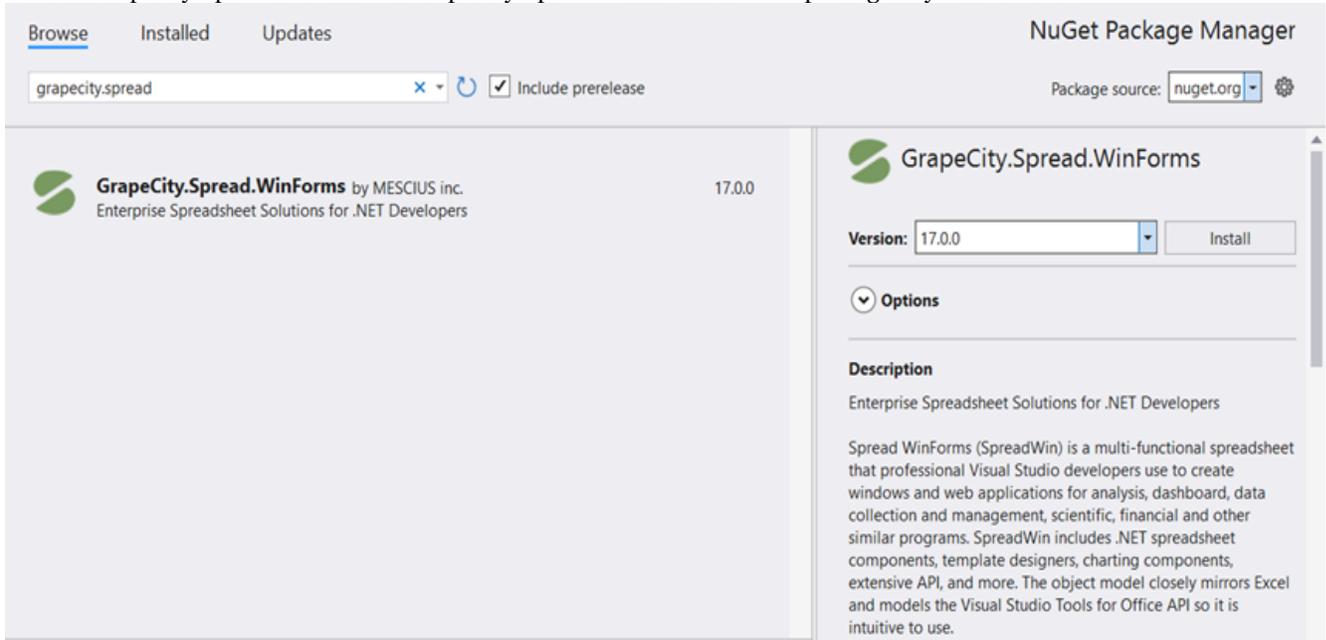
Rebuild GrapeCity.AgingReport project to observe that the build is successful and the project has been successfully migrated to .NET 6 platform.

Step 3: Install SpreadWin NuGet Package

1. Select NuGet Package Manager > Manage NuGet Packages for Solution from the Visual Studio 2019 v16.8+Tools tab.



2. Search 'Grapecity.Spread' and install GrapeCity.Spread.WinForms NuGet package in your solution.



3. Rebuild the solution to observe that the build is successful. The Grapecity.AgingReport solution is successfully migrated to .NET 6 platform.

Adding a Component to a Visual Studio 2019 Project

Use the following steps to add the component to a project in Visual Studio .NET.

The first step is to create a new project in Visual Studio .NET, and to add a Spread Windows Forms component to the project.

1. Start Visual Studio .NET.
2. Click **Create a new project** in the "Get Started" section that appears on the right to create a new project. Alternatively, you can also go to the **File** menu, choose **New, Project** to create a new project.
3. In the **Create a new Project** dialog, select a project type depending on the language environment in which you are developing and click **Next**. For example, choose **Windows Forms App (.NET Framework)** containing **Visual Basic** under it.
4. Now, configure your new project as described in the steps shared below:
 - a. In the **Project Name** box, type the name of the new project. The default is **WindowsApp1** for the first Windows Forms application.
 - b. In the **Location** box, leave the location path as the designated path, or click the Browse button to change the path to a new directory.
 - c. Click **Create**.

If your project does not display the **Solution Explorer**, from the **View** menu, choose **Solution Explorer**.

5. In the **Solution Explorer**, right-click on the form name, **Form1**. Choose **Rename** from the pop-up menu, then type the new form name you prefer for the new form name.

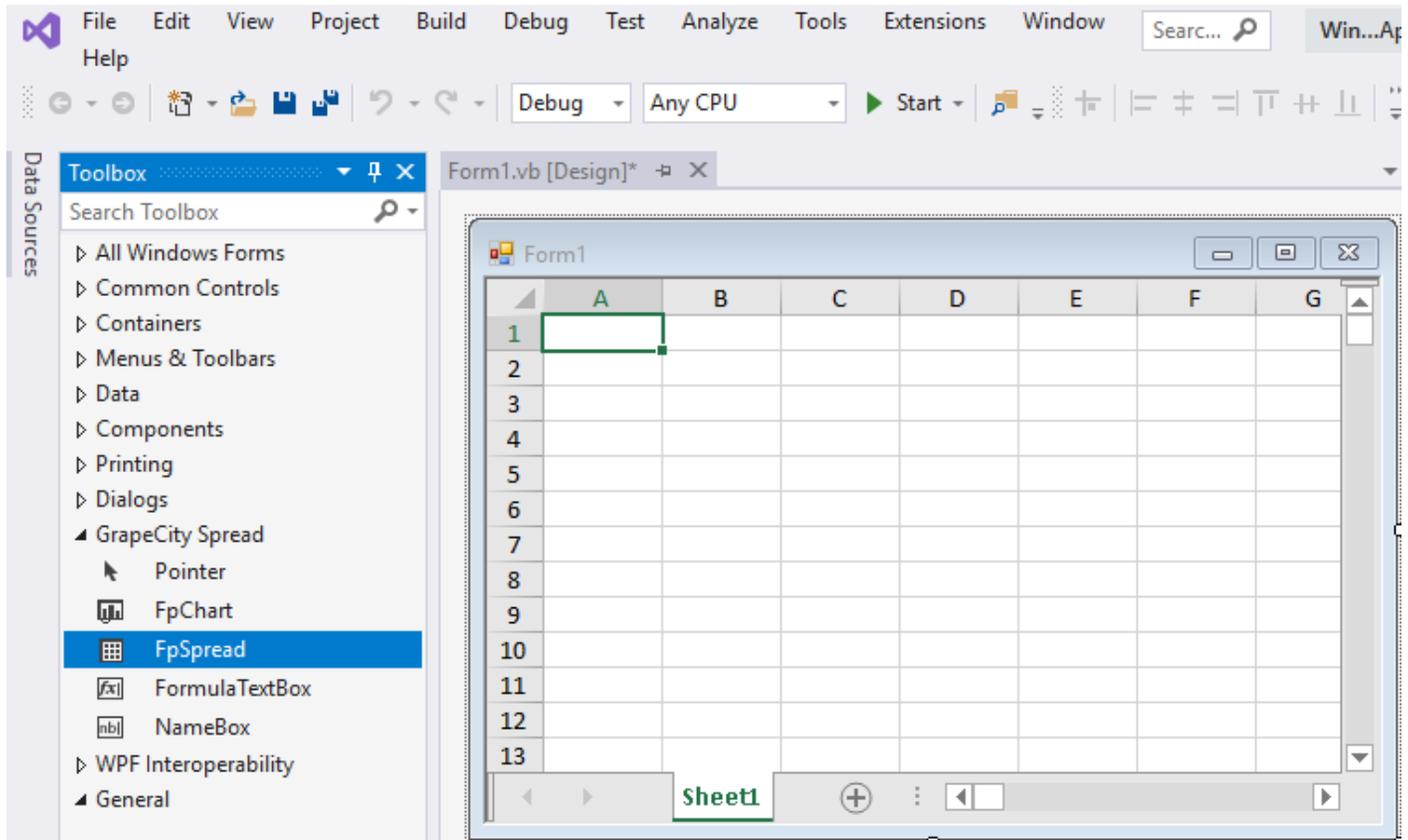
Use the following steps to add the component to the toolbox if the component is not listed in the toolbox.

1. If the Toolbox is not displayed, from the **View** menu choose **Toolbox**.
2. Once the Toolbox is displayed, look in the **Spread** category (or in any other category if you have installed Spread and placed the toolbox icon in a different category).
3. If the Spread component is not in the Toolbox:
 - a. Right-click in the Toolbox, and from the pop-up menu choose **Choose Items**.
 - b. In the **Choose Toolbox Items** dialog, click the **.NET Framework Components** tab.
 - c. In the **.NET Framework Components** tab, the FpSpread component (in the FarPoint.Win.Spread namespace) should be displayed in the list of components. Select the Spread component check box and click **OK**. Select fpChart (FarPoint.Win.Chart namespace) for the chart control.
If the Spread component is not displayed in the list of components, click **Browse** and browse to the installation path for the Spread Windows Forms component. Once there, select the FarPoint.Win.Spread.dll and click **Open**. The Spread component is now displayed in the list of components. Select it and click **OK**. Select FarPoint.Win.Chart.dll for the chart control.
 - d. You can test that the component has been added by opening a project and inserting the component.

The next step is to add the Spread component to a project.

1. With an open project, in the Toolbox under **Spread** (or whatever category to which you added it), select the FpSpread component.
2. On your Windows Forms page, draw a Spread component by dragging a rectangle the size that you would like the initial component or simply double-click on the page. The Spread component appears. The Spread Designer also appears by default. Close the designer.

Your project should look similar to the following picture.



You have successfully added the Spread component to the project.

Adding a Component to a Visual Studio 2017 Project

Use the following steps to add the component to a project in Visual Studio .NET.

The first step is to create a new project in Visual Studio .NET, and to add a Spread Windows Forms component to the project.

1. Start Visual Studio .NET.
2. From the **File** menu, choose **New, Project**.
3. In the **New Project** dialog, in the **Installed** area, select a project type depending on the language environment in which you are developing. For example, choose **Windows** under **Visual Basic**.
 - a. Choose the type of project such as **Windows Forms Application**.
 - b. In the **Name** box, type the name of the new project. The default is **WindowsApplication1** for the first Windows Forms application.
 - c. In the **Location** box, leave the location path as the designated path, or click **Browse** to change the path to a new directory.
 - d. Click **OK**.

If your project does not display the **Solution Explorer**, from the **View** menu, choose **Solution Explorer**.

4. In the **Solution Explorer**, right-click on the form name, **Form1**. Choose **Rename** from the pop-up menu, then type the new form name you prefer for the new form name.

Use the following steps to add the component to the toolbox if the component is not listed in the toolbox.

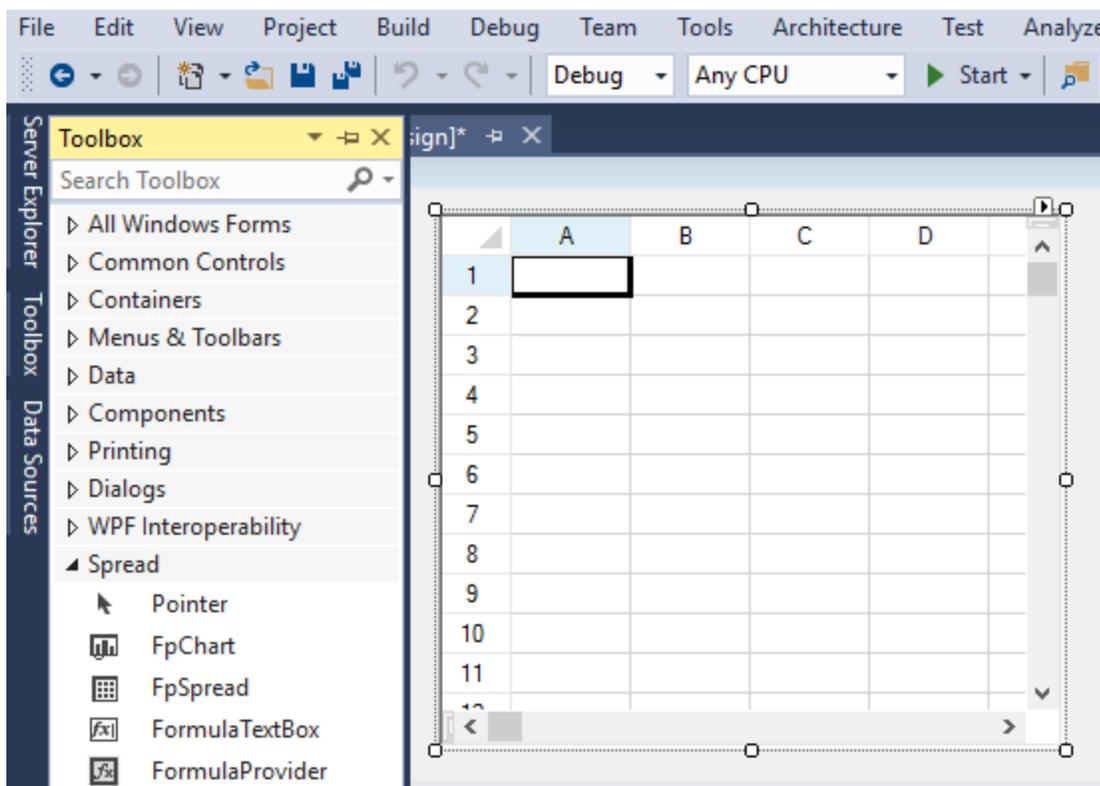
1. If the Toolbox is not displayed, from the **View** menu choose **Toolbox**.

2. Once the Toolbox is displayed, look in the **Spread** category (or in any other category if you have installed Spread and placed the toolbox icon in a different category).
3. If the Spread component is not in the Toolbox:
 - a. Right-click in the Toolbox, and from the pop-up menu choose **Choose Items**.
 - b. In the **Choose Toolbox Items** dialog, click the **.NET Framework Components** tab.
 - c. In the **.NET Framework Components** tab, the FpSpread component (in the FarPoint.Win.Spread namespace) should be displayed in the list of components. Select the Spread component check box and click **OK**. Select fpChart (FarPoint.Win.Chart namespace) for the chart control.
If the Spread component is not displayed in the list of components, click **Browse** and browse to the installation path for the Spread Windows Forms component. Once there, select the FarPoint.Win.Spread.dll and click **Open**. The Spread component is now displayed in the list of components. Select it and click **OK**. Select FarPoint.Win.Chart.dll for the chart control.
 - d. You can test that the component has been added by opening a project and inserting the component.

The next step is to add the Spread component to a project.

1. With an open project, in the Toolbox under **Spread** (or whatever category to which you added it), select the FpSpread component.
2. On your Windows Forms page, draw a Spread component by dragging a rectangle the size that you would like the initial component or simply double-click on the page. The Spread component appears. The Spread Designer also appears by default. Close the designer.

Your project should look similar to the following picture.



You have added the Spread component to the project.

Setting the Properties

There are several ways to set the properties of the Spread component in design mode. Here, we will explain the

following methods.

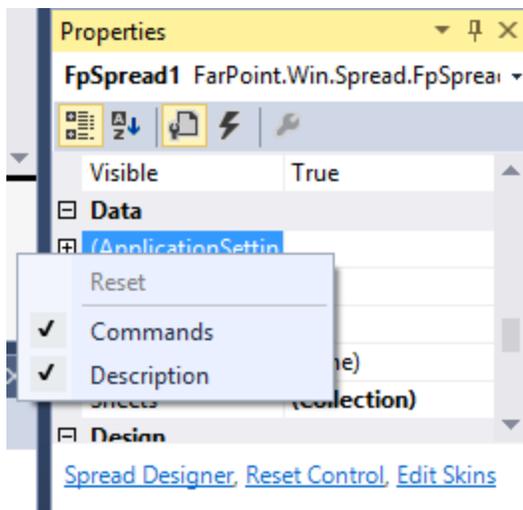
- **Using Verbs in the Properties Window**
- **Using Smart Tags Drop-Down**
- **Working with Collection Editors**

You can use the Spread designer to set the more advanced properties. For more information on the designer, refer to **Spread Designer Guide (on-line documentation)**.

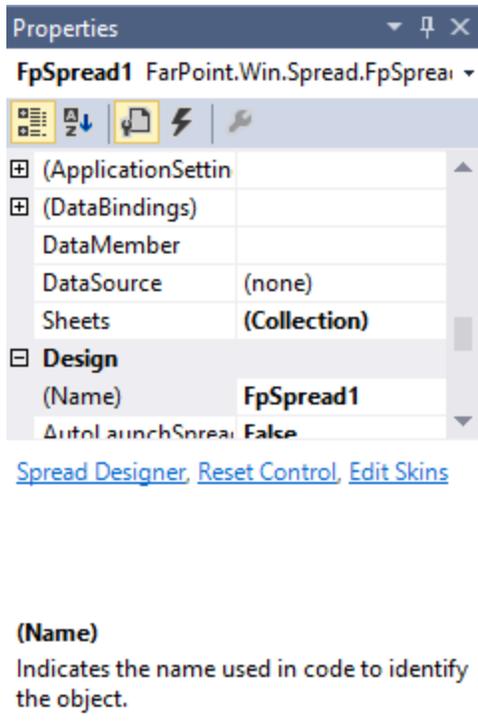
Using Verbs in the Properties Window

You can launch various editors or reset values from the verbs in the property window in Visual Studio .NET as a quick way of handling some settings. There are different verbs available depending on the item selected.

Right-click on the **Properties** window and select **Commands** to see the verbs.



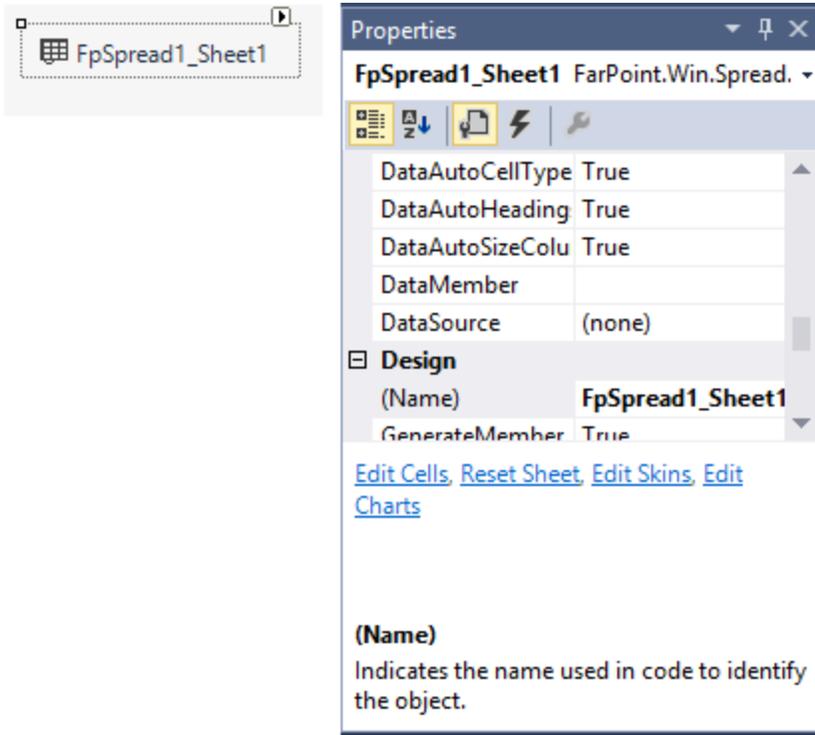
The following image displays Spread verbs.



The following table summarizes the Spread component verbs:

Verb	Description
Spread Designer	This opens the Spread Designer and allows you to edit various properties of most of the spreadsheet and its parts as well as the overall component. For more information on the use of the Spread Designer, refer to Spread Designer Guide (on-line documentation) .
Reset Control	This restores all the settings for the Spread component to their default values. For more information on resetting properties, refer to Resetting Parts of the Interface .
Edit Skins	This opens the Spread Skin Editor and allows you to edit various properties of the skin that apply to the spread sheet. For more information on managing skins, refer to Creating a Custom Skin for a Component and Applying a Skin to the Component .

The following image displays the sheet verbs.



The following table summarizes the sheet verbs:

Verbs Description

Edit Cells	This opens the Cell Editor and allows you to edit various properties of the selected cell or cells of a sheet. For more information on setting cell properties, refer to Customizing the Appearance of a Cell .
Reset Sheet	This restores all the settings of the selected sheet to their default values. For more information on resetting properties, refer to Resetting Parts of the Interface .
Edit Skins	This opens the Spread Skin Editor and allows you to edit various properties of the skin that apply to the spread sheet. For more information on managing skins, refer to Creating a Custom Skin for a Component and Applying a Skin to the Component .
Edit Charts	This opens the SpreadChart Collection Editor. For more information, refer to SpreadChart Collection Editor (on-line documentation) .

Using Smart Tags Drop-Down

You can perform any of several tasks, launch various editors, and set various properties from the smart tags drop-down available from the Spread component on a Form in Visual Studio .NET. The smart tag is the arrow icon at the top, right edge of the control. The Spread tasks available in the smart tags are summarized below.

Task

Explanation or Reference

Choose Data Source

Refer to **Data Binding**.

Edit Sheets

Refer to **Customizing the Individual Sheet Appearance**.

Edit Cells

Refer to **Customizing the Appearance of a Cell** and **Customizing Interaction in Cells**.

Component Name

This is the name of the Spread component.

Operation Mode

Refer to **Specifying What the User Can Select**.

User Options

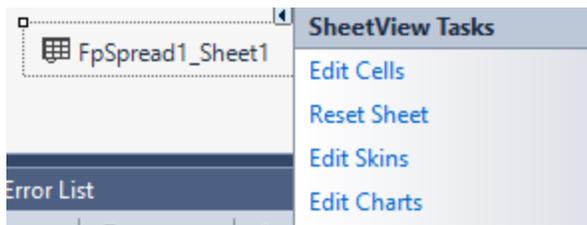
- EditModePermanent
- EditModeReplace
- AllowColumnMove

Refer to **Moving Rows or Columns, Allowing the User to Zoom the Display of the Component, Filling Cells with Drag and Drop, Filling Cells with Drag and Fill, Allowing the User to Enter Formulas, and Understanding Edit Mode in a Cell**.

- AllowRowMove
- AllowUserZoom
- AllowDragDrop
- AllowDragFill
- AllowUserFormulas

AutoClipboard	This allows the short cut keys to work.
Clipboard Options	This determines what can be copied and pasted.
Visual Styles	Whether to allow the visual styles.
Edit Sheet Skins	This can be used to edit sheet skins.
Edit Named Styles	This can be used to edit named styles.
Spread Designer	This can be used to bring up the designer.
Quick Start Wizard	This can be used to bring up the wizard. Refer to Understanding the Spread Wizard .
AutoLaunch Spread Designer	This can be unchecked to prevent the auto launch of the designer.
Docking in Parent Container	This sets the docking to fill.
Product Version	Version of the product.

The sheet tasks available in the smart tag are summarized below.

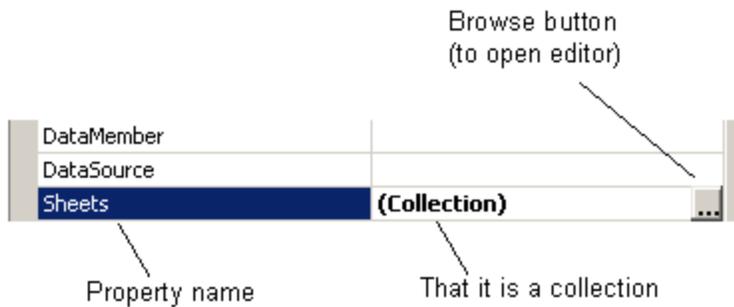


Tasks Description

Edit Cells	This opens the Cell Editor and allows you to edit various properties of the selected cell or cells of a sheet. For more information on setting cell properties, refer to Customizing the Appearance of a Cell .
Reset Sheet	This restores all the settings of the selected sheet to their default values. For more information on resetting properties, refer to Resetting Parts of the Interface .
Edit Skins	This opens the Spread Skin Editor and allows you to edit various properties of the skin that apply to the spread sheet. For more information on managing skins, refer to Creating a Custom Skin for a Component and Applying a Skin to the Component .
Edit Charts	This opens the SpreadChart Collection Editor. For more information, refer to SpreadChart Collection Editor (on-line documentation) .

Working with Collection Editors

Several properties that appear in the **Properties** window are associated with collections. To view and modify these settings, click on the **Browse** button (...) and a separate **Collection Editor** window appears. This is the case for the **NamedStyles** ('**NamedStyles Property**' in the on-line documentation) property and the **Sheets** ('**Sheets Property**' in the on-line documentation) property in the Spread component.



With these collection editors, you must click **OK** to see the results of a change to a setting. (The collection editors rely on part of the Microsoft .NET framework and do not have an **Apply** button.)

Adding Support for High DPI Settings

Spread supports high DPI settings provided that the application has high DPI support enabled.

Refer to Microsoft's web site for more information about enabling DPI support for your application.

Enabling DPI support in .NET 4.6.2

You can refer to the following steps for a general example of how to enable DPI support in .NET 4.6.2.

1. Use the Windows API with the following code.

Code

```
static class Program
{
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main()
    {
        SetProcessDPIAware();
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        Application.Run(new MainForm());
    }

    [System.Runtime.InteropServices.DllImport("user32.dll")]
    public extern static IntPtr SetProcessDPIAware();
}
```

 **Note:** The **SetProcessDPIAware** API must be called before any window is created; otherwise, a window created before using this API will have the wrong size and font.

2. Add Spread code:

Code

```
//add code in InitializeComponent() of cs file
this.fpSpread1.SpreadScaleMode = FarPoint.Win.Spread.ScaleMode.ZoomDpiSupport;
this.AutoScaleDimensions = new System.Drawing.SizeF(96F, 96F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Dpi;
```

3. Enable Windows Forms High DPI support by adding code to the App.config.

Code

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="EnableWindowsFormsHighDpiAutoResizing" value="true" />
  </appSettings>
</configuration>
```

Enabling HDPI for .NET Framework 4.7

Use the following steps to enable DPI support.

1. Install the .NET 4.7 Framework.
2. Create a new windows forms application.
3. Add an App.config with the following information to the root directory of the form (same directory as Program.cs).

Code

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7"/>
  </startup>
  <System.Windows.Forms.ApplicationConfigurationSection>
    <add key="DpiAwareness" value="PerMonitorV2"/>
    <!-- Uncomment each value to disable the fixes one by one. -->
    <!--
    <add key="Form.DisableSinglePassScalingOfDpiForms" value="true"/>
    <add key="ToolStrip.DisableHighDpiImprovements" value="true"/>
    <add key="CheckedListBox.DisableHighDpiImprovements" value="true"/>
    <add key="MonthCalendar.DisableHighDpiImprovements" value="true"/>
    <add key="AnchorLayout.DisableHighDpiImprovements" value="true"/>
    <add key="DataGridView.DisableHighDpiImprovements" value="true"/>
    -->
  </System.Windows.Forms.ApplicationConfigurationSection>
</configuration>
```

4. Add an App.manifest with the following information to the root directory of the form (same directory as Program.cs).

Code

```
<?xml version="1.0" encoding="utf-8"?>
<assembly manifestVersion="1.0" xmlns="urn:schemas-microsoft-com:asm.v1">
  <assemblyIdentity version="1.0.0.0" name="MyApplication.app"/>
  <trustInfo xmlns="urn:schemas-microsoft-com:asm.v2">
    <security>
      <requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">
        <!-- UAC Manifest Options
        If you want to change the Windows User Account Control level replace
the
        requestedExecutionLevel node with one of the following.
        <requestedExecutionLevel level="asInvoker" uiAccess="false" />
        <requestedExecutionLevel level="requireAdministrator" uiAccess="false" />
```

```

        <requestedExecutionLevel level="highestAvailable" uiAccess="false" />
        Specifying requestedExecutionLevel element will disable file and
registry virtualization.
        Remove this element if your application requires this virtualization
for backwards
        compatibility.
        -->
        <requestedExecutionLevel level="asInvoker" uiAccess="false" />
    </requestedPrivileges>
</security>
</trustInfo>
<compatibility xmlns="urn:schemas-microsoft-com:compatibility.v1">
    <application>
        <!-- A list of the Windows versions that this application has been tested on
and is
        is designed to work with. Uncomment the appropriate elements and
Windows will
        automatically selected the most compatible environment. -->

        <!-- Windows 10 -->
        <supportedOS Id="{8e0f7a12-bfb3-4fe8-b9a5-48fd50a15a9a}" />
    </application>
</compatibility>
<!--<application xmlns="urn:schemas-microsoft-com:asm.v3">
    <windowsSettings>
        <dpiAware
xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSettings">true</dpiAware>
    </windowsSettings>
</application-->

</assembly>

```

5. Verify that Program.cs has the following settings.

Code

```

static class Program
{
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        Application.Run(new Form1());
    }
}

```

6. Check that Form1.designer.cs has the following settings.

Code

```

this.fpSpread1.SpreadScaleMode = FarPoint.Win.Spread.ScaleMode.ZoomDpiSupport;
this.AutoScaleDimensions = new System.Drawing.SizeF(96F, 96F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Dpi;

```

Spreadsheet Objects

You can customize the appearance of various parts of the Spread component. Roughly speaking, the objects in the Spread Windows Forms component fall into two categories: objects that represent real world things in a spreadsheet, like column, rows, and cells, and objects that are conceptual representations of underlying pieces of the spreadsheet, such as data and a grid.

The real world objects can be accessed through a built-in set of shortcut objects. The shortcut objects help you interact with the Spread Windows Forms component in a way that is probably familiar to you from working with other components or applications. The conceptual objects, the more abstract objects, are referred to as "models." These models are responsible for managing the style information, formatting, and data in the Spread component.

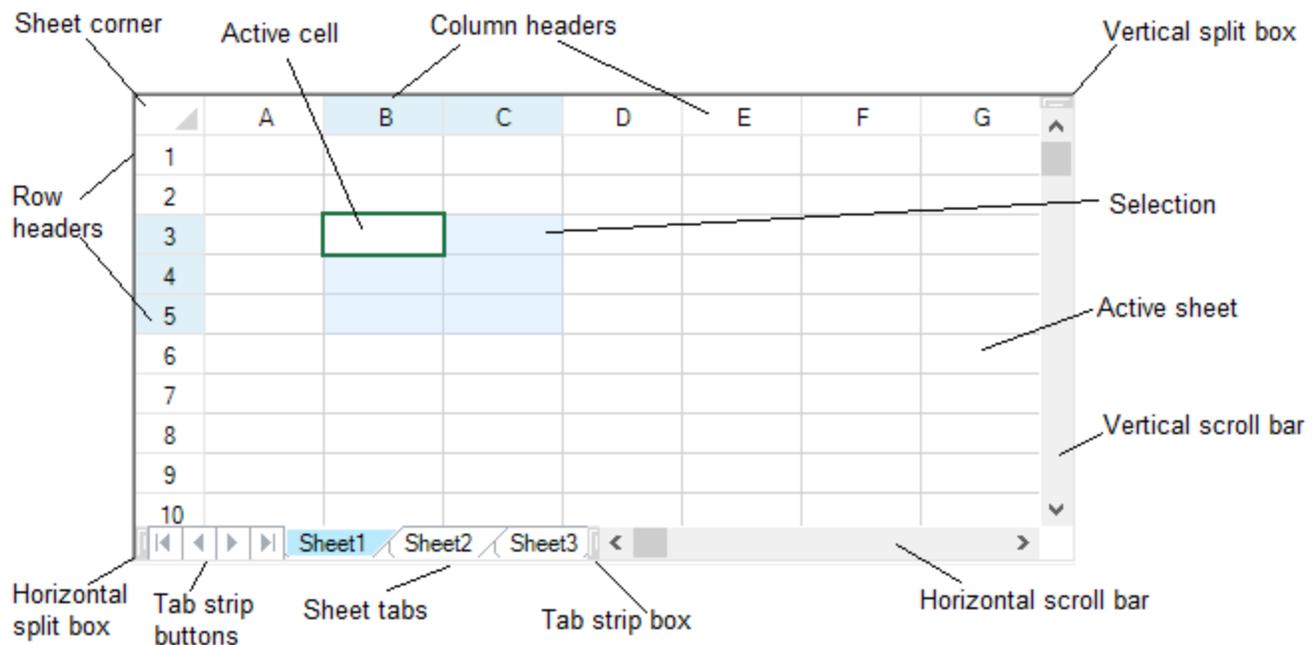
In actuality, the shortcut objects call the model objects. However, the shortcut objects allow you to interact with the Spread Windows Forms component without dealing too much with the underlying object models. If you are new to working with Spread, or are new to developing in an object-oriented environment, you might want to use the shortcut objects at first, as you become familiar with features of Spread Windows Forms. However, intensive use of the shortcut objects can degrade your application's performance.

This topic describes about the Spread components. It includes:

- **Understanding Parts of the Component**
- **Object Parentage**
- **Resetting Parts of the Interface**
- **Rich Text Editing**
- **Improving Performance by Suspending the Layout**
- **Allowing User Functionality**
- **Allowing the User to Zoom the Display of the Component**
- **Adding a Status Bar**
- **Customizing Viewports**
- **Customizing Split Boxes**
- **Working With Slicers**
- **Sheets**
- **Rows and Columns**
- **Headers**
- **Cells**

Understanding Parts of the Component

The Spread component is made up of the spreadsheet that displays the data along with scroll bars and, if multiple sheets, sheet tabs in a tab strip. The figure below shows the major parts of the Spread component. Several of these can be hidden, but this shows the default display.



For more information on ...

- Sheet
- Sheet Corner
- Sheet Tab
- Cell
- Scroll bar
- Column/Row Header
- Row and Column
- Focus Indicator (Active cell)
- Selection
- Active Sheet
- Horizontal/vertical split box

Refer to ...

- Sheets**
- Customizing the Sheet Corner Appearance**
- Customizing the Sheet Name Tabs**
- Cells**
- Customizing the Scroll Bars**
- Headers**
- Rows and Columns**
- Customizing the Focus Indicator for a Cell**
- Customizing User Selection and Deselection of Data**
- Working with the Active Sheet**
- Customizing Split Boxes**

Object Parentage

For the objects in the Spread component, such as the sheet, column, and cell, there are formatting and other properties that each object inherits from what is called its "parent." A cell may inherit some formatting, for example the background color, from the sheet. If you set the alignment of text for all the cells in a column, the cell inherits that as well. Because of this object parentage, many properties and methods can be applied in different ways to different parts of a spreadsheet.

Of course, you can override the formatting that an individual cell inherits. But by default, objects inherit properties from their parents. So in a given context, the settings of any object are the composite of the settings of its parents that are being applied to that object. For example, you may set the text color for a cell at the cell level, but it may inherit the vertical alignment from the row and the border from its column, and the background color from the sheet. Since the background color may be set at several of these levels, certain rules of precedence must apply.

The closer to the cell level, the higher the precedence. So if you set the background color of the cell, the settings inherited from the parents are overridden. Refer to the list to see the order of precedence of these properties. The closer to the cell (the lower the number) the higher the precedence.

1. Cell
2. Row
3. Column
4. Alternating Row
5. Sheet
6. Component

Although formatting and other attributes inherit the object from its "parent", the values of the properties of the individual objects themselves are not updated. For example, even if you change the text color (ForeColor property) of a column (Column class) to red ("Red"), the value of the ForeColor property of the cell (Cell class) on that column does not change by its own. If you want to check the formatting etc. that will eventually be applied to the cell, you can use **GetStyleInfo ('GetStyleInfo Method' in the on-line documentation)** method of **SheetView ('SheetView Class' in the on-line documentation)** class.

Resetting Parts of the Interface

You can reset various settings on various parts of the Spread component interface back to default or original values. You can also clear parts of the data area of various items, both data and formatting.

The ways in which parts of the component can be reset include:

- Reset the component to its original state using the **FpSpread ('FpSpread Class' in the on-line documentation)** class **Reset ('Reset Method' in the on-line documentation)** method.
- Reset the size of the component to its original size using the **FpSpread ('FpSpread Class' in the on-line documentation)** class **DefaultSize** property.
- Reset the sheet to its original state using the **SheetView ('SheetView Class' in the on-line documentation)** class **Reset ('Reset Method' in the on-line documentation)** method.
- Reset the skin properties for a sheet or sheets using the **DefaultSkins ('DefaultSkins Class' in the on-line documentation)** class **Reset ('Reset Method' in the on-line documentation)** method.
- Reset the value of a cell or the text in a cell to empty using the **Cell ('Cell Class' in the on-line documentation)** class, **ResetText ('ResetText Method' in the on-line documentation)** or **ResetValue ('ResetValue Method' in the on-line documentation)** method.
- Reset all the named style properties to their default values using the **NamedStyle ('NamedStyle Class' in the on-line documentation)** class **Reset ('Reset Method' in the on-line documentation)** method. There are also individual reset methods for each of the settings in a style:
- Reset all the style settings in the **StyleInfo ('StyleInfo Class' in the on-line documentation)** object to the default settings using the **StyleInfo ('StyleInfo Class' in the on-line documentation)** class **Reset ('Reset Method' in the on-line documentation)** method.

Reset the settings for cells, rows, or columns using the individual reset methods for each setting in the **Cell ('Cell Class' in the on-line documentation)** or **Row ('Row Class' in the on-line documentation)** or **Column ('Column Class' in the on-line documentation)** class:

Resetting Desired

Background color for individual cells or for row or column of cells

Border for individual cells or for row or column of cells

Cell type for individual cells or for row or column of cells

Method Name

ResetBackColor ('ResetBackColor Method' in the on-line documentation)

ResetBorder ('ResetBorder Method' in the on-line documentation)

ResetCellType ('ResetCellType Method' in the on-line documentation)

Text font for individual cells or for row or column of cells	ResetFont ('ResetFont Method' in the on-line documentation)
Foreground color for individual cells or for row or column of cells	ResetForeColor ('ResetForeColor Method' in the on-line documentation)
Row height	ResetHeight ('ResetHeight Method' in the on-line documentation)
Cell contents horizontal alignment for individual cells or for row or column of cells	ResetHorizontalAlignment ('ResetHorizontalAlignment Method' in the on-line documentation)
Cell header label	ResetLabel ('ResetLabel Method' in the on-line documentation)
Locked status for individual cells or for row or column of cells	ResetLocked ('ResetLocked Method' in the on-line documentation)
Merge policy for a row or column of cells	ResetMergePolicy ('ResetMergePolicy Method' in the on-line documentation)
Cell note indicator color for individual cells or for row or column of cells	ResetNoteIndicatorColor ('ResetNoteIndicatorColor Method' in the on-line documentation)
Parent style name for individual cells or for row or column of cells	ResetParentStyleName ('ResetParentStyleName Method' in the on-line documentation)
Column resizable	ResetResizable ('ResetResizable Method' in the on-line documentation)
Column sort indicator	ResetSortIndicator ('ResetSortIndicator Method' in the on-line documentation)
Cell contents vertical alignment for individual cells or for row or column of cells	ResetVerticalAlignment ('ResetVerticalAlignment Method' in the on-line documentation)
Row or column visible	ResetVisible ('ResetVisible Method' in the on-line documentation)
Column width	ResetWidth ('ResetWidth Method' in the on-line documentation)

Reset all the named style properties to their default values using the **NamedStyle ('NamedStyle Class' in the on-line documentation)** class **Reset ('Reset Method' in the on-line documentation)** method. There are also individual reset methods for each of the settings in a style:

Method Name

- Reset ('Reset Method' in the on-line documentation)**
- ResetBackColor ('ResetBackColor Method' in the on-line documentation)**
- ResetBorder ('ResetBorder Method' in the on-line documentation)**
- ResetCanFocus ('ResetCanFocus Method' in the on-line documentation)**
- ResetCellType ('ResetCellType Method' in the on-line documentation)**
- ResetEditor ('ResetEditor Method' in the on-line documentation)**
- ResetFont ('ResetFont Method' in the on-line documentation)**

Resetting Desired

- All the properties of the style
- Background color of a cell
- Border around a cell
- Whether the cell can receive focus
- Cell type of the cell
- Editor used for editing the cell
- Font face used in the text of the cell

ResetForeColor ('ResetForeColor Method' in the on-line documentation)	Text (foreground) color in a cell
ResetFormatter ('ResetFormatter Method' in the on-line documentation)	Formatter for formatting the contents of the cell
ResetHorizontalAlignment ('ResetHorizontalAlignment Method' in the on-line documentation)	horizontal alignment of text in a cell
ResetLocked ('ResetLocked Method' in the on-line documentation)	Whether the cell is marked as locked
ResetName ('ResetName Method' in the on-line documentation)	Default name of the style
ResetNoteIndicatorColor ('ResetNoteIndicatorColor Method' in the on-line documentation)	Color of the cell note indicator
ResetNoteStyle ('ResetNoteStyle Method' in the on-line documentation)	Cell note style
ResetParent ('ResetParent Method' in the on-line documentation) (Inherited from StyleInfo)	Parent style name
ResetProperty ('ResetProperty Method' in the on-line documentation)	Specified setting property
ResetRenderer ('ResetRenderer Method' in the on-line documentation)	Renderer for painting the cell
ResetTabStop ('ResetTabStop Method' in the on-line documentation)	Whether the user can set focus to the cell using the Tab key
ResetVerticalAlignment ('ResetVerticalAlignment Method' in the on-line documentation)	Vertical alignment of text in a cell

If you are setting the background color for an individual cell, then **ResetBackColor** resets that cell's background color to its default color. If, instead, you set the background color using the **BackColor** property for a **Row** object, then you must use the **ResetBackColor** for that row. If you want to loop all the cells setting the color to White, you can speed this up by invalidating the painting in the Spread while you are looping the cells as shown in the following code.

Visual Basic

```
fpSpread1.SuspendLayout()
fpSpread1.Sheets(0).Cells(0, 0, 499, 499).BackColor = Color.White
fpSpread1.ResumeLayout(True)
```

C#

```
fpSpread1.SuspendLayout();
fpSpread1.Sheets[0].Cells(0, 0, 499, 499).BackColor = Color.White;
fpSpread1.ResumeLayout(true);
```

Resetting the component or a sheet to its default settings returns the component or the sheet to its initial state prior to any design-time or run-time changes. It clears data, resets colors, and returns cells to the default cell type. Resetting the component resets everything in the component to the state when the component is first drawn on the form.

 **Note:** Resetting the component or a sheet clears the data in the sheet(s) as well as the formatting. If you provide a way for users to reset their sheet(s), be sure to have them confirm the action before resetting the sheet(s).

You can find out the default size of the component using the **FpSpread DefaultSize** property.

For more information about clearing data, refer to **Removing Data from a Sheet**.

For information on resetting values creating during the design using Spread Designer, refer to the explanation of this procedure in the **Spread Designer Guide (on-line documentation)**.

Rich Text Editing

The **Rich Text Editing** feature in Spread for WinForms helps to render different text styles and add formulas in the worksheet. The **IFeatures** interface provides the **RichTextEdit** property to indicate whether the rich text is editable or not.

2	
3	The quick brown fox jumps over the lazy dog
4	The quick brown fox jumps over the lazy dog
5	The quick brown fox jumps over the lazy dog
6	The quick brown fox jumps over the lazy dog
7	

The **FarPoint.Win.Spread** namespace provides the **RichEditMode** enumeration to indicate whether you can edit rich text in Spread worksheet. This enumeration provides three values:

- **Legacy:** Value in the cell is edited as plain text.
- **On:** Value in the cell is edited as rich text.
- **Unspecified:** Rich text editing mode is determined via **FarPoint.Win.Spread.FpSpread.EditModeStarting** event.

To use the rich text editing feature, enable the **flat style** mode in Spread, and set the **RichEditMode** to On.

C#

```
// Flat style
fpSpread1.LegacyBehaviors = LegacyBehaviors.None;
// Enable rich text editing
fpSpread1.Features.RichTextEdit = RichEditMode.On;
```

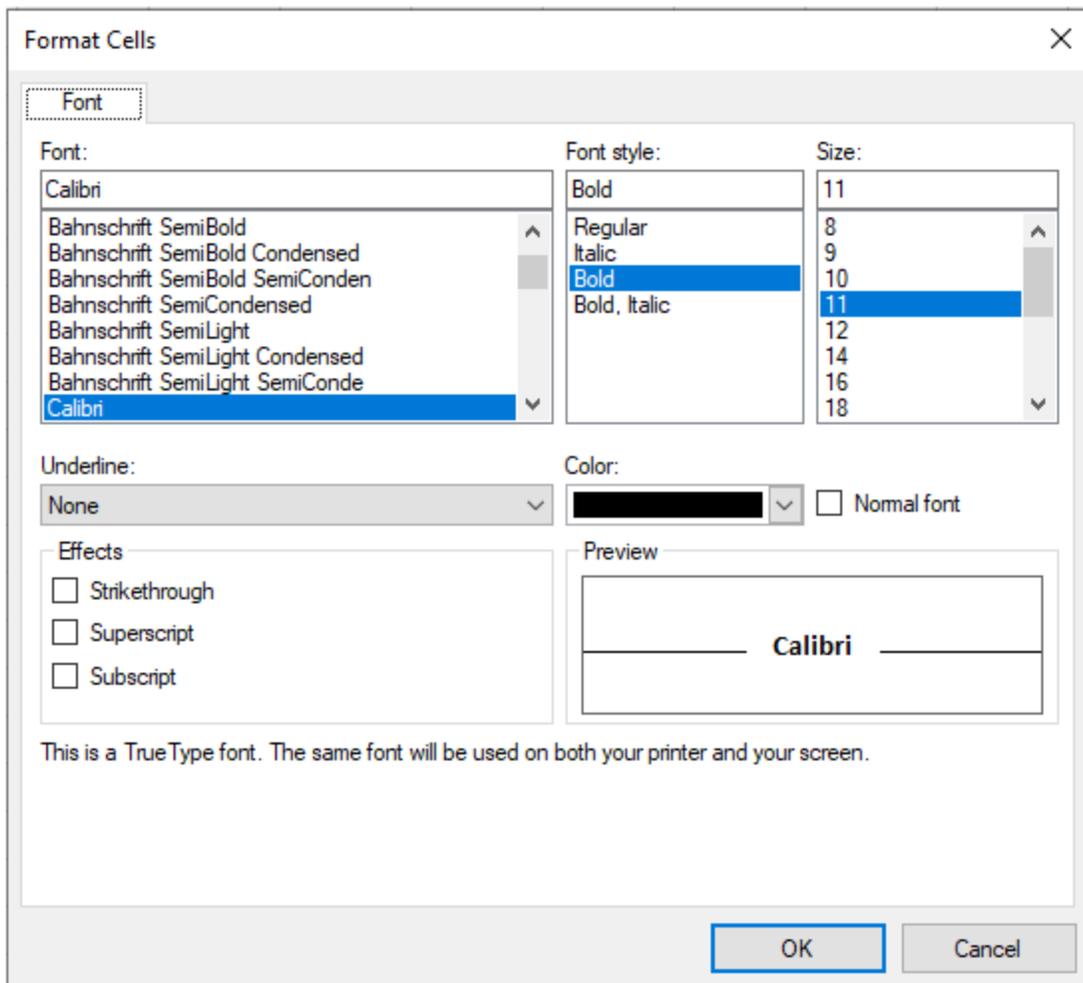
Font Dialog

You can show the built-in Font editor dialog to edit font settings for the currently selected text using the **FormatCells** method of the **BuiltInDialogs** class. To learn more about the dialog, see **Working with BuiltInDialogs**.

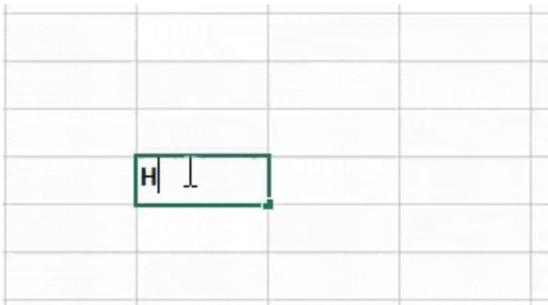
C#

```
if (fpSpread1.EditingControl is IRichTextEditor richTextEditor)
{
    BuiltInDialogs.FormatCells(richTextEditor);
}
```

You can manipulate the font of the text while editing a cell in the worksheet using the **Format Cells** dialog at runtime, for example, you can change the font type, font style, text size, text color, text effects, etc.



Spread allows you to manipulate the rich text editor using the **IRichTextEditor** interface. Here is an example of toggling the bold settings when the user edits a cell.



C#

```
private void ToggleBold(FpSpread spread)
{
    if (spread.EditingControl is IRichTextEditor richTextEditor)
    {
        GrapeCity.Spreadsheet.Font font = GrapeCity.Spreadsheet.Font.Empty;
        font.Bold = !richTextEditor.SelectionFont.Bold;
        richTextEditor.ApplyFont(font, -1, -1); //apply to current selected text
    }
}
```

```
}  
}
```

Insert Function dialog

You can show the Insert Function dialog using the **ToggleInsertFunction** property of the **BuiltInDialogs** class. To learn more about the dialog, see **Working with BuiltInDialogs**.

The Spread Designer UI provides many rich text editing features like Intellisense support, auto-complete, pick from dropdown list, etc. You can learn more about the rich text editing feature in Spread Designer from the **Rich Text Editing (on-line documentation)** topic in the **Designing in the Data Area** section.

Improving Performance by Suspending the Layout

One way to improve the performance of the control, if there are changes to many cells, is to hold the repainting until all the changes are complete. By holding the repainting (suspending the layout) while all the changes and recalculations are done, and then resuming the layout and repainting all the cells, the control can save a lot of time and still deliver a refreshed interface to the user.

The following describes the layout and its suspension.

Layout Objects

A layout is an object that stores calculated values (mostly width and height of cells, spans, and viewports) used for painting the control in its current state. This may include how many viewports there are, what the top left cell in each viewport is, how big each column and row is and how many are currently visible in each viewport, etc. The purpose of the layout object is to optimize painting of the control by storing the calculated layout values used during painting and reusing them each time the control repaints instead of recalculating them each time. When a change is tracked that requires the layout object to be regenerated, the existing objects are discarded and a new one is calculated by the paint code. The layout objects are not part of the public API, but they cache all of the layout information required to paint the sheet, like the column widths, row heights, cell spans, cell overflows and the rectangles of cell notes that are always visible (**Cell ('Cell Class' in the on-line documentation)**.**NoteStyle ('NoteStyle Property' in the on-line documentation)** = NoteStyle.StickyNote).

Suspending the Layout Logic

To improve performance, you can suspend the layout, which stops the layout object from being updated and thus the component does not spend any time making calculations for repainting until the layout is resumed. Two methods accomplish this, the **SuspendLayout ('SuspendLayout Method' in the on-line documentation)** and **ResumeLayout ('ResumeLayout Method' in the on-line documentation)** methods in the **FpSpread ('FpSpread Class' in the on-line documentation)** class. Be sure to use the two methods together within a particular scope of operation, otherwise a problem may occur with the layout being suspended and not able to resume.

The **SuspendLayout ('SuspendLayout Method' in the on-line documentation)** method prevents the component from recomputing the layout of columns, rows, and cells when changes are made to the sheet. If you are making lots of changes to the sheet in a block of code, using **SuspendLayout** prevents the component from doing redundant intermediate recalculations of the layout objects as each change is made, and using **ResumeLayout ('ResumeLayout Method' in the on-line documentation)** (true) recomputes the layout once after all of your changes are made. This approach increases performance greatly, but there are additional approaches you can do depending on what features your sheets require, as described in the section, "Other Performance Improvements."

Suspended Notification

If the layout is suspended without a corresponding resume method in the same scope and an exception occurs, the

component displays a notification, as shown in the following figure. If the state of the component changes such that the layout object contains invalid data (usually the incorrect number of items) then an exception can result when the component tries to paint with the invalid layout data. The notification is shown whenever there is an unhandled exception that occurs during the painting of the control, and the layout is suspended when the exception occurred.



This should only happen when the layout is suspended with `SuspendLayout`, and then changes are made to the component state and the component somehow made to paint again with an invalid layout object. It is possible that there could be an exception that causes this message to be displayed that is not related to the layout being suspended, for example, if an exception is thrown by a custom cell type object during a call to `IRenderer.PaintCell`. Any changes made to the component state could trigger layout recalculation, but not all changes do so. Changes that rearrange rows or columns, such as sorting and filtering, definitely require it, but when setting text only, the layout is recalculated under certain circumstances, such as when **AllowCellOverflow ('AllowCellOverflow Property' in the on-line documentation)** property of the `FpSpread` class is enabled. If the layout is suspended, but the Spread is able to paint using old layout information without any problems, then the Spread may act in unexpected ways, for example, it will not scroll when you try, but the notification is not displayed.

Using the Methods Together

A rough outline of the code for suspending layout would be:

```
SuspendLayout
```

```
    Insert your code here
```

```
ResumeLayout
```

The methods are intended for temporarily ignoring changes to the layout so that many changes can be made without performing the redundant layout recalculations between each change. While layout calculation is suspended, event handlers tracking changes to the component are not able to recalculate the layout and the paint code does not access the new layout. For a nested loop that makes a change to every cell, say changing a value in each cell, this is a case that would definitely benefit from suspending the layout before and resuming the layout afterward.



- Do not use these methods unless the changes are such that the performance can benefit from the layout being suspended temporarily.
- Always use the two methods together in the same scope, otherwise the component might not paint correctly if **SuspendLayout ('SuspendLayout Method' in the on-line documentation)** is called without a matching call to **ResumeLayout ('ResumeLayout Method' in the on-line documentation)** in the same scope.

Sample Code

The following sample code is an example of temporarily disabling the painting of the control and setting the background color to the cell.

C#

```
fpSpread1.SuspendLayout();
```

```
fpSpread1.Sheets[0].Cells[0, 0, 499, 499].BackColor = Color.White;
fpSpread1.ResumeLayout(true);
```

Visual Basic

```
FpSpread1.SuspendLayout()
FpSpread1.Sheets(0).Cells(0, 0, 499, 499).BackColor = Color.White
FpSpread1.ResumeLayout(True)
```

Ways to Improve Performance

You can use the following methods to improve the performance of Spread Winforms control.

- When you do not want to display cell note (**NoteStyle ('NoteStyle Property' in the on-line documentation)** property of **Cell ('Cell Class' in the on-line documentation)** object is set to something other than StickyNote), set the AutoUpdateNotes property of SheetView class to False. By doing so, this will prevent checks on cell note of the StickyNote type, which will allow you to change or move the display state.
- If you set the **AllowCellOverflow ('AllowCellOverflow Property' in the on-line documentation)** property of the FpSpread class to enable, you can accelerate layout recalculation by disabling it. With this feature enabled, a large amount of text width calculation is performed each time cell data changes.
- When using a formula, set the **SheetView ('SheetView Class' in the on-line documentation)** class's **AutoCalculation ('AutoCalculation Property' in the on-line documentation)** property to False, update the sheet, set it back to True, and call the Recalculate method. You can eliminate the redundant intermediate process of recalculating formulas each time you update.
- Since layout objects calculate only the visible part of the sheet, you can improve performance by reducing the control size or reducing the number of columns and rows displayed at one time.
- It is also effective to implement custom sheet model object (such as custom data model object that implements the **ISheetDataModel ('ISheetDataModel Interface' in the on-line documentation)** interface) with unnecessary feature removed. For example, if you do not need the data binding feature, you do not need to implement the data binding interface.

 **Note:** You can also improve the performance by suspending layout logic. For more information, refer **Improving Performance by Suspending the Layout**.

Improve Performance Using CacheOptions

The performance of following formula functions can be improved with exact match.

- [XLOOKUP](#)
- [VLOOKUP](#)
- [HLOOKUP](#)
- [MATCH](#)
- [XMATCH](#)

The **CacheOptions** enumeration in **CalculationEngine** class can be used to define the type of caching while using these functions:

- **On:** (default) Caching is used. Cached data has short life in memory.
- **Aggressive:** Caching is used as much as possible. Cached data has a longer life in memory.
- **None:** No caching is used.

The performance is improved in the following order:

Aggressive > On > None

The following example applies Aggressive CacheOptions option and records the time taken to load the imported Excel file.

C#

```
fpSpread1.AsWorkbook().WorkbookSet.CalculationEngine.CacheOptions =
CacheOptions.Aggressive;
// Sample Excel file containing XMATCH function
fpSpread1.OpenExcel("bigTable - XMatch.xlsx");
Stopwatch stopwatch = new Stopwatch();
stopwatch.Start();
// We check the performance by measuring the time to invoke Calculate only
fpSpread1.AsWorkbook().WorkbookSet.CalculationEngine.Calculate(fpSpread1.AsWorkbook(),
true);
stopwatch.Stop();
long milliseconds = stopwatch.ElapsedMilliseconds;
listBox1.Items.Add(milliseconds.ToString());
```

Allowing User Functionality

There are several aspects of the Spread component that can be set to allow or restrict how the user can interact with the component.

Here is a list summarizing the things you can allow the user to do (or prevent the user from doing) with the data area of the component:

User functionality to allow	Related property or method
Drag and drop cell	FpSpread ('FpSpread Class' in the on-line documentation).AllowDragDrop ('AllowDragDrop Property' in the on-line documentation) property
Drag and fill cell	FpSpread.AllowDragFill ('AllowDragFill Property' in the on-line documentation) property
Edit cell notes	SheetView ('SheetView Class' in the on-line documentation).AllowNoteEdit ('AllowNoteEdit Property' in the on-line documentation) property
Input formulas	FpSpread.AllowUserFormulas ('AllowUserFormulas Property' in the on-line documentation) property
Filter rows	Column ('Column Class' in the on-line documentation).AllowAutoFilter ('AllowAutoFilter Property' in the on-line documentation) property
Expand or collapse hierarchy	SheetView.GetRowExpandable ('GetRowExpandable Method' in the on-line documentation), SetRowExpandable ('SetRowExpandable Method' in the on-line documentation) method
Movement of hands on clock face	FarPoint.PluginCalendar.WinForms.FpClock.AllowMoveHands property
Move rows and columns	FpSpread.AllowRowMove ('AllowRowMove Property' in the on-line documentation) property and AllowColumnMove ('AllowColumnMove Property' in the on-line documentation) property
Perform a standard search	FpSpread.SearchWithDialog ('SearchWithDialog Method' in the on-line documentation) method
Perform an	FpSpread.SearchWithDialogAdvanced ('SearchWithDialogAdvanced Method' in

advanced search	the on-line documentation) method
Resize rows or columns	Column. Resizable (' Resizable Property ' in the on-line documentation) property and Row (' Row Class ' in the on-line documentation). Resizable (' Resizable Property ' in the on-line documentation) property
Sort by clicking the sort indicator in the column header	Column. AllowAutoSort (' AllowAutoSort Property ' in the on-line documentation) property
Filtering a range	Cells. AutoFilter method
Access filter settings	Worksheet. AutoFilter property, Worksheet.Tables. AutoFilter property
Sorting a range	Sort. Apply method

Here is a list summarizing the things you can allow the user to do (or prevent the user from doing) with the component:

User functionality to allow	Related property or method
Restrict access to rows or columns	SheetView. RestrictColumns (' RestrictColumns Property ' in the on-line documentation) property and RestrictRows (' RestrictRows Property ' in the on-line documentation) property
Zoom, or scale the display of the component	FpSpread. AllowUserZoom (' AllowUserZoom Property ' in the on-line documentation) property
Use of Clipboard shortcuts (keys)	SuperEditBase (' SuperEditBase Class ' in the on-line documentation). AllowClipboardKeys (' AllowClipboardKeys Property ' in the on-line documentation) property
Edit the sheet name	TabStrip (' TabStrip Class ' in the on-line documentation). Editable (' Editable Property ' in the on-line documentation) property

Here is a list of things you can allow the user to do (or prevent the user from doing) with the shapes (on the drawing space layer):

User functionality to allow	Related property or method
Move shapes	PSObject (' PSObject Class ' in the on-line documentation). CanMove (' CanMove Property ' in the on-line documentation) property
Resize shapes	PSObject. CanSize (' CanSize Property ' in the on-line documentation) property
Rotate shapes	PSObject. CanRotate (' CanRotate Property ' in the on-line documentation) property

Allowing the User to Zoom the Display of the Component

You can allow the user to change the scale of the display of the Spread component, in other words to zoom in or zoom out.

Using the **AllowUserZoom** ('**AllowUserZoom Property**' in the on-line documentation) property of the **FpSpread** ('**FpSpread Class**' in the on-line documentation) class, allows the user to zoom in or out by pressing the Ctrl key and turning the mouse wheel. The user can zoom in up to 400% and out to 10% of the default display. The scroll bars are unaffected by zooming; only the corner, headers, and data area change their appearance with zooming.

Using the Spread Designer

1. Select the **View** menu and then the **Zoom** option.
2. Set the percentage.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit the Spread Designer.

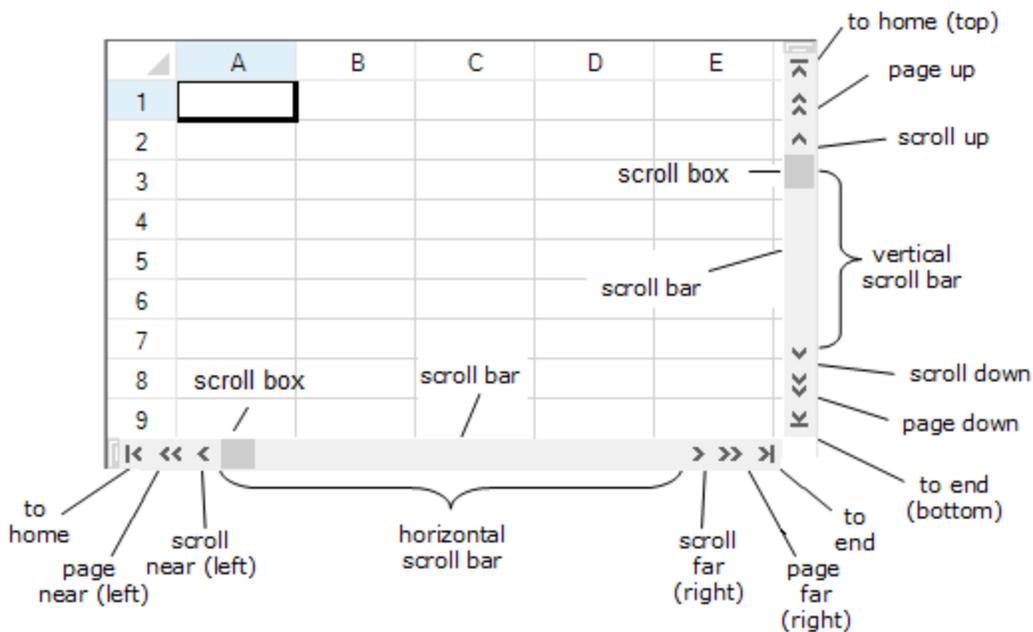
Working with Scroll Bars

The following topics explain about customizing the scroll bars and its operation on the spreadsheet.

- **Customizing the Scroll Bars**
- **Customizing the Position in the Display**
- **Customizing Scroll Bar Tips**

Customizing the Scroll Bars

You can customize the display and operation of the scroll bars on the spreadsheet. The parts of the scroll bars are shown in the following figure.



Setting scroll bar

You can use the properties of the **FpSpread ('FpSpread Class' in the on-line documentation)** class to apply scroll bars settings for the entire component. Also, to set the scroll bars for the viewports, use the properties in the **SpreadView ('SpreadView Class' in the on-line documentation)** class. The table below links to properties in the FpSpread class.

Property in FpSpread

HorizontalScrollBarPolicy

Description

When to display either a vertical or horizontal scroll bar or

<p>(HorizontalScrollBarPolicy Property' in the on-line documentation) VerticalScrollBarPolicy (VerticalScrollBarPolicy Property' in the on-line documentation)</p>	<p>both on the edges of the sheet in the component</p>
<p>HorizontalScrollBarHeight (HorizontalScrollBarHeight Property' in the on-line documentation) VerticalScrollBarWidth (VerticalScrollBarWidth Property' in the on-line documentation)</p>	<p>Dimensions of the scroll bars</p>
<p>ScrollBarTrackPolicy (ScrollBarTrackPolicy Property' in the on-line documentation)</p>	<p>Whether the spreadsheet scrolls across the display when the user moves the scroll box (tracking)</p>
<p>ScrollBarShowMax (ScrollBarShowMax Property' in the on-line documentation)</p>	<p>Whether scroll bars are based on only the area that has data or on the entire spreadsheet</p>
<p>ScrollBarMaxAlign (ScrollBarMaxAlign Property' in the on-line documentation)</p>	<p>Whether to align the scroll bars with the last row and column</p>
<p>TabStripRatio (TabStripRatio Property' in the on-line documentation)</p>	<p>To set the width of the horizontal scroll bar, set the tabstrip ratio. Default is 0.50 (divide the area by 50% with tab strip and scroll bar)</p>

Scroll bar width

To increase the width of the scroll bar, for example, you could change the tab strip ratio to 0.25, which would divide the area between 25% for the tab strip and 75% for the scroll bar, and allocate an area larger than the default to the scroll bar. For more information on the tab strip, refer to **Customizing the Sheet Name Tabs**.

Event

There are two events that indicate that the end user has moved the scroll bars. The **TopChange** (**TopChange Event' in the on-line documentation**) event is raised when the end user moves the vertical scroll bar. The **LeftChange** (**LeftChange Event' in the on-line documentation**) event is raised when the horizontal scroll bar is moved. There is no event raised to indicate that the user has resized the tab strip.

Smooth Scrolling

By default, scrollbars scroll the sheet by row and column. But, you can set smooth scrolling by setting the **VerticalScrollBarMode** (**VerticalScrollBarMode Property' in the on-line documentation**) and **HorizontalScrollBarMode** (**HorizontalScrollBarMode Property' in the on-line documentation**) properties of the **FpSpread** (**FpSpread Class' in the on-line documentation**) class. You can set the following values for the VerticalScrollMode and HorizontalScrollMode enumeration.

Values for VerticalScrollMode and HorizontalScrollMode enumeration	Description
Row Column	Scroll by row or column.
Pixel	Scroll in pixel.
PixelEnhanced	For vertical scroll: Scroll in pixel (same as 'Pixel' value) as well as scroll in pixel

while using mouse wheel operation.

For horizontal scroll: Scroll in pixel (same as 'Pixel' value)

The number of pixels scrolled by mouse wheel in vertical scroll is calculated by the expression:

$$\text{VerticalScrollBarSmallChange} * \text{MouseWheelScrollLine}$$

(The default value of MouseWheelScrollLine is 3)

PixelAndColumn
PixelAndRow

While scrolling in pixel with the mouse, scroll by row and column when you release the mouse button.



- You can display scroll bar tips. For more information, kindly refer to "**Customizing Scroll Bar Tips**".
- Scroll bar is also a type of control and inherit the benefits of controls. For example, context menus are available for horizontal and vertical scrollbars, which are enabled by default.
- By default, the scroll bar does not display page button, home button, or end button. For information regarding the method to display all scroll bar buttons, kindly refer to the sample code.
- You can customize the display and operation of the scroll bar by using each member of the **EnhancedScrollBarRenderer ('EnhancedScrollBarRenderer Class' in the on-line documentation)** class that represents the scroll bar renderer.

Sample Code

The following sample code sets the scroll bar of the control. Scroll bars appear larger than the default size. For scrolling horizontally, the spreadsheet scrolls as you move the scroll box; for scrolling vertically, the scroll bar tip shows the row number, but the spreadsheet does not scroll until you are done.

C#

```
FarPoint.Win.Spread.FpSpread fpSpread1 = new FarPoint.Win.Spread.FpSpread();
FarPoint.Win.Spread.SheetView shv = new FarPoint.Win.Spread.SheetView();
fpSpread1.Location = new Point(10, 10);
fpSpread1.Height = 250;
fpSpread1.Width = 400;
Controls.Add(fpSpread1);
fpSpread1.Sheets.Add(shv);
fpSpread1.HorizontalScrollBarPolicy = FarPoint.Win.Spread.ScrollBarPolicy.Always;
fpSpread1.VerticalScrollBarPolicy = FarPoint.Win.Spread.ScrollBarPolicy.AsNeeded;
// Setting vertical bar pixel change
fpSpread1.VerticalScrollBarSmallChange = 10;
// Setting Vertical scroll mode to PixelEnhanced to support pixel scrolling through
mouse
fpSpread1.VerticalScrollBarMode = FarPoint.Win.VerticalScrollMode.PixelEnhanced;
fpSpread1.HorizontalScrollBarHeight = 30;
fpSpread1.VerticalScrollBarWidth = 30;
fpSpread1.ScrollBarMaxAlign = true;
fpSpread1.ScrollBarShowMax = true;
fpSpread1.ScrollBarTrackPolicy = FarPoint.Win.Spread.ScrollBarTrackPolicy.Horizontal;
fpSpread1.ScrollTipPolicy = FarPoint.Win.Spread.ScrollTipPolicy.Vertical;
```

Visual Basic

```
Dim FpSpread1 As New FarPoint.Win.Spread.FpSpread()
Dim shv As New FarPoint.Win.Spread.SheetView()
FpSpread1.Location = New Point(10, 10)
```

```

FpSpread1.Height = 250
FpSpread1.Width = 400
Controls.Add(FpSpread1)
FpSpread1.Sheets.Add(shv)
FpSpread1.HorizontalScrollBarPolicy = FarPoint.Win.Spread.ScrollBarPolicy.Always
FpSpread1.VerticalScrollBarPolicy = FarPoint.Win.Spread.ScrollBarPolicy.AsNeeded
// Setting vertical bar pixel change
fpSpread1.VerticalScrollBarSmallChange = 10;
// Setting Vertical scroll mode to PixelEnhanced to support pixel scrolling through
mouse
fpSpread1.VerticalScrollBarMode = FarPoint.Win.VerticalScrollMode.PixelEnhanced;
FpSpread1.HorizontalScrollBarHeight = 30
FpSpread1.VerticalScrollBarWidth = 30
FpSpread1.ScrollBarMaxAlign = True
FpSpread1.ScrollBarShowMax = True
FpSpread1.ScrollBarTrackPolicy = FarPoint.Win.Spread.ScrollBarTrackPolicy.Horizontal
FpSpread1.ScrollTipPolicy = FarPoint.Win.Spread.ScrollTipPolicy.Vertical

```

The following sample code displays all the scroll bar buttons.

C#

```

fpSpread1.VerticalScrollBar.Buttons = FarPoint.Win.Spread.ScrollBarButtons.HomeEnd |
FarPoint.Win.Spread.ScrollBarButtons.PageUpDown |
FarPoint.Win.Spread.ScrollBarButtons.LineUpDown |
FarPoint.Win.Spread.ScrollBarButtons.Thumb;
fpSpread1.HorizontalScrollBar.Buttons = FarPoint.Win.Spread.ScrollBarButtons.HomeEnd |
FarPoint.Win.Spread.ScrollBarButtons.LineUpDown |
FarPoint.Win.Spread.ScrollBarButtons.PageUpDown |
FarPoint.Win.Spread.ScrollBarButtons.Thumb;

```

Visual Basic

```

FpSpread1.VerticalScrollBar.Buttons = FarPoint.Win.Spread.ScrollBarButtons.HomeEnd Or
FarPoint.Win.Spread.ScrollBarButtons.PageUpDown Or
FarPoint.Win.Spread.ScrollBarButtons.LineUpDown Or
FarPoint.Win.Spread.ScrollBarButtons.Thumb
FpSpread1.HorizontalScrollBar.Buttons = FarPoint.Win.Spread.ScrollBarButtons.HomeEnd Or
FarPoint.Win.Spread.ScrollBarButtons.LineUpDown Or
FarPoint.Win.Spread.ScrollBarButtons.PageUpDown Or
FarPoint.Win.Spread.ScrollBarButtons.Thumb

```

Using the Spread Designer

1. From the **Settings** menu, select **Scrollbars**.
2. In the **Scroll Bar** tab, set the display and tracking by selecting the options.
3. Click **OK**.
4. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

or

1. Select the Spread component (or select **Spread** from the pull-down menu).
2. In the property list for the component (in the **Behavior** category), select one of the scroll bar properties.
3. Click the drop-down arrow to display the choices and select a value. Repeat this for each property.
4. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Horizontal Scrolling using Mouse Wheel

Spread provides a convenient way to navigate horizontally within a worksheet using the mouse wheel and Ctrl+ Shift keys.

- **To scroll Right:** Hold the Ctrl + Shift keys and scroll down the mouse wheel.
- **To Scroll Left:** Hold the Ctrl + Shift keys and scroll up the mouse wheel.

This feature is enabled in a worksheet by default. However, if you wish to disable the horizontal scrolling functionality then use the following code in the **MouseWheel** event.

C#

```
private void FpSpread1_MouseWheel(object sender, MouseEventArgs e)
{
    if ((Control.ModifierKeys & (Keys.Control | Keys.Shift)) == (Keys.Control |
Keys.Shift) && e.HandledMouseEventArgs handledMouseEventArgs)
    {
        handledMouseEventArgs.Handled = true;
    }
}
```

Visual Basic

```
Private Sub FpSpread1_MouseWheel(ByVal sender As Object, ByVal e As MouseEventArgs)
    Dim handledMouseEventArgs As HandledMouseEventArgs = Nothing
    If (Control.ModifierKeys And (Keys.Control Or Keys.Shift)) Is (Keys.Control Or
Keys.Shift) AndAlso CSharpImpl.__Assign(handledMouseEventArgs, TryCast(e,
HandledMouseEventArgs)) IsNot Nothing Then
        handledMouseEventArgs.Handled = True
    End If
```

Customizing the Position in the Display

You can customize the position of the data area of the spreadsheet in the display by choosing a row, column, or cell, and moving it to a particular position in the displayed portion of the spreadsheet in the Spread component.

Use the **ShowRow** (**ShowRow Method' in the on-line documentation**), **ShowColumn** (**ShowColumn Method' in the on-line documentation**), **ShowCell** (**ShowCell Method' in the on-line documentation**), or **ShowActiveCell** (**ShowActiveCell Method' in the on-line documentation**) methods in the **FpSpread** (**FpSpread Class' in the on-line documentation**) class, and the **HorizontalPosition** (**HorizontalPosition Enumeration' in the on-line documentation**) and **VerticalPosition** (**VerticalPosition Enumeration' in the on-line documentation**) enumerations. These methods scroll the display until the specified item is displayed in the specified location.

Using Code

Set the **ShowActiveCell** (**ShowActiveCell Method' in the on-line documentation**) method.

Example

This example sets the active cell and then displays the cell in the top, center of the spreadsheet.

C#

```
fpSpread1.Sheets[0].SetActiveCell(3, 4);
fpSpread1.ShowActiveCell(FarPoint.Win.Spread.VerticalPosition.Top,
```

```
FarPoint.Win.Spread.HorizontalPosition.Center);
```

VB

```
fpSpread1.Sheets(0).SetActiveCell(3,4)
fpSpread1.ShowActiveCell(FarPoint.Win.Spread.VerticalPosition.Top,
FarPoint.Win.Spread.HorizontalPosition.Center)
```

Customizing Scroll Bar Tips

As an additional aid to the end users, you can turn on scroll bar tips which, by default, display the row number when the pointer is over the vertical scroll bar and the column number when the pointer is over the horizontal scroll bar. In the following figure, the scroll bar tip shows the column number for horizontal scrolling.



You can display scroll bar tooltip by using **ScrollTipPolicy** ('**ScrollTipPolicy Property**' in the on-line documentation) property of the **FpSpread** ('**FpSpread Class**' in the on-line documentation) class.

You can specify how the scroll bar tips are displayed using the **ScrollingContentInfo** ('**ScrollingContentInfo Class**' in the on-line documentation) class. You can use the **ColumnIndices** ('**ColumnIndices Property**' in the on-line documentation) property of the **ScrollingContentInfo** class to specify the column index of the data to be displayed on the vertical scroll bar tip. If you set the index for a column that displays command button or image cell, then the cells will be displayed in the vertical scroll tip. The following image shows a vertical scroll bar tip with a command button cell.

	A	B	C	D	E	F	G	H
1	test	Region	Sale man	Sale office	Product	Unit	Price	
2	test	North	John	Si	test	Mark	Currently, row 6 is scrolled.	
3	test	East	Duke	Alpha	Piza	90	34	

You can use the **ScrollTipFetch** ('**ScrollTipFetch Event**' in the on-line documentation) event of the **FpSpread** class to customize the text of the row number displayed on the scroll bar tip. Use the **TipText** ('**TipText Property**' in the on-line documentation) property of the **ScrollTipFetchEventArgs** ('**ScrollTipFetchEventArgs Class**' in the on-line documentation) class to represent event argument. In the above image, we use the **ScrollingContentInfo** class which is described above to display the data in the first and third column, and now we are using the **ScrollTipFetch** event to customize the row number text ("Currently, row 24 ...").

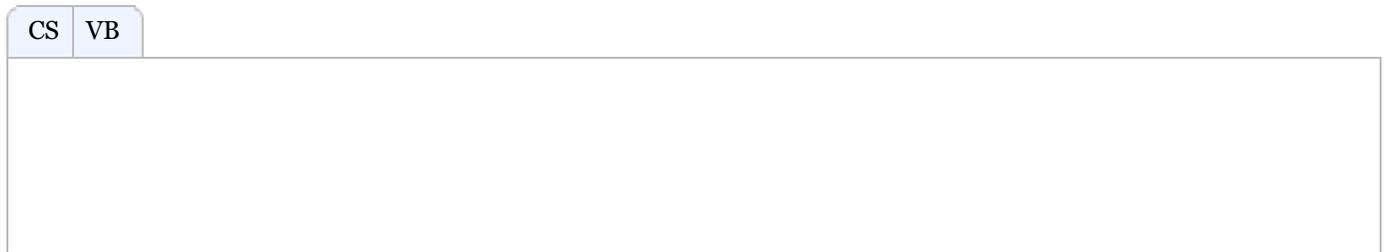
Sample Code

In this example, the scroll bar tip displays custom text for the vertical scroll bar and default text for the horizontal scroll bar.

CS

VB

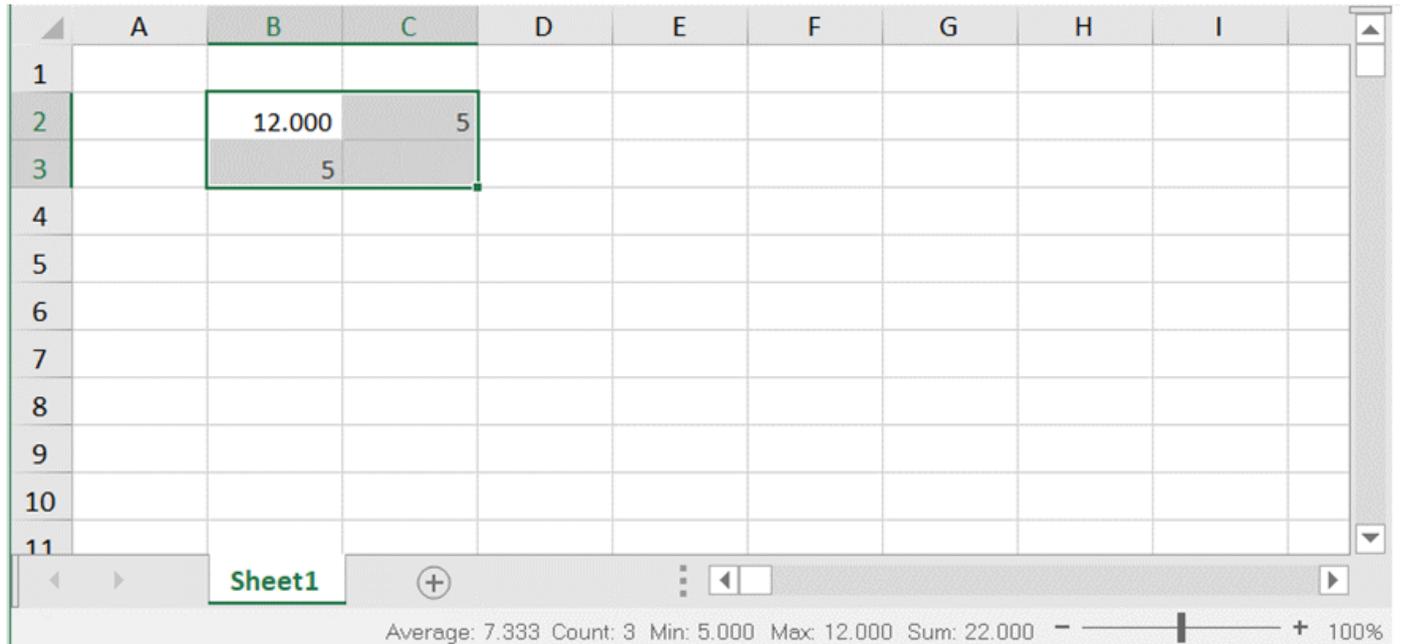
In this example, the scroll bar tip is set using the ScrollingContentInfo class to display data in the first and third column. You can also add code for the ScrollTipFetch event used in the previous sample to customize the row number display text.



Adding a Status Bar

You can add a status bar to the component. The status bar allows you to set zoom settings and displays information such as the average, sum, and counts for selected cells.

The following image displays the sum, average, count, and zoom slider in the status bar.



The StatusBar uses the formatter of the active cell to display the text. It displays the value of the active cell along with its formatter. For example, If the active cell has a NumberCellType with DecimalPlaces set to 3, then in the StatusBar it will display the value with three decimal places.

The following options are available in the status bar.

Option	Description
Average	This option displays the average from the selected cells that contain numerical values.
Count	This option displays the number of selected cells.
Minimum	This option displays the minimum numerical value in the selected cells.
Maximum	This option displays the maximum numerical value in the selected cells.
Numerical	This option displays the number of selected cells that contain numerical values.

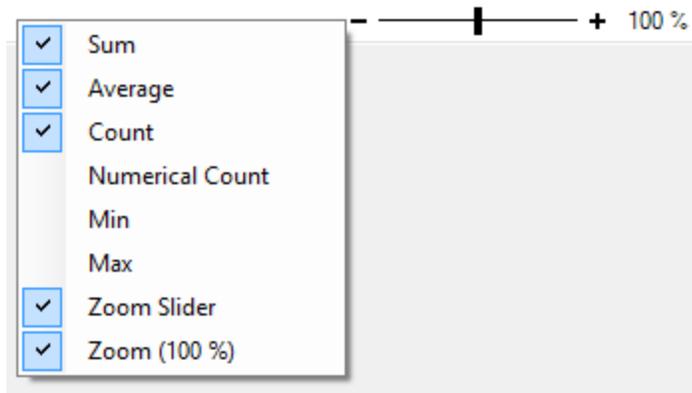
Count

Sum This option displays the sum of numerical values in selected cells.

Zoom (100%) This option displays the zoom level. Select the zoom percentage to display a dialog that allows you to set additional magnification options.

Zoom Slider This option displays a slider with plus and minus buttons for changing the zoom options.

You can right-click on the status bar to display menu options that you can check or uncheck.



Setting Method

You can display the status bar by setting **StatusBarVisible** ('**StatusBarVisible Property**' in the **on-line documentation**) property of **FpSpread** ('**FpSpread Class**' in the **on-line documentation**) class to true.

Sample Code

This example adds a status bar to the component and sets colors for the status bar.

C#

```
fpSpread1.StatusBarVisible = true;
fpSpread1.StatusBar.BackColor = Color.LemonChiffon;
fpSpread1.StatusBar.ZoomSliderColor = Color.Turquoise;
fpSpread1.StatusBar.ForeColor = Color.OliveDrab;
fpSpread1.StatusBar.ZoomButtonHoverColor = Color.Gold;
fpSpread1.StatusBar.ZoomSliderTrackColor = Color.DodgerBlue;
fpSpread1.StatusBar.ZoomSliderHoverColor = Color.BurlyWood;
```

Visual Basic

```
FpSpread1.StatusBarVisible = True
FpSpread1.StatusBar.BackColor = Color.LemonChiffon
FpSpread1.StatusBar.ZoomSliderColor = Color.Turquoise
FpSpread1.StatusBar.ForeColor = Color.OliveDrab
FpSpread1.StatusBar.ZoomButtonHoverColor = Color.Gold
FpSpread1.StatusBar.ZoomSliderTrackColor = Color.DodgerBlue
FpSpread1.StatusBar.ZoomSliderHoverColor = Color.BurlyWood
```

Using the Spread Designer

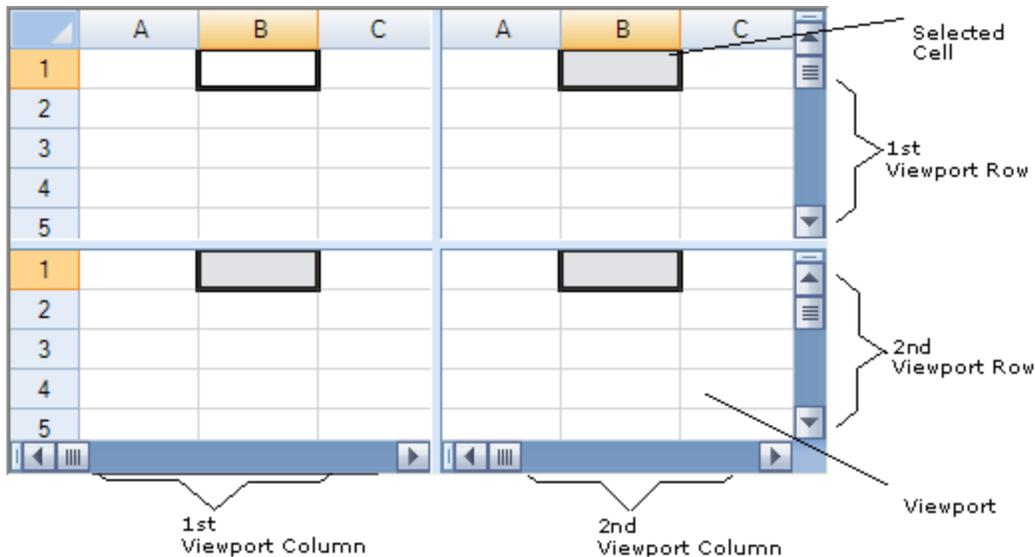
1. Select Spread in the Properties window.
2. Set StatusBarVisible property to true.

Customizing Viewports

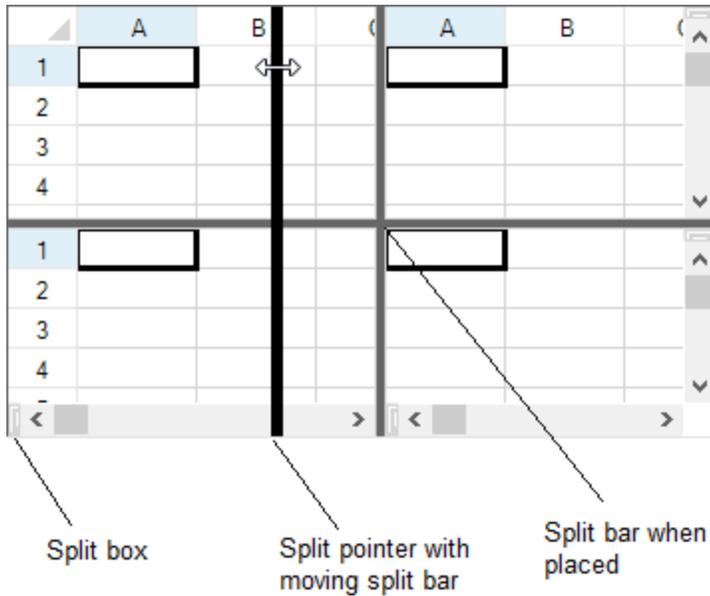
You can divide up the display into separately scrollable viewports. You can set up the following configurations:

- A set of horizontal viewports (called a viewport row since it is a row of viewports)
- A set of vertical viewports (called a viewport column since it is a column of viewports)
- A set of both (as shown in the following figure)

The viewports allow you to display different parts of a very large spreadsheet in a very limited viewing area. You can add, remove, and customize viewports programmatically, and you can allow your end user to create and use viewports.



For an end user to create a viewport, the end user can click on the split box and drag it to the desired location. To allow the end user to divide the display, set the policy for displaying the split boxes. The end user can create multiple viewports in either orientation. You can allow split bars in only one orientation by setting the policy to display only the split box in that orientation. For more information on split boxes, refer to **Customizing Split Boxes**. The figure below shows how to create a viewport.



- The scroll bar must be visible for the split boxes to be accessible. For more information about scroll bars, refer to "**Customizing the Scroll Bars**".
- Each viewport row or viewport column has its own scroll bars.
- The split bars show the border of each viewport. To remove a split bar, the end user can either double-click on the split bar or click and drag it all the way to the edge of the sheet.

There are several properties and methods that relate to the use of viewports; many of these provide customizations programmatically as summarized in this table.

Customization

Add and remove viewports

Set the height and width of viewports

Determine the viewport for a given location in the display

Method or Property in FpSpread (or SpreadView) class

AddViewport ('AddViewport Method' in the on-line documentation)

RemoveViewport ('RemoveViewport Method' in the on-line documentation)

SetViewportPreferredHeight ('SetViewportPreferredHeight Method' in the on-line documentation)

SetViewportPreferredWidth ('SetViewportPreferredWidth Method' in the on-line documentation)

GetViewportPreferredHeight ('GetViewportPreferredHeight Method' in the on-line documentation)

GetViewportPreferredWidth ('GetViewportPreferredWidth Method' in the on-line documentation)

GetViewPortX ('GetViewPortX Method' in the on-line documentation)

GetViewPortY ('GetViewPortY Method' in the on-line documentation)

GetViewPortHeight ('GetViewPortHeight Method' in the on-line documentation)

Determine the row or column of cells in a particular viewport row or viewport column	<p>GetViewportWidth ('GetViewportWidth Method' in the on-line documentation)</p> <p>GetViewportBottomRow ('GetViewportBottomRow Method' in the on-line documentation)</p> <p>GetViewportLeftColumn ('GetViewportLeftColumn Method' in the on-line documentation)</p> <p>GetViewportRectangle ('GetViewportRectangle Method' in the on-line documentation)</p> <p>GetViewportTopRow ('GetViewportTopRow Method' in the on-line documentation)</p> <p>SetViewportLeftColumn ('SetViewportLeftColumn Method' in the on-line documentation)</p> <p>SetViewportTopRow ('SetViewportTopRow Method' in the on-line documentation)</p>
Determine which is the active viewport	<p>GetActiveColumnViewportIndex ('GetActiveColumnViewportIndex Method' in the on-line documentation)</p> <p>GetActiveRowViewportIndex ('GetActiveRowViewportIndex Method' in the on-line documentation)</p> <p>SetActiveViewport ('SetActiveViewport Method' in the on-line documentation)</p>
Determine the index of a particular viewport row or viewport column	<p>GetColumnViewportIndexFromX ('GetColumnViewportIndexFromX Method' in the on-line documentation)</p> <p>GetRowViewportIndexFromY ('GetRowViewportIndexFromY Method' in the on-line documentation)</p>
Determine the number of viewports in either orientation	<p>GetColumnViewportCount ('GetColumnViewportCount Method' in the on-line documentation)</p> <p>GetRowViewportCount ('GetRowViewportCount Method' in the on-line documentation)</p>
Position the viewport in the display	<p>ShowColumn ('ShowColumn Method' in the on-line documentation)</p> <p>ShowRow ('ShowRow Method' in the on-line documentation)</p>

The width and height set in the `SetViewportPreferredXXX` method are suggested size. The Spread component attempts to layout the viewports as close as possible to the suggested sizes. However, if the sum of the suggested sizes is larger (or smaller) than the size of the component then the actual sizes of one or more of the viewports must be larger (or smaller) than the suggested size.

The width and height set by the `SetViewportPreferredXXX` method can set to -1 or a positive number. A -1 indicates a variable size. A positive number indicates a fixed size in pixels. When the component lays out the viewports, available space is first allocated for the fixed-size viewports. Any remaining space is evenly divided among the variable-sized viewports.

The **ColumnViewportWidthChanged ('ColumnViewportWidthChanged Event' in the on-line documentation)** and the **RowViewportHeightChanged ('RowViewportHeightChanged Event' in the on-line documentation)** events are raised any time the split boxes are moved by the end user. These events are not raised when you add a viewport programmatically; they are raised only when the user changes the width or height of a viewport.

Frozen row and column

For frozen rows and columns, the frozen row or column is a separate viewport. The leading frozen row or column is a separate viewport while the trailing frozen row or column is also a separate viewport. Indexes for the frozen viewports are as follows.

Index	Frozen Viewport
-1	Leading frozen
0	First scrollable
1	Second scrollable
$n-1$	Last scrollable
n	Trailing frozen

For more information about frozen rows and columns, refer to **Setting Fixed (Frozen) Rows or Columns**.

Sample Code

This example adds a viewport and sets its various properties.

C#

```
fpSpread1.AddViewport(1, 2);
fpSpread1.SetViewportLeftColumn(1, 3);
fpSpread1.SetViewportTopRow(0, 6);
fpSpread1.SetViewportPreferredHeight(0, 100);
fpSpread1.SetViewportPreferredWidth(0, 100);
```

Visual Basic

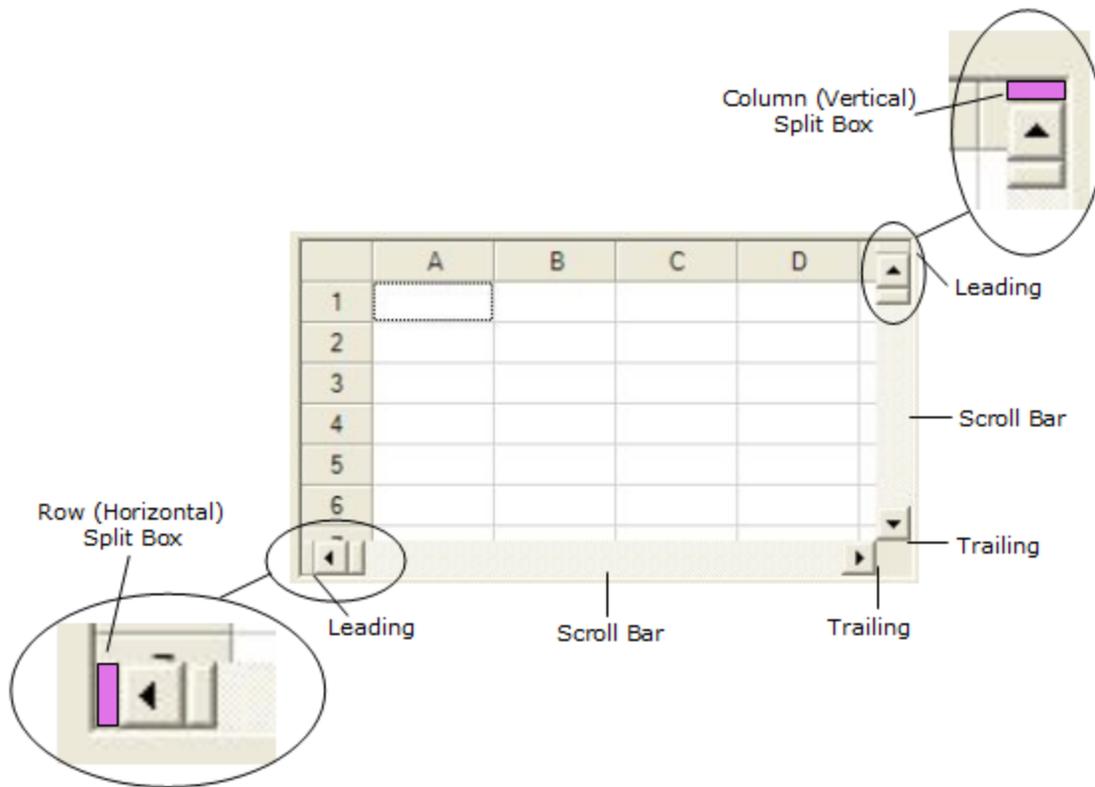
```
FpSpread1.AddViewport(1, 2)
FpSpread1.SetViewportLeftColumn(1, 3)
FpSpread1.SetViewportTopRow(0, 6)
FpSpread1.SetViewportPreferredHeight(0, 100)
FpSpread1.SetViewportPreferredWidth(0, 100)
```

Using the Spread Designer

1. Select the Spread component (or select Spread from the pull-down menu).
2. In the main window, select the split boxes from the edge of the scroll bars and drag them each to the position to create the viewports as needed.
3. From the **Settings** menu, choose **Preferences** and select **Save Split Bars on Apply** to allow Spread Designer to save these viewports when you apply the changes. Some developers create split bars for viewing certain aspects of their spreadsheet, but do not want them saved.
4. From the **File** menu, choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Customizing Split Boxes

By default, the split boxes are shown and appear at the leading edge of each of the scroll bars, as shown in the figure below. When the end user clicks and drags the split box, the view is split into separate viewports. For more information on viewports, refer to **Customizing Viewports**.



Shown in purple for emphasis

You can customize the display and placement of the split boxes by using the following properties of the **FpSpread** ('FpSpread Class' in the on-line documentation) class.

Property

ColumnSplitBoxAlignment
(**'ColumnSplitBoxAlignment Property'** in the on-line documentation) property

RowSplitBoxAlignment
(**'RowSplitBoxAlignment Property'** in the on-line documentation) property

ColumnSplitBoxPolicy
(**'ColumnSplitBoxPolicy Property'** in the on-line documentation) property
RowSplitBoxPolicy (**'RowSplitBoxPolicy Property'** in the on-line documentation) property

Description

Placement of split box relative to the scroll bar. Sets the value of **SplitBoxAlignment** (**'SplitBoxAlignment Enumeration'** in the on-line documentation) enumeration.

Whether or when to display the split box. Sets the value of **SplitBoxPolicy** (**'SplitBoxPolicy Enumeration'** in the on-line documentation) enumeration.

Sample Code

In this example, the split box for the columns is at the leading edge of the scroll bar and is displayed as needed; for the rows, the split box is at the trailing edge of the scroll bar and is always displayed.

C#

```
fpSpread1.ColumnSplitBoxAlignment = FarPoint.Win.Spread.SplitBoxAlignment.Leading;
fpSpread1.RowSplitBoxAlignment = FarPoint.Win.Spread.SplitBoxAlignment.Trailing;
fpSpread1.ColumnSplitBoxPolicy = FarPoint.Win.Spread.SplitBoxPolicy.AsNeeded;
```

```
fpSpread1.RowSplitBoxPolicy = FarPoint.Win.Spread.SplitBoxPolicy.Always;
```

Visual Basic

```
FpSpread1.ColumnSplitBoxAlignment = FarPoint.Win.Spread.SplitBoxAlignment.Leading  
FpSpread1.RowSplitBoxAlignment = FarPoint.Win.Spread.SplitBoxAlignment.Trailing  
FpSpread1.ColumnSplitBoxPolicy = FarPoint.Win.Spread.SplitBoxPolicy.AsNeeded  
FpSpread1.RowSplitBoxPolicy = FarPoint.Win.Spread.SplitBoxPolicy.Always
```

Using the Spread Designer

1. From the **Settings** menu, select **SplitBox**.
2. In the **Split Box** tab, set the values for the **ColumnSplitBoxAlignment**, **RowSplitBoxAlignment**, **ColumnSplitBoxPolicy**, or **RowSplitBoxPolicy** property.
3. Click **OK**.
4. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

or

1. Select the Spread component (or select Spread from the pull-down menu).
2. In the property list for the component (in the **Splitters** category), select the **ColumnSplitBoxAlignment**, **RowSplitBoxAlignment**, **ColumnSplitBoxPolicy**, or **RowSplitBoxPolicy** property.
3. Click the drop-down arrow to display the choices and select the value. Repeat this for each property.
4. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Working With Slicers

Spread for WinForms provides support for using slicers (visual filters) in the worksheet. Slicers enable users to perform quick filtration of the data in tables without using drop-down lists.

Users can have multiple "copies" of a slicer operating together on different worksheets. You can select an item or drag across a number of items by using the keyboard shortcuts "Ctrl + Click" or "Shift + Click" to include the filter items and the data for that item will be displayed in the report as per the specific requirements.

Add a Slicer

To add a slicer in the worksheet, users first need to set the value of the **EnhancedShapeEngine** (**'EnhancedShapeEngine Property' in the on-line documentation**) property to "true" in order to enable the new shape engine. After enabling the **EnhancedShapeEngine**, users can either use the **Add()** (**'Add Method' in the on-line documentation**) method of the **ISlicerCaches** (**'ISlicerCaches Interface' in the on-line documentation**) interface or simply click anywhere on the table and select the "Insert" button on the ribbon bar to add a slicer to the worksheet.

A slicer contains a header, a caption, slicer items and a clear button. The following image shows a slicer added to the worksheet.

	A	B	C	D	E	F
1	Name	City	Weight			
2	Bob	NewYork	80	<div style="border: 1px solid black; padding: 5px;"> <p>Name</p> <ul style="list-style-type: none"> Alice Betty Bob </div>		
3	Betty	Chicago	72			
4	Alice	Washington	71			
5						
6						
7						
8						
9						
10						
11						
12						

The following example code shows how to add a slicer in the spreadsheet.

C#

```
// Enable Shape Engine using EnhancedShapeEngine property to use the slicer feature
fpSpread1.Features.EnhancedShapeEngine = true;
// Add the Slicer
private static void AddSlicer(IWorkbook WorkBook, IWorksheet WorkSheet)
{
    // Initialize Data in worksheet
    WorkSheet.Cells[0, 0].Text = "Name";
    WorkSheet.Cells[0, 1].Text = "City";
    WorkSheet.Cells[0, 2].Text = "Weight";
    WorkSheet.Cells[1, 0].Text = "Bob";
    WorkSheet.Cells[1, 1].Text = "NewYork";
    WorkSheet.Cells[1, 2].Value = 80;
    WorkSheet.Cells[2, 0].Text = "Betty";
    WorkSheet.Cells[2, 1].Text = "Chicago";
    WorkSheet.Cells[2, 2].Value = 72;
    WorkSheet.Cells[3, 0].Text = "Alice";
    WorkSheet.Cells[3, 1].Text = "Washington";
    WorkSheet.Cells[3, 2].Value = 71;
    // Create Table
    ITable table = WorkSheet.Tables.Add("A1:C4", YesNoGuess.Yes);

    // Add SlicerCache using the Column index
    ISlicerCache slicerCache = WorkBook.SlicerCaches.Add(table, 0, "slicerCache");

    // Add Slicer to SlicerCache
    ISlicer slicer = slicerCache.Slicers.Add(WorkSheet, "slicer", "Name", 200, 20, 200,
    200);
}
```

Add Multiple Slicers

Users can also add multiple slicers to the worksheet by using the **Add()** ('Add Method' in the on-line **documentation**) method of the **ISlicerCaches** ('ISlicerCaches Interface' in the on-line **documentation**)

interface. The following image shows a spreadsheet with multiple slicers.

	A	B	C	D	E	F	G	H	I	J
1	Name	City	Weight							
2	Bob	NewYork	80	Name		City				
3	Betty	Chicago	72	Alice		Chicago				
4	Alice	Washington	71	Betty		NewYork				
5				Bob		Washington				
6										
7										
8										
9										
10										
11										
12										

The following example code shows how to add multiple slicers in a spreadsheet.

C#

```
// Enable Shape Engine using EnhancedShapeEngine property to use the slicer feature
fpSpread1.Features.EnhancedShapeEngine = true;
private static void AddMultipleSlicer(IWorkbook Workbook, IWorksheet Worksheet)
{
    // Initialize Data in worksheet
    Worksheet.Cells[0, 0].Text = "Name";
    Worksheet.Cells[0, 1].Text = "City";
    Worksheet.Cells[0, 2].Text = "Weight";
    Worksheet.Cells[1, 0].Text = "Bob";
    Worksheet.Cells[1, 1].Text = "NewYork";
    Worksheet.Cells[1, 2].Value = 80;
    Worksheet.Cells[2, 0].Text = "Betty";
    Worksheet.Cells[2, 1].Text = "Chicago";
    Worksheet.Cells[2, 2].Value = 72;
    Worksheet.Cells[3, 0].Text = "Alice";
    Worksheet.Cells[3, 1].Text = "Washington";
    Worksheet.Cells[3, 2].Value = 71;
    // Create Table
    ITable table = Worksheet.Tables.Add("A1:C4", YesNoGuess.Yes);

    // Add SlicerCache using the Column index
    ISlicerCache slicerCache = Workbook.SlicerCaches.Add(table, 0, "slicerCache");

    // Add Slicer to SlicerCache
    ISlicer slicer = slicerCache.Slicers.Add(Worksheet, "slicer", "Name", 200, 20, 200,
    200);

    // Add another SlicerCache using the Column Name
    ISlicerCache slicerCache2 = Workbook.SlicerCaches.Add(table, "City", "slicerCache2");

    // Add another Slicer to another SlicerCache
    ISlicer slicer2 = slicerCache2.Slicers.Add(Worksheet, "slicer2", "City", 410, 20, 200,
    200);
}
```

Select Slicer Items

Users can select and deselect slicer items by setting the **Selected ('Selected Property' in the on-line documentation)** property of the **ISlicerItem ('ISlicerItem Interface' in the on-line documentation)** interface to "true" or "false" respectively.

The following example code shows how to select slicer items in the spreadsheet.

C#

```
private static void Filter(IWorkbook Workbook)
{
    Workbook.SlicerCaches[0].SlicerItems["Bob"].Selected = false;
    Workbook.SlicerCaches[0].SlicerItems["Betty"].Selected = true;
    Workbook.SlicerCaches[0].SlicerItems["Alice"].Selected = false;
}
```

Alternatively, you can use the keyboard shortcut "Ctrl + Click" or "Alt+S" to enable the multi selection mode. Then, you can click on the specific sites to filter out the data of selected items. To select multiple slicer items, you simply need to click the first item on the slicer, drag the mouse pointer towards the last item and then all the items between the first and last item will be selected.

Clear Slicer Filters

Users can also remove the slicer filters when they want to show all the data by using the **ClearAllFilters() ('ClearAllFilters Method' in the on-line documentation)** method, the **ClearManualFilter() ('ClearManualFilter Method' in the on-line documentation)** method and the **ClearDateFilter() ('ClearDateFilter Method' in the on-line documentation)** method of the **ISlicerCache ('ISlicerCache Interface' in the on-line documentation)** interface. Alternatively, you can also click the "Clear Filter" button at the top right of the slicer or use the keyboard shortcut "Alt + C" to clear slicer filters from the spreadsheet.

The following example code shows how to clear slicer filters from the spreadsheet.

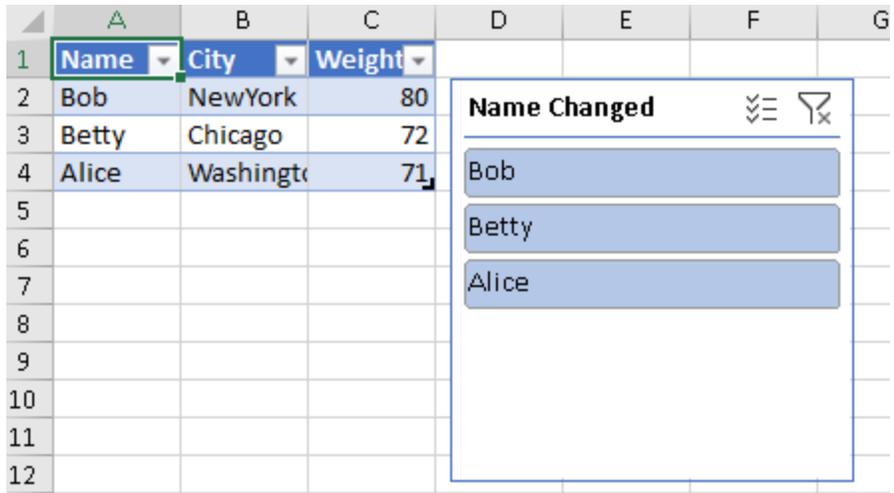
C#

```
private static void ClearFilter(IWorkbook Workbook)
{
    // Clear all filters from the Slicer
    Workbook.SlicerCaches[0].ClearAllFilters();
}
```

Modify Slicer Settings

Users can also modify slicer settings as per their specific preferences by using the **Caption ('Caption Property' in the on-line documentation)** property of the **ISlicer ('ISlicer Interface' in the on-line documentation)** interface and the **SortItems ('SortItems Property' in the on-line documentation)** property of the **ISlicerCache ('ISlicerCache Interface' in the on-line documentation)** interface.

The following image shows the slicer with changed slicer caption and modified sorting order (names are showing up in the descending order).



The following example code shows how to modify slicer settings in the spreadsheet.

C#

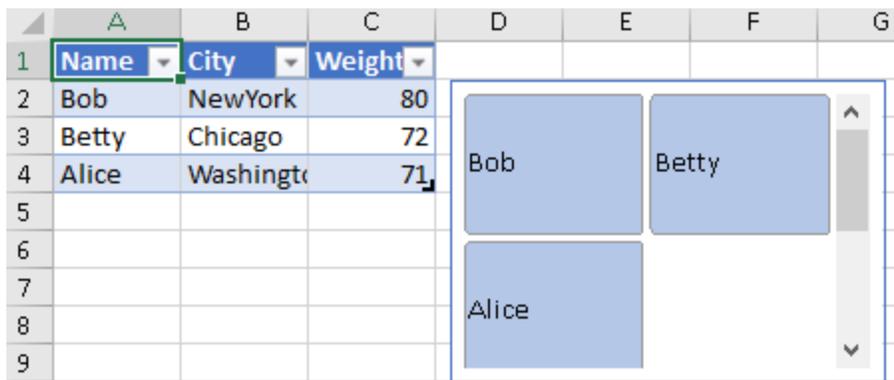
```
private static void ModifySlicerSetting(IWorkbook WorkBook, IWorksheet WorkSheet)
{
    // Change the Caption Name
    WorkSheet.Slicers["slicer"].Caption = "Name Changed";

    // Change the Sorting Order
    WorkBook.SlicerCaches["slicerCache"].SortItems = SlicerSort.Descending;
}
```

Change Slicer Formatting

Users can also modify the formatting of the slicer by using the **NumberOfColumns** ('**NumberOfColumns Property**' in the on-line documentation) property (to set the number of columns), the **Width** ('**Width Property**' in the on-line documentation) property (to set the width of the slicer), the **Height** ('**Height Property**' in the on-line documentation) property (to set the height of the slicer), the **ColumnWidth** ('**ColumnWidth Property**' in the on-line documentation) property (to set the width of the columns), the **RowHeight** ('**RowHeight Property**' in the on-line documentation) property (to set the height of the rows) and the **DisplayHeader** ('**DisplayHeader Property**' in the on-line documentation) property (to configure the visibility of the header) of the **ISlicer** ('**ISlicer Interface**' in the on-line documentation) interface.

The following image shows a slicer with custom formatting.



The following example code shows how to change slicer formatting in the spreadsheet.

C#

```
private static void ChangeSlicerFormatting(IWorksheet WorkSheet)
{
    // Change Slicer Formatting
    WorkSheet.Slicers["slicer"].NumberOfColumns = 2;
    WorkSheet.Slicers["slicer"].Width = 250;
    WorkSheet.Slicers["slicer"].Height = 150;
    WorkSheet.Slicers["slicer"].ColumnWidth = 100;
    WorkSheet.Slicers["slicer"].RowHeight = 70;
    WorkSheet.Slicers["slicer"].DisplayHeader = false;
}
```

Customize Slicer Style

Users can customize the slicer style as per their specific preferences by defining a custom style, setting the **ShowAsAvailableSlicerStyle** ('**ShowAsAvailableSlicerStyle Property**' in the on-line documentation) property of the **ITableStyle** ('**ITableStyle Interface**' in the on-line documentation) interface to true and applying the custom style to the slicer.

The following image shows a slicer with custom font style, font size and font color configured in the spreadsheet.

	A	B	C	D	E	F	G
1	Name	City	Weight				
2	Bob	NewYork	80				
3	Betty	Chicago	72				
4	Alice	Washington	71	Bob	Betty		
5							
6							
7							
8				Alice			
9							

The following example code shows how to customize slicer styles in the spreadsheet.

C#

```
private static void CustomizeSlicerStyle(IWorkbook WorkBook, IWorksheet WorkSheet)
{
    // Set Custom Style to Slicer
    ITableStyle slicerStyle = WorkBook.TableStyles.Add("CustomStyle1");

    // Enable ShowAsAvailableSlicerStyle to true to set custom style to Slicer
    slicerStyle.ShowAsAvailableSlicerStyle = true;
    slicerStyle.TableStyleElements[TableStyleElementType.SelectedItemWithData].Font.Name =
    "Arial";
    slicerStyle.TableStyleElements[TableStyleElementType.SelectedItemWithData].Font.Bold =
    true;
    slicerStyle.TableStyleElements[TableStyleElementType.SelectedItemWithData].Font.Italic
    = true;
    slicerStyle.TableStyleElements[TableStyleElementType.SelectedItemWithData].Font.Size =
```

```
20;
slicerStyle.TableStyleElements[TableStyleElementType.SelectedItemWithData].Font.Color =
GrapeCity.Spreadsheet.Color.FromThemeColor(GrapeCity.Core.ThemeColors.Accent3);
WorkSheet.Slicers["slicer"].Style = slicerStyle;
}
```

Delete Slicer

Users can delete the slicer from the spreadsheet by using the **Delete()** ('Delete Method' in the on-line documentation) method of the **ISlicer** ('ISlicer Interface' in the on-line documentation) interface.

If you want to delete the slicer cache and all its slicers, you can use the **Delete()** ('Delete Method' in the on-line documentation) method of the **ISlicerCache** ('ISlicerCache Interface' in the on-line documentation) interface.

The following example code shows how to delete a slicer from the spreadsheet.

C#

```
private static void DeleteSlicer(IWorkbook Workbook)
{
    // Delete Slicer
    Workbook.SlicerCaches["slicerCache"].Slicers["slicer"].Delete();
}
```

Working with BuiltIn Dialogs

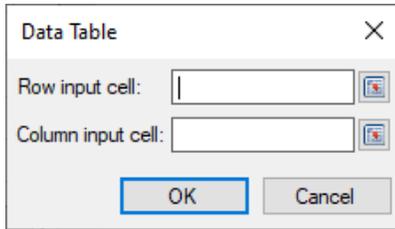
The Spread Designer provides a number of built-in dialogs that enable users to add data tables, format cells, insert functions, create forecast sheets, remove duplicates, add fill effects, etc.

You can invoke the built-in dialogs at runtime using the **BuiltInDialogs** class.

- **Data Table Dialog**
- **Fill Effects Dialog**
- **Create Forecast Worksheet Dialog**
- **Format Cells Dialog**
- **Goal Seek Dialog**
- **Colors Dialog**
- **Name Manager Dialog**
- **New Name Dialog**
- **Remove Duplicates Dialog**
- **Convert Text To Columns Wizard Dialog**
- **Insert Function Dialog**

Data Table Dialog

The **Data Table** dialog, you can prompt the user to enter the desired row or column input cells at run-time to create a data table in the Spread worksheet. The **BuiltInDialogs** class provides the **DataTable** method to invoke the DataTable dialog at runtime.



The following example code shows how to use the `DataTable` method to call the Data Table dialog at runtime:

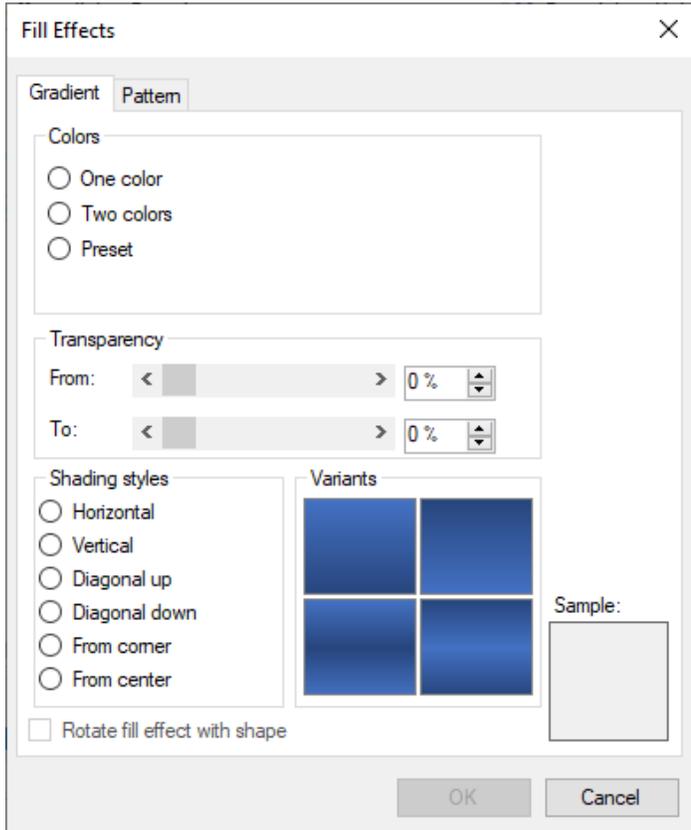
C#

```
FarPoint.Win.Spread.Dialogs.BuiltInDialogs.DataTable(fpSpread1).Show(fpSpread1);
```

For more information on data tables, see **DataTable** topic.

Fill Effects Dialog

You can add fill effects like color gradients and pattern to a shape using the **Fill Effects** dialog, which provides options like colors, variants, shading styles, patterns, and a sample preview to view the gradient effect. You can invoke the dialog using the **FillEffects** method of the **BuiltInDialogs** class.



The following example code shows how to call the Fill Effects dialog box at runtime:

C#

```
FpSpread1.Features.EnhancedShapeEngine = true;
IShape shape1 = FpSpread1.AsWorkbook.ActiveSheet.Shapes.AddShape(AutoShapeType.Rectangle, 50, 100,
100, 200);
FarPoint.Win.Spread.Dialogs.BuiltInDialogs.FillEffects(FpSpread1.AsWorkbook(),
((Shape)shape1).Brush, Color.FromArgb(255, 255, 255), Color.FromArgb(0, 0, 0)).ShowDialog();
```

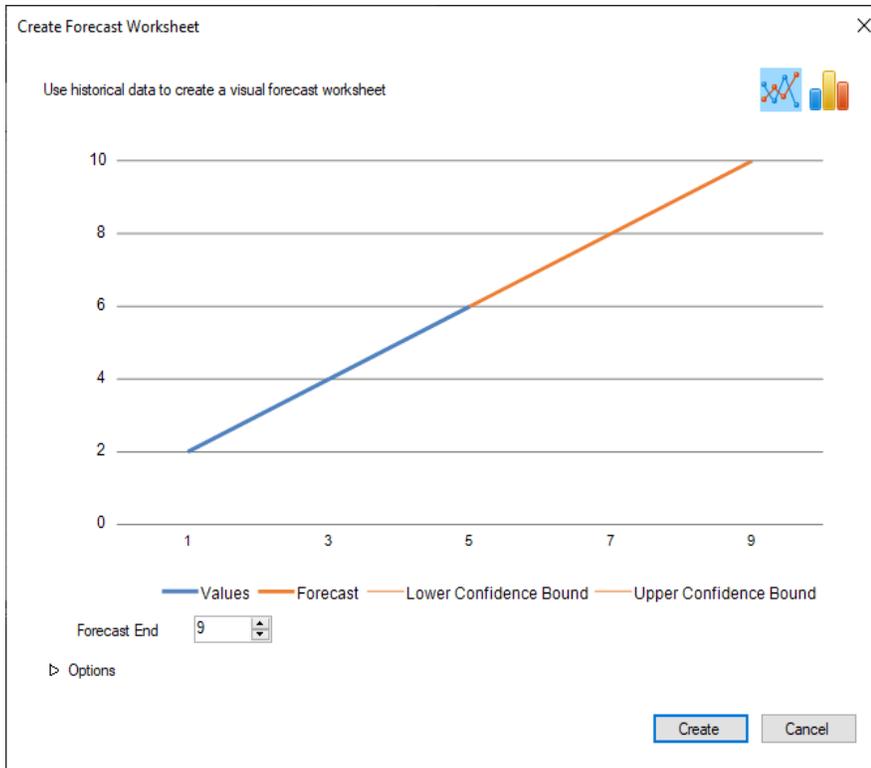
VB

```
FpSpread1.Features.EnhancedShapeEngine = True
Dim shapel As IShape = FpSpread1.AsWorkbook.ActiveSheet.Shapes.AddShape(AutoShapeType.Rectangle,
50, 100, 100, 200)
FarPoint.Win.Spread.Dialogs.BuiltInDialogs.FillEffects(FpSpread1.AsWorkbook(), CType(shapel,
Shape).Brush, Color.FromArgb(255, 255, 255), Color.FromArgb(0, 0, 0)).ShowDialog()
```

For more information on fill effects in shapes, see **Gradient Fill Effects** topic.

Create Forecast Worksheet Dialog

The **BuiltInDialogs** class provides the **ForecastSheet** method to create a Forecast Sheet dialog that allows to create a new worksheet that predicts data trends.



The following example code shows how to call the Forecast Sheet dialog box at runtime:

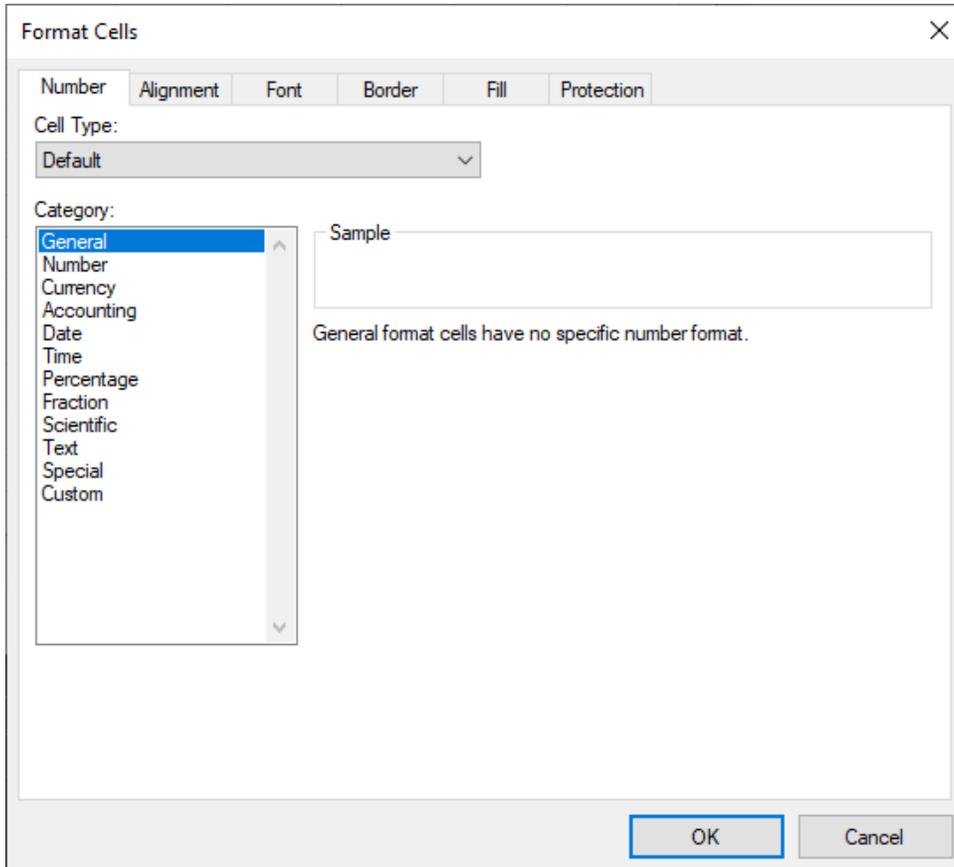
C#

```
fpSpread1.ActiveSheet.SetClip(0, 0, 3, 1, "1\n3\n5");
fpSpread1.ActiveSheet.SetClip(0, 1, 3, 1, "2\n4\n6");
TestActiveSheet.Cells["A1:B3"].Select();
FarPoint.Win.Spread.Dialogs.BuiltInDialogs.ForecastSheet(fpSpread1).Show(fpSpread1);
```

For more information about ForeCast feature, see **Forecast Sheet ('Forecast Sheet Dialog' in the on-line documentation)** topic.

Format Cells Dialog

The **Format Cells** dialog allows the user to customize the style settings of the specified range. The **BuiltInDialogs** class provides the **FormatCells** method to invoke this dialog at runtime. The **FormatCells** method has two overloads, **FormatCells(GrapeCity.Spreadsheet.IRange cells, [FarPoint.Win.Spread.FormatCells.FormatCellsTab activeTab = o])** and **FormatCells(FarPoint.Win.Spread.CellType.IRichTextEditor richTextEditor)**.



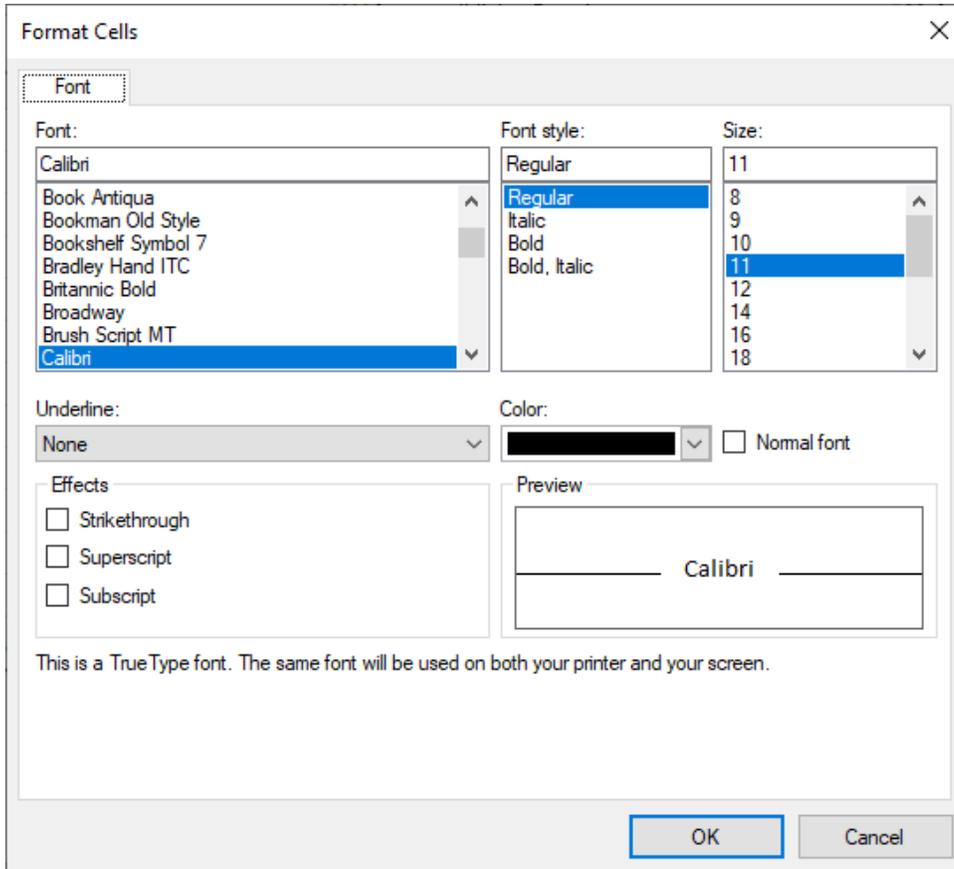
The following example code shows how to call the **Format Cells** dialog box to manipulate the style settings in the cell range using the **FormatCells(GrapeCity.Spreadsheet.IRange cells, [FarPoint.Win.Spread.FormatCells.FormatCellsTab activeTab = o])** method.

C#

```
FarPoint.Win.Spread.Dialogs.BuiltInDialogs.FormatCells(TestActiveSheet.Cells["A1:B1"]).ShowDialog();
```

For more information on formatting cells, refer to **Cell Format ('Formatter and Editor' in the on-line documentation)** topic.

The **FormatCells(FarPoint.Win.Spread.CellType.IRichTextEditor richTextEditor)** method allows the user to format and edit the rich text in cells at run-time.



The following example code shows how to invoke the **Format Cells** dialog box for rich text editing:

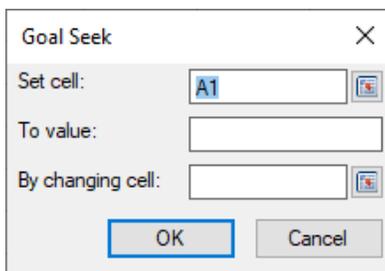
C#

```
fpSpread1.Features.RichTextEdit = RichTextEditMode.On;
TestActiveSheet.Cells["A1"].Value = "abc";
private void fpSpread1_EditModeOn(object sender, EventArgs e)
{
    if (fpSpread1.EditingControl is IRichTextEditor richTextEditor)
    {
        FarPoint.Win.Spread.Dialogs.BuiltInDialogs.FormatCells(richTextEditor);
    }
}
```

To learn about rich text editing in Spread, see **Rich Text Editor ('Rich Text Editing' in the on-line documentation)** topic.

Goal Seek Dialog

The **Goal Seek** dialog allows the user to calculate the values necessary to achieve a specific goal in the Spread worksheet. For this purpose, the **BuiltInDialogs** class provides the **GoalSeek** method.



The following example code shows how to call the **Goal Seek** dialog at runtime:

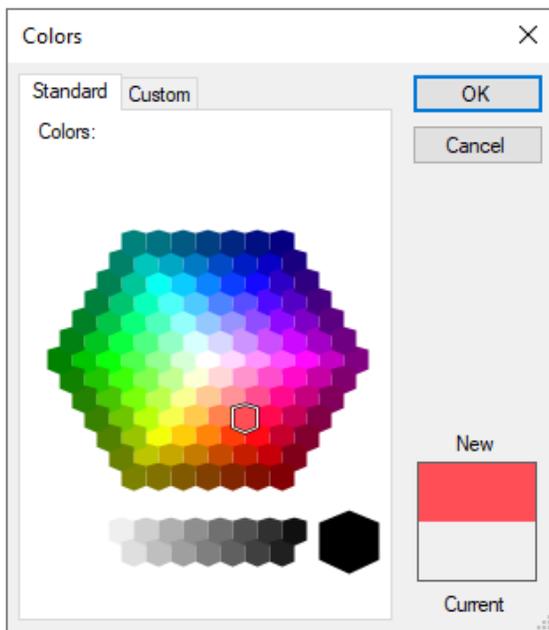
C#

```
TestActiveSheet.Cells["B1"].Formula = "A1";  
TestActiveSheet.Cells["B2"].Formula = "A2";  
FarPoint.Win.Spread.Dialogs.BuiltInDialogs.GoalSeek(fpSpread1).Show(fpSpread1);
```

To learn in detail about the Goal Seek feature, see **Goal Seek topic ('Goal Seek Dialog' in the on-line documentation)**.

Colors Dialog

The **Colors** dialog allows users to choose customized colors in Spread. It lets you choose between standard and custom colors at runtime. The **BuiltInDialogs** class provides the **MoreColors** method for this purpose.



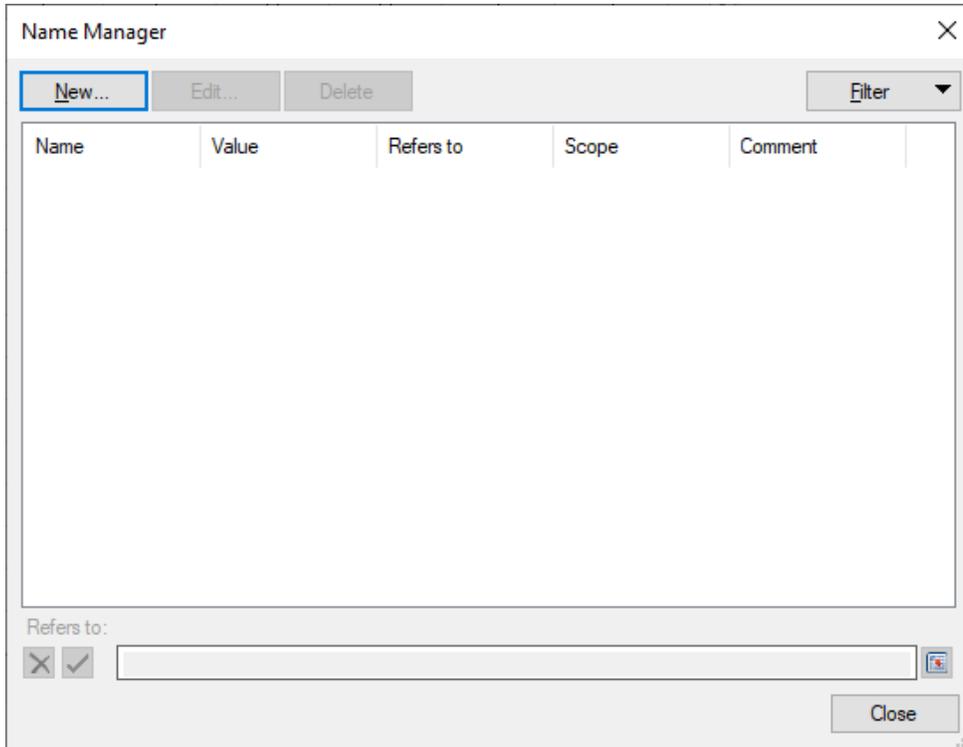
The following example code shows how to call the **Colors** dialog box using **MoreColors** method at runtime.

C#

```
IWorkbook activeWorkbook = fpSpread1.AsWorkbook();  
IRange range = activeWorkbook.ActiveSheet.ActiveCell;  
FarPoint.Win.Spread.Dialogs.MoreColors colorDlg =  
FarPoint.Win.Spread.Dialogs.BuiltInDialogs.MoreColors(range.Font.Color, activeWorkbook);  
if (colorDlg.ShowDialog() == DialogResult.OK)  
{  
    range.Interior.Color = colorDlg.Color;  
}  
else  
{  
    colorDlg.Dispose();  
    return;  
}
```

Name Manager Dialog

The **Name Manager** dialog lets you create, edit, or delete custom names in the Spread Designer. The **BuiltInDialogs** class provides the **NameManager** method to invoke the Name Manager dialog at runtime.



The following example code shows how to call the **Name Manager** dialog at runtime:

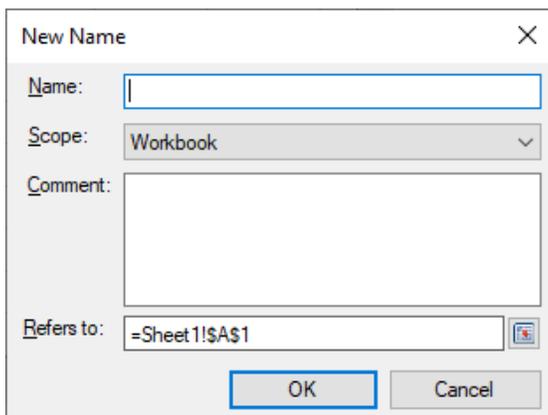
C#

```
FarPoint.Win.Spread.Dialogs.BuiltInDialogs.NameManager(fpSpread1).Show(fpSpread1);
```

For more information, see [Name Manager Dialog](#) topic.

New Name Dialog

The **New Name** dialog allows users to create new defined names in Spread. You can create names for cell ranges, choose whether a name should be accessible within an workbook or an individual worksheet only, and also add comments, if necessary. The **BuiltInDialogs** class provides the **NewName** method to invoke this dialog at runtime.



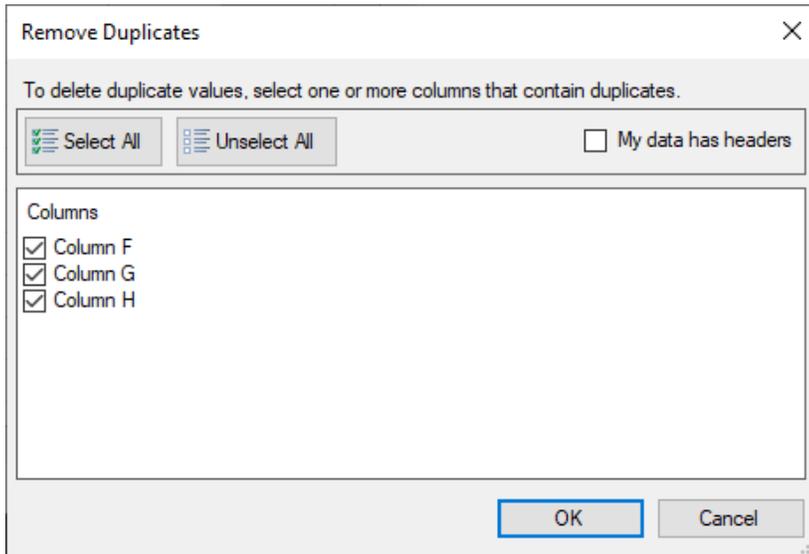
The following example code shows how to call the **New Name** dialog box:

C#

```
FarPoint.Win.Spread.Dialogs.BuiltInDialogs.NewName(fpSpread1).Show(fpSpread1);
```

Remove Duplicates Dialog

The **Remove Duplicates** dialog allows the user to remove duplicate values from a range of values. Spread allows you to invoke this dialog at runtime. For this purpose, the **BuiltInDialog** class provides the **RemoveDuplicates** method.



The following example code shows how to call the **Remove Duplicates** dialog at runtime:

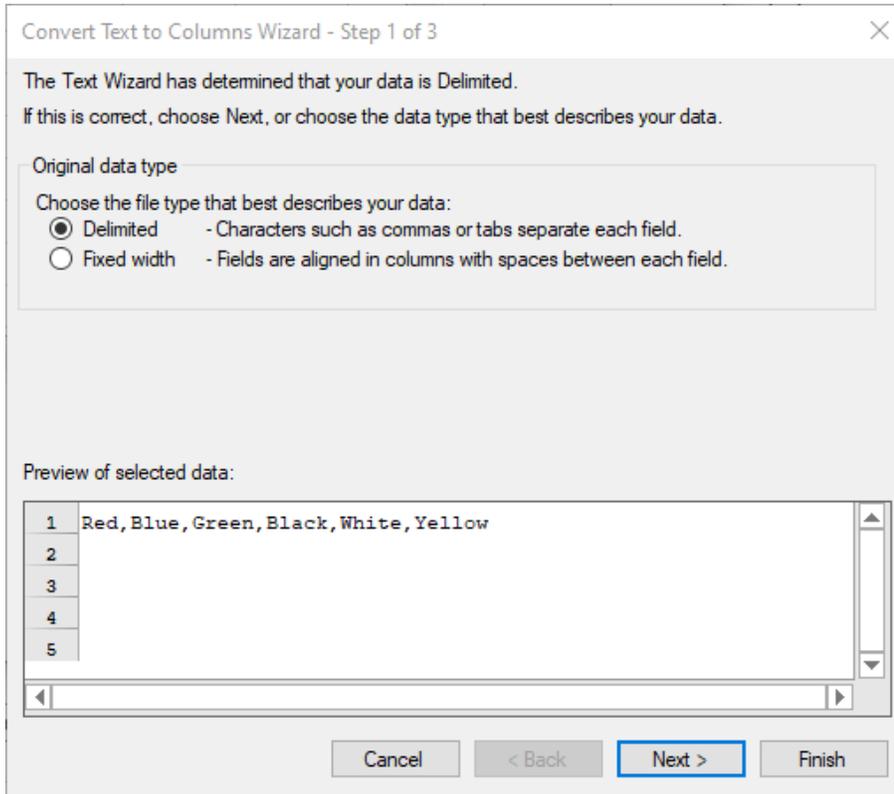
C#

```
fpSpread1.ActiveSheet.SetClip(0, 1, 6, 1, "2\n1\n3\n2\n5\n3");  
TestActiveSheet.Cells["B1:B8"].Select();  
FarPoint.Win.Spread.Dialogs.BuiltInDialogs.RemoveDuplicates(fpSpread1).ShowDialog(fpSpread1);
```

For more information about removing duplicates in Spread, see **Removes Duplicates** topic.

Convert Text To Columns Wizard Dialog

The **Convert Text To Columns Wizard** dialog lets you parse the text from one cell or column into multiple columns using a delimiter. The **BuiltInDialogs** class provides the **TextToColumns** method to invoke this dialog at runtime.



The following example code shows how to call the **Convert Text To Columns Wizard** dialog at runtime.

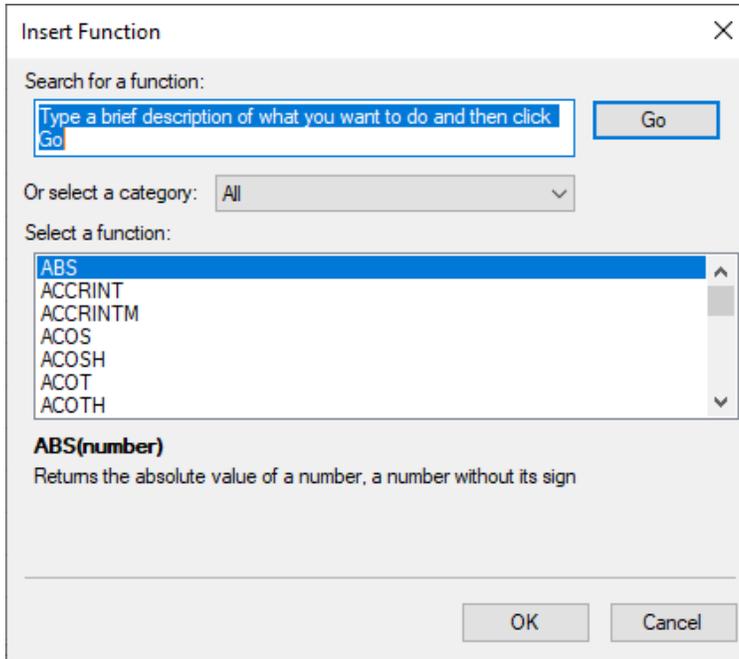
C#

```
TestActiveSheet.Cells["A1"].Value = "Red,Blue,Green,Black,White,Yellow";  
FarPoint.Win.Spread.Dialogs.BuiltInDialogs.TextToColumns(fpSpread1).Show(fpSpread1);
```

For more information about the feature, see **Text to Columns** topic.

Insert Function Dialog

Users can call the **Insert Function** dialog at runtime using the **ToggleInsertFunction** method of the **BuiltInDialogs** class. The **Insert Function** dialog lets you search and select functions, and insert them in cells.



The following example code shows how to invoke the **Insert Function** dialog:

C#

```
FarPoint.Win.Spread.Dialogs.BuiltInDialogs.ToggleInsertFunction(fpSpread1);
```

To learn more about Insert Functions, see **Insert Function ('Insert Function Dialog' in the on-line documentation)** Dialog.

Ribbon Control

Spread ribbonBar control provides an Excel-like ribbon UI for Spread Win, including contextual behaviors, command execution, and full customization options. The ribbonBar control serves as a replacement for the traditional menus and toolbars model and organizes related commands into tabs and groups so that the commands are easier to find.

The ribbonBar control consists of default tabs, groups, and group items to be used with the Spread Win control.



Runtime Modes

There are properties that change supported runtime behaviors of Spread. The ribbonBar displays different items based on the current modes of Spread.

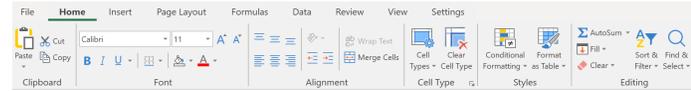
Legacy Behaviors

If the composite style is used (**LegacyBehaviors.Style** is active), Spread uses a legacy style system. Therefore, certain Excel-compatible items, such as Cell Styles and Excel-like number formats, will be hidden as shown below.

Flat Style



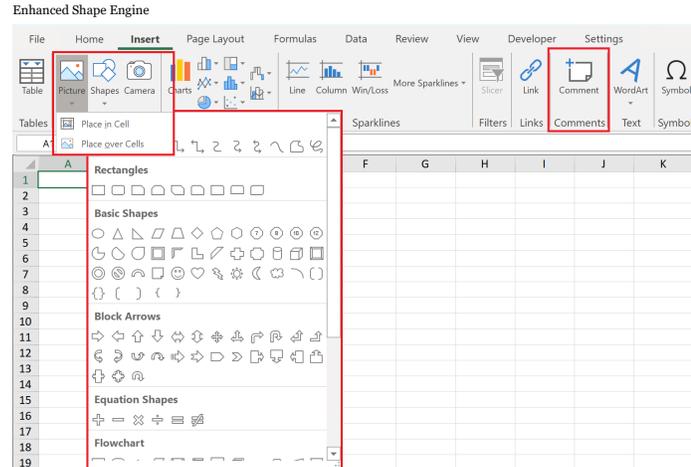
Composite Style



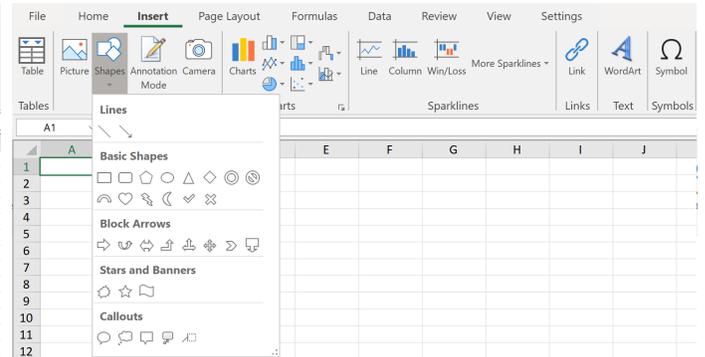
Enhanced Shape Engine

If the **EnhancedShapeEngine** property is set to true, Spread lets you interact seamlessly with all Excel-compatible shapes, notes, and comments. However, the options will be limited for **LegacyShapeEngine**.

Insert tab options



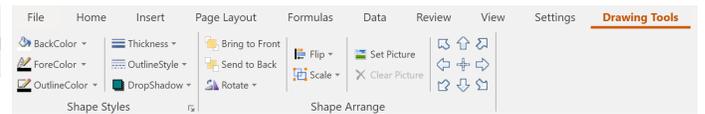
Legacy Shape Engine



Contextual tab of shape



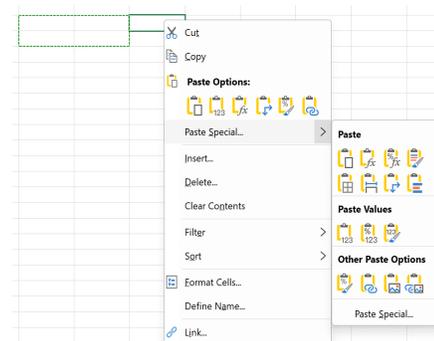
Legacy Shape Engine



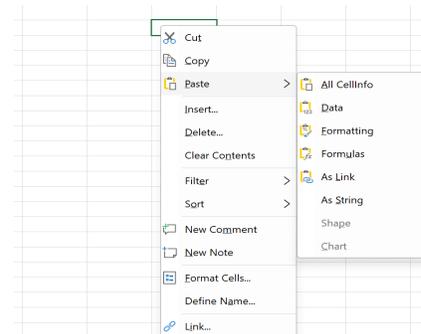
Rich Clipboard

If the **RichClipboard** property is set to true, different context menus are displayed by default when Spread does not have a specified context menu.

Rich Clipboard



Legacy Clipboard



Generate RibbonBar Items Manually

Spread allows you to manually regenerate the ribbonBar to reflect default changes in behaviors at runtime using the following code.

```
C#
ribbonBar1.GenerateDefaultItems();
```

Attach RibbonBar Control

Additionally, you can use the **Attach** function to use the ribbonBar control with the specified FpSpread control. Once the ribbonBar is attached, all default tabs will be automatically generated.

```
C#
// Attach ribbonBar control to FpSpread control
ribbonBar1.Attach(fpSpread1);
```

Override Built-In RibbonBar Commands

To override a built-in command, use **CommandExecuting** and **CommandExecuted** events. You can also skip the built-in command by changing the **CommandExecuting.Handled** value to true.

The following is an example code for the override command. The "Cut" command here is overridden when used via the ribbonBar control, where it is not cutting the selected cell text, instead of filling it with the bgcolor. Please note that this overriding behavior will not work when performed using shortcut keys (Ctrl+C).

```
C#
// Override command
ribbonBar1.Attach(fpSpread1);
ribbonBar1.CommandExecuting += RibbonBar1_CommandExecuting;
ribbonBar1.CommandExecuted += RibbonBar1_CommandExecuted;
fpSpread1.AsWorkbook().ActiveSheet.ActiveCell.Value = 123;
private void RibbonBar1_CommandExecuted(object sender, ExecuteCommandEventArgs e)
{
    Console.WriteLine("executing: " + e.CommandName);
}
private void RibbonBar1_CommandExecuting(object sender, ExecuteCommandEventArgs e)
{
    if (e.CommandName == "Cut")
    {
        e.Handled = true;
        GrapeCity.Spreadsheet.Color color = GrapeCity.Spreadsheet.Color.FromKnownColor(GrapeCity.Core.KnownColor.BlueViolet);
        IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;
        TestActiveSheet.ActiveCell.Interior.Color = color;
    }
}
```

Execute RibbonBar Command

To manually execute the ribbonBar commands at runtime, use the ExecuteCommand function. The following example code shows the use of the ExecuteCommand function to the fill color to change the appearance accordingly.

```
C#
// Execute command at runtime
ribbonBar1.Attach(fpSpread1);
GrapeCity.Spreadsheet.Color color = GrapeCity.Spreadsheet.Color.FromKnownColor(GrapeCity.Core.KnownColor.BlueViolet);
ribbonBar1.ExecuteCommand(GrapeCity.Spreadsheet.WinForms.Ribbon.BuiltInCommands.FillColor, color);
```

Add Custom Items to RibbonBar

A ribbonBar comprises of different tab items that perform a specified command or action. There are various types of group items available in a ribbonBar's tab, such as toolbars, menus, tab items, icons, and so on. You can also add, remove, or modify the ribbonBar items as per your requirements.

The following example code shows how to add a new tab along with a new tab group and group item.

```
C#
// Add custom items on ribbonBar
ribbonBar1.Attach(fpSpread1);
ribbonBar1.Tabs.Add(new GrapeCity.Spreadsheet.WinForms.Ribbon.RibbonTab());
ribbonBar1.Tabs[0].Text = "New Tab";
ribbonBar1.Tabs[0].Groups.Add(new RibbonGroup());
ribbonBar1.Tabs[0].Groups[0].Text = "New Group";
ribbonBar1.Tabs[0].Groups[0].Items.Add("New Item");
ribbonBar1.Tabs[0].Groups[0].Items[0].Name = "test";
ribbonBar1.Tabs[0].Groups[0].Items["test"].CommandName = "Orientation";
ribbonBar1.Tabs[0].Groups[0].Items["test"].CommandParameter = 30;
ribbonBar1.Tabs[0].Groups[0].Items[0].Visible = false;
(RibbonButton)ribbonBar1.Tabs[1].Groups[0].Items[0].Text = "New Name";
```

Limitations

- Spread does not support the Quick Access Toolbox, which appears in the title bar.
- The Spread ribbonBar control only works with built-in models.
- Invalid selection prevents ribbonBar from functioning.
- The font for the ribbonBar control's item text, group text, and tab text cannot be modified.

Sheets

You can have multiple sheets within a workbook. Each sheet is a separate spreadsheet and can have its own appearance and settings for user interaction. Each sheet has a unique name and sheet name tab for easy navigation between sheets.

These tasks relate to working with and setting the appearance of the sheets inside the Spread component:

- **Working with the Active Sheet**
- **Working with Multiple Sheets**
- **Customizing the Sheet Name Tabs**
- **Navigating Sheet Tabs**
- **Adding a Sheet**
- **Adding ChartSheet**
- **Removing a Sheet**
- **Showing or Hiding a Sheet**
- **Moving a Sheet**
- **Selecting Multiple Sheets**
- **Copying and Inserting a Sheet**
- **Protecting a Worksheet**
- **Form Controls**
- **Adding a Title and Subtitle to a Sheet**
- **Placing Child Controls on a Sheet**
- **Displaying a Footer for Columns or Groups**
- **Adding a Tag to a Sheet**
- **Working with 1-Based Indexing**
- **Customizing Clipboard Operation Options**

For more details on the objects involved, refer to the **SheetView ('SheetView Class' in the on-line documentation)** class and **SheetViewCollection ('SheetViewCollection Class' in the on-line documentation)** class.

Working with the Active Sheet

The active sheet is the sheet that currently receives any user interaction and displays on top of other sheets. You can specify the active sheet programmatically by using the **ActiveSheet ('ActiveSheet Property' in the on-line documentation)** property of the **FpSpread ('FpSpread Class' in the on-line documentation)** object. You can also specify the index of the active sheet by using the **ActiveSheetIndex ('ActiveSheetIndex Property' in the on-line documentation)** property.

Usually, the active sheet is displayed on top of other sheets.

Method to use

You can use `ActiveSheet` as a shortcut object for the active sheet when specifying properties of the sheet.

Example

The following example set properties of the active sheet and assign the active sheet to sheet number 2.

```
C#  
// Create three sheets in the component.  
fpSpread1.Sheets.Count = 3;
```

```
// Set third sheet (in zero-based index) be set to active sheet.
fpSpread1.ActiveSheetIndex = 2;
// Set some properties of the active sheet.
fpSpread1.ActiveSheet.ColumnCount = 8;
fpSpread1.InterfaceRenderer = NULL;
fpSpread1.ActiveSheet.GrayAreaBackColor = Color.Purple;
```

VB

```
' Create three sheets in the component.
fpSpread1.Sheets.Count = 3
' Set third sheet (in zero-based index) be set to active sheet.
fpSpread1.ActiveSheetIndex = 2
' Set some properties of the active sheet.
fpSpread1.ActiveSheet.ColumnCount = 8
fpSpread1.InterfaceRenderer = Nothing
fpSpread1.ActiveSheet.GrayAreaBackColor = Color.Purple
```

Working with Multiple Sheets

The component allows multiple sheets. You can specify the number of sheets with the **Count ('Count Property' in the on-line documentation)** property of the **SheetViewCollection ('SheetViewCollection Class' in the on-line documentation)** class. For example, you can set the Spread to have 5 sheets (and each sheet has a sheet name tab) using this code.

C#

```
fpSpread1.Sheets.Count = 5;
```

Visual Basic

```
FpSpread1.Sheets.Count = 5
```

You can set the properties of each sheet in code by using **ActiveSheet ('ActiveSheet Property' in the on-line documentation)** property of the **SheetView ('SheetView Class' in the on-line documentation)** or **FpSpread ('FpSpread Class' in the on-line documentation)** class.

You can name the sheets or use the default sheet names. The default sheet name is "Sheet1" and as other sheets are added, the sheet are named incrementally "Sheet2", "Sheet3", and so on. Use the **SheetName ('SheetName Property' in the on-line documentation)** property of the **SheetView ('SheetView Class' in the on-line documentation)** class to name the sheet programmatically. The addition and removal of sheets are described below.

- **Add a sheet**
- **Remove a sheet**

The **Sheets ('Sheets Property' in the on-line documentation)** property of the **FpSpread ('FpSpread Class' in the on-line documentation)** class returns a **SheetViewCollection ('SheetViewCollection Class' in the on-line documentation)** object that holds all the sheets of the control. In the **Sheets** property, specify an integer starting with "0" as an index to refer to a specific sheet.

The following sample code copies the sheet with the index "1" of the fpSpread1 control (second sheet) to the sheet with the index "0" of the fpSpread2 control (the first sheet). Since both refer to the same SheetView object in this sample code, changes made to one sheet gets reflected to the other sheet as well.

C#

```
fpSpread2.Sheets[0] = fpSpread1.Sheets[1];
```

Visual Basic

```
FpSpread2.Sheets(0) = FpSpread1.Sheets(1)
```

To return a sheet by name, you can use the **Find ('Find Method' in the on-line documentation)** method of the **SheetViewCollection ('SheetViewCollection Class' in the on-line documentation)** class as shown in the following code.

C#

```
fpSpread1.Sheets.Find("Sheet1").Cells[0, 0].Value = "test";
```

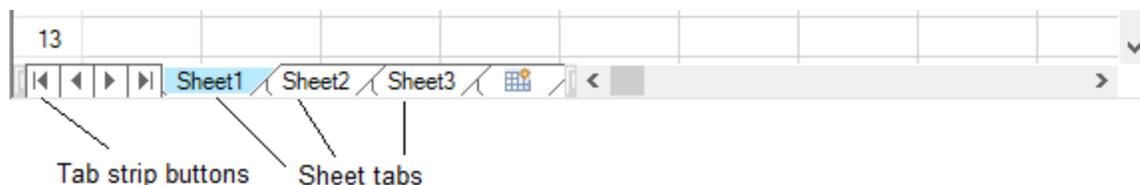
Visual Basic

```
FpSpread1.Sheets.Find("Sheet1").Cells(0, 0).Value = "test"
```

For information about the display of the sheet names, kindly refer to "**Customizing the Sheet Name Tabs**".

Customizing the Sheet Name Tabs

If there is more than one sheet in the workbook, the tab strip displays the sheet names tabs in a tab strip with the tab for the active sheet highlighted. The sheet tabs provide a way for the user to navigate to different sheets.



You can customize how and if to display the sheet names in tabs of the Spread component. By default, the tab strip is not displayed because there is only one sheet in the component until more sheets are added.

The contents that can be customized are as follows.

Tab Strip Display

You can set the component to always or never display the tab strip, or to display only when there are at least two sheets. For more information and code examples, refer to these members:

- **TabStripPolicy ('TabStripPolicy Property' in the on-line documentation)** property of **FpSpread ('FpSpread Class' in the on-line documentation)** class
- **TabStrip ('TabStrip Property' in the on-line documentation)** property of **FpSpread ('FpSpread Class' in the on-line documentation)** class
- **ButtonPolicy ('ButtonPolicy Property' in the on-line documentation)** property of **TabStrip ('TabStrip Class' in the on-line documentation)** class
- **TabStripButtonPolicy ('TabStripButtonPolicy Enumeration' in the on-line documentation)** enumeration



- If you are planning to export the contents of the Spread to import into Excel, do not use characters in the sheet name that are invalid in Excel. Invalid Excel sheet name characters include: ? / \ * []
- For more information on how to add sheets to the workbook, kindly refer to "**Adding a Sheet**".

Tab Strip Appearance

You can customize the appearance of the entire tab strip as well as the individual sheet name tabs.

You can set properties for the tab strip such as the background color (set the **InterfaceRenderer** (**InterfaceRenderer Property** in the on-line documentation) property of **FpSpread** (**FpSpread Class** in the on-line documentation) class to **Nothing** or **null** and visual styles to **Off**) and text font for the sheet tabs. The default sheet names are **Sheet1**, **Sheet2**, and so on. You can specify other names for the sheets and these appear in the sheet tabs. You can also allow the user to edit the sheet names.

For more information, refer to these class.

- **SheetTab** (**SheetTab Class** in the on-line documentation) class
- **TabStrip** class (on-line documentation)

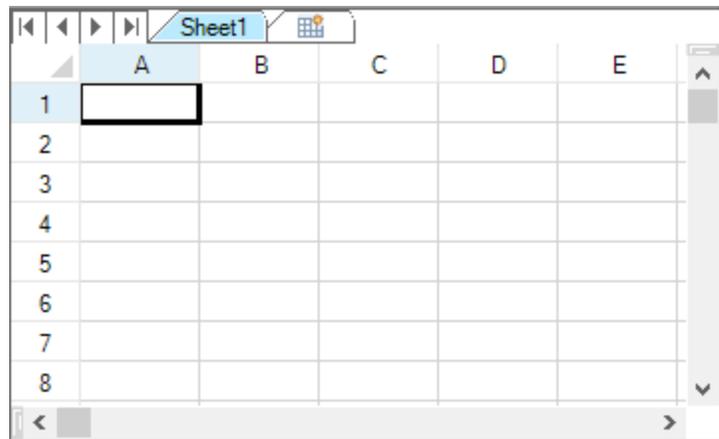
Tab Strip Placement

You can customize where the tab strip is displayed in the overall component.

Placement Value

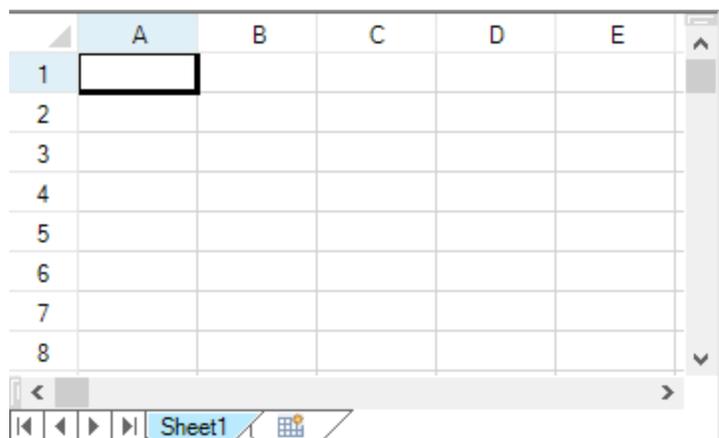
Top(at the top of the component above the headers)

Sample Showing Placement

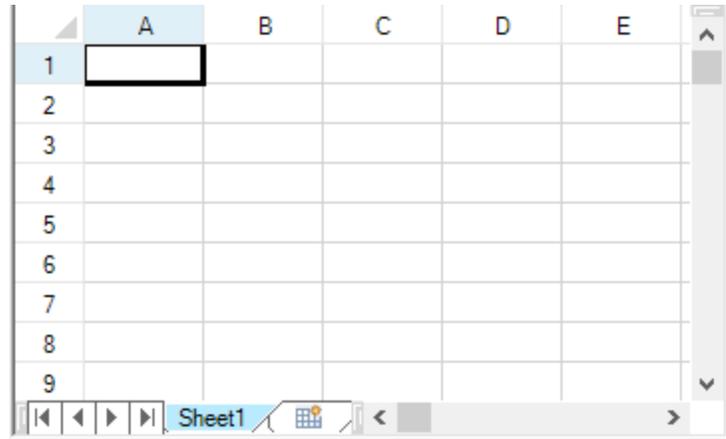


 If the grouping display is turned on, the tab strip appears below the group bar.

Bottom(under the scroll bar at the bottom of the component)



WithScrollBar(alongside the scroll bar at the bottom of the component)



For more information and code examples, refer to the **TabStripPlacement ('TabStripPlacement Property' in the on-line documentation)** property of the **FpSpread ('FpSpread Class' in the on-line documentation)** class.

Tab Strip Width

You can specify the width of the tab strip in relation to the overall scroll bar width, if the tab strip and scroll bar are displayed together in line.

You can set how wide the tab strip is, and therefore, how many sheet tabs are displayed. If the number of tabs exceeds the width of the tab strip, the component displays buttons. Click the buttons to display the next (or previous) sheet tabs. The width is set by setting the **FpSpread TabStripRatio ('TabStripRatio Property' in the on-line documentation)** property of the **FpSpread ('FpSpread Class' in the on-line documentation)** class, which sets the width of the tab strip as a percentage of the length of the entire component. By default, the ratio is set to 0.50 (area is divided 50% tab strip and 50% scroll bar).

For more information on setting the scroll bar properties, kindly refer to "**Customizing the Scroll Bars**."

Pointer Display over Tab Strip

You can specify that the pointer changes appearance when it is over the tab strip. Use the **TabStrip** value of the **CursorType ('CursorType Enumeration' in the on-line documentation)** enumeration to display a pointer in the sheet tabs.

First Tab in Tab Strip

You can set which sheet tab to display as the left-most tab with the **LeftTab ('LeftTab Property' in the on-line documentation)** property of the **FpSpread** class.

Tab Strip Events

You can work with the following events and event handlers.

- **OnSheetTabClick ('OnSheetTabClick Method' in the on-line documentation)** Method and **OnSheetTabDoubleClick ('OnSheetTabDoubleClick Method' in the on-line documentation)** Method of **FpSpread ('FpSpread Class' in the on-line documentation)** Class
- **SheetTabClick ('SheetTabClick Event' in the on-line documentation)** Event
- **SheetTabClickEventArgs ('SheetTabClickEventArgs Class' in the on-line documentation)** Class

- **SheetTabClickEventHandler** ('SheetTabClickEventHandler Delegate' in the on-line documentation) Delegate
- **SheetTabDoubleClick** ('SheetTabDoubleClick Event' in the on-line documentation) Event
- **SheetTabDoubleClickEventArgs** ('SheetTabDoubleClickEventArgs Class' in the on-line documentation) Class
- **SheetTabDoubleClickEventHandler** ('SheetTabDoubleClickEventHandler Delegate' in the on-line documentation) Delegate

The name of the event that occurs when a user clicks on the sheet name tab is the **SheetTabClick** ('SheetTabClick Event' in the on-line documentation) event. The tab that the user has clicked can be determined by getting the **SheetTabIndex** ('SheetTabIndex Property' in the on-line documentation) value.

The following is sample code to get the tab name (sheet name) from the index of the sheet.

C#

```
private void fpSpread1_SheetTabClick(object sender,
FarPoint.Win.Spread.SheetTabClickEventArgs e)
{
    string name = fpSpread1.Sheets[e.SheetTabIndex].SheetName;
}
```

Visual Basic

```
Private Sub FpSpread1_SheetTabClick(ByVal sender As Object, ByVal e As
FarPoint.Win.Spread.SheetTabClickEventArgs) Handles
FpSpread1.SheetTabClick
    Dim name as String = FpSpread1.Sheets(e.SheetTabIndex).SheetName
End Sub
```

You can customize all these features using code, and some can be set in the Spread Designer. For more information on how to work with the tab strip settings in Spread Designer, refer to the Spread Settings, General Tab in the Spread Designer Guide.

Example

This example sets the sheet tabs to always appear, sets the tab strip buttons to only appear as needed, sets the background color of the sheet tabs to Bisque, and sets the width of the tab strip to 60%.

C#

```
// Set the sheet tabs to always appear
fpSpread1.TabStripPolicy = FarPoint.Win.Spread.TabStripPolicy.Always;
// Set the width to 60%
fpSpread1.TabStripRatio = 0.60;
// Display the tab strip buttons as needed
fpSpread1.TabStrip.ButtonPolicy = FarPoint.Win.Spread.TabStripButtonPolicy.AsNeeded;
// Set the background color
fpSpread1.TabStrip.BackColor = Color.Bisque;
fpSpread1.InterfaceRenderer = null;
```

Visual Basic

```
' Set the sheet tabs to always appear
FpSpread1.TabStripPolicy = FarPoint.Win.Spread.TabStripPolicy.Always
' Set the width to 60%
FpSpread1.TabStripRatio = 0.60
' Display the tab strip buttons as needed
```

```
FpSpread1.TabStrip.ButtonPolicy = FarPoint.Win.Spread.TabStripButtonPolicy.AsNeeded
' Set the background color
FpSpread1.TabStrip.BackColor = Color.Bisque
FpSpread1.InterfaceRenderer = Nothing
```

Using the Spread Designer

1. From the **Settings** menu, choose **Tab Strip (Appearance)** section).
2. Under **TabStripPolicy**, select when you want the sheet tabs to be displayed or select **Never** to hide the sheet tabs.
No matter which item you select, inside Spread Designer the sheet tabs are always displayed to assist you in designing your component. When you exit Spread Designer and apply your changes, you can see the effect of the tab settings in your component, or you can see it by previewing the component inside Spread Designer. To preview inside Spread Designer, from the **File** menu choose **Preview**.
3. Set the width of the tab strip by setting the value in the **Sheet Tab Percentage** box.
4. Click **OK** to close the **Spread Options** dialog.
5. Select the Spread object.
6. In the property list, select the **TabStrip** property to see its properties.
7. Change the **ButtonPolicy** property if you want to change when the tab strip buttons are displayed.
8. Change the **BackColor** property if you want to change the background color for the tab strip.
9. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Navigating Sheet Tabs

If a workbook contains more than one sheet, the tab strip displays the sheet name tabs with the tab for the active sheet highlighted. In this scenario, by using the previous and next tab strip buttons present in the bottom left corner of the workbook, you can navigate to the previous or next sheet of the active sheet.

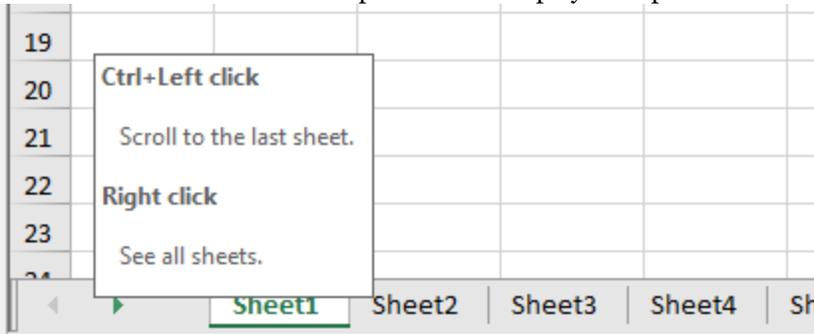
However, if you want to jump to the first or last sheets without any manual scrolling, follow the steps below.

1. Run the following example code to load a spread control with 6 sheets.

C#

```
fpSpread1.Width = 800;
fpSpread1.Sheets.Count = 6;
fpSpread1.TabStripRatio = 0.7;
```

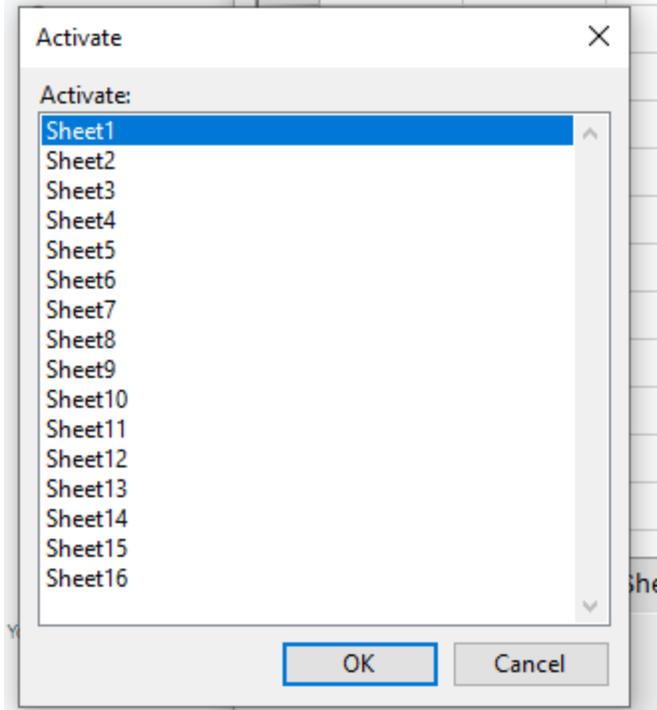
2. Hover the mouse on the tab strip buttons to display tooltips with a description of their functions.



 This tooltip functionality is only available in the Excel 2019 skin.

3. Hold **Ctrl** key and click on the **Previous** tab strip button to jump to the first sheet.

4. Hold **Ctrl** key and click on the **Next** tab strip button to jump to the last sheet.
5. Right-click on the tab strip buttons to immediately jump to the selected sheet. This will display the **Activate** dialog with the list of sheets available in the workbook. You can select any sheet of your choice from the list and click **OK** or double-click the sheet to open.



Spread also allows you to open the **Activate** dialog using **ActivateSheet** method of the **BuiltInDialogs** class as shown below.

C#

```
// Open Activate dialog by code
fpSpread1.Sheets.Count = 3;
FarPoint.Win.Spread.Dialogs.BuiltInDialogs.ActivateSheet(fpSpread1);
```

However, when not in use, you can hide the **Activate** dialog and tooltip and disable the tab strip buttons.

To hide Activate dialog

C#

```
fpSpread1.GetActionMap().Parent.Remove(SpreadActions.ShowActivateSheetDialog);
```

To hide the tooltip for tab strip buttons

C#

```
fpSpread1.SetToolTip(SpreadToolTip.TabStripNextSheet, "");
fpSpread1.SetToolTip(SpreadToolTip.TabStripPreviousSheet, "");
```

To disable tab strip buttons

C#

```
private void FpSpread1_MouseDown(object sender, MouseEventArgs e)
{
```

```
if ((Control.ModifierKeys & Keys.Control) != 0)
{
    FpSpread spread = (FpSpread)sender;
    HitTestInformation ht = spread.HitTest(e.X, e.Y);
    if (ht.Type == HitTestType.TabStrip)
    {
        switch (ht.TabStripInfo.Button)
        {
            case TabStripButton.Next:
            case TabStripButton.Previous:
                spread.Capture = false;
                break;
        }
    }
}
```

Adding a Sheet

You can add a sheet or add several sheets to the Spread component. By default, the component has one sheet, named Sheet 1 and referenced as sheet index 0. The sheet index is zero-based. In code, you can simply change the sheet count or you can explicitly add the sheet(s). If you are using custom sheet names be sure to specify the name of the sheet.

The user is allowed to add new sheets by default. They can do this by clicking on the new sheet icon on the tab strip next to the sheet name (provided the tab strip is visible). If the sheet count is greater than 1, the tab strip is displayed by default. You can change this by setting the **TabStripPolicy** (**'TabStripPolicy Property' in the on-line documentation**) property. You can prevent the user from adding new sheets by setting the **TabStripInsertTab** (**'TabStripInsertTab Property' in the on-line documentation**) property to false.

You can use the **Add** (**'Add Method' in the on-line documentation**) method to add a new sheet to the **SheetViewCollection** (**'SheetViewCollection Class' in the on-line documentation**) for the component. You can also use the **AddNewSheetView** (**'AddNewSheetView Method' in the on-line documentation**) method add a sheet.

For more information on how the sheet names appear in the sheet tabs, and how to customize the sheet tabs, refer to **Customizing the Sheet Name Tabs**.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. Click the **Add** button to add a sheet to the collection.
A new sheet named SheetView n (where n is an integer) is added to the component.
5. If you want to change the name of the new sheet, click the **SheetName** property in the property list, and then type the new name for the sheet.
6. Click **OK** to close the editor.

Method to use

Create an instance of the **SheetView** (**'SheetView Class' in the on-line documentation**) class that represents a sheet, and set each property of the class (such as the sheet name).

Add the created sheet with the **Add** (**'Add Method' in the on-line documentation**) method of **SheetViewCollection** (**'SheetViewCollection Class' in the on-line documentation**) class referenced by

Sheets ('Sheets Property' in the on-line documentation) property of **FpSpread ('FpSpread Class' in the on-line documentation)** class.

Example

This example code adds a new sheet to the component, then names the sheet "North" and sets it to have 10 columns and 100 rows.

C#

```
// Create a new sheet
FarPoint.Win.Spread.SheetView newsheet = new FarPoint.Win.Spread.SheetView();
newsheet.SheetName = "North";
newsheet.ColumnCount = 10;
newsheet.RowCount = 100;
// Add the new sheet to the component
fpSpread1.Sheets.Add(newsheet);
```

Visual Basic

```
' Create a new sheet
Dim newsheet As New FarPoint.Win.Spread.SheetView()
newsheet.SheetName = "North"
newsheet.ColumnCount = 10
newsheet.RowCount = 100
' Add the new sheet to the component
FpSpread1.Sheets.Add(newsheet)
```

Adding ChartSheet

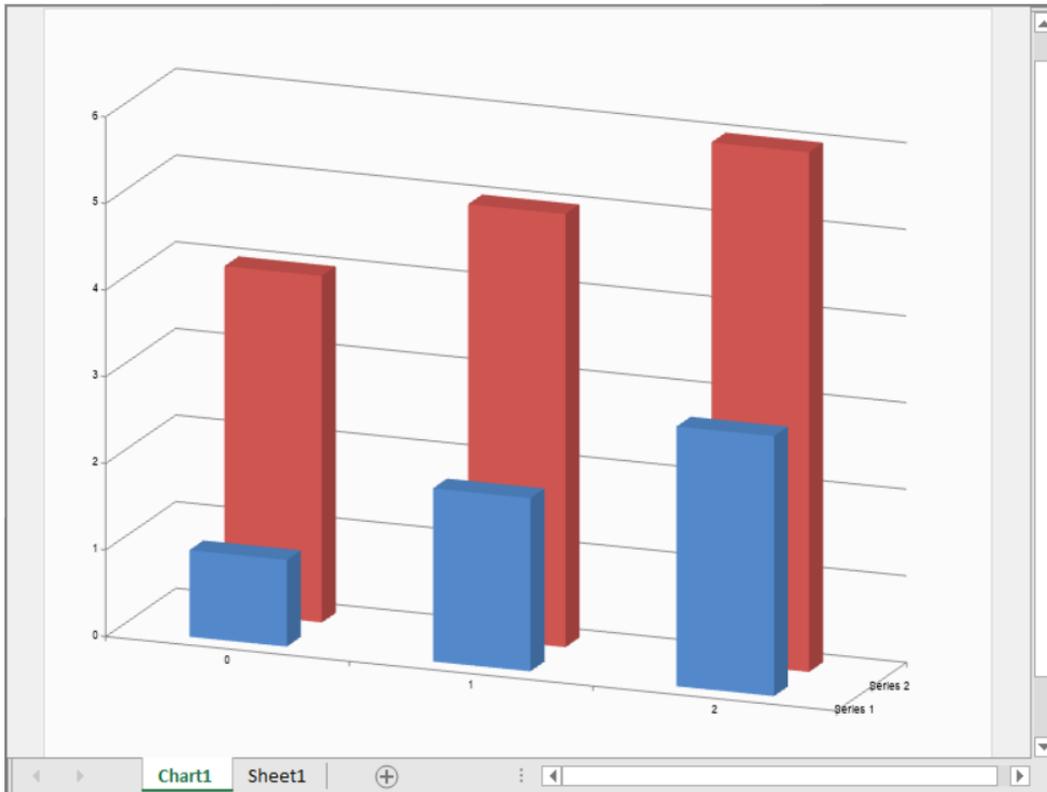
ChartSheet represents a sheet that contains only a chart. To add a ChartSheet in the workbook, users first need to set the value of the **EnhancedShapeEngine** property to True in order to enable the new shape engine. You can then add a ChartSheet in a workbook using the **Charts.Add()** method of the **IWorkbook** Interface. Here, the **Charts** property returns the collection of charts in the specified workbook.

Note that you can fit the chart to the current viewport in the ChartSheet by setting the value of the **View.ZoomToFit** property to true as shown below:

```
fpSpread1.AsWorkbook().Charts[0].View.ZoomToFit = true;
```

However, if you want to embed a chart in a worksheet that contains other items, like gridlines, cells, data, etc., then you need to add a **Chart** to a worksheet instead.

The image below depicts a sample ChartSheet.



The following sub-sections discuss various ChartSheet operations in a workbook.

Adding ChartSheet Using Code

A ChartSheet can be added before or after the active sheet. While adding, you can specify the position and number of ChartSheets to add.

The following example code displays different scenarios of adding ChartSheets.

C#

```
fpSpread1.Features.EnhancedShapeEngine = true;
fpSpread1.AsWorkbook().ActiveSheet.SetValue(1, 1, new int[,] { { 1, 4 }, { 2, 5 }, { 3, 6 } });

//Add a chart before the active sheet
fpSpread1.AsWorkbook().Charts.Add();
fpSpread1.Sheets[0].Charts[0].DataFormula = "Sheet1!B2:C4";
fpSpread1.Sheets[0].Charts[0].ViewType = ChartViewType.View3D;

//Add 2 charts before Sheet1
fpSpread1.AsWorkbook().Charts.Add(before: 1, count: 2);
//or
//fpSpread1.AsWorkbook().Charts.Add(before: "Sheet1", count: 2);
fpSpread1.Sheets[1].Charts[0].DataFormula = "Sheet1!B2:B4";
fpSpread1.Sheets[2].Charts[0].DataFormula = "Sheet1!C2:C4";
//Add one chart after Sheet1
fpSpread1.AsWorkbook().Charts.Add(after: 3, count: 1);
//or
//fpSpread1.AsWorkbook().Charts.Add(after: "Sheet1", count: 1);
fpSpread1.Sheets[4].Charts[0].DataFormula = "Sheet1!B2:C2";
```

Adding ChartSheet at Runtime

You can add a ChartSheet in a workbook at runtime as well by following the steps below.

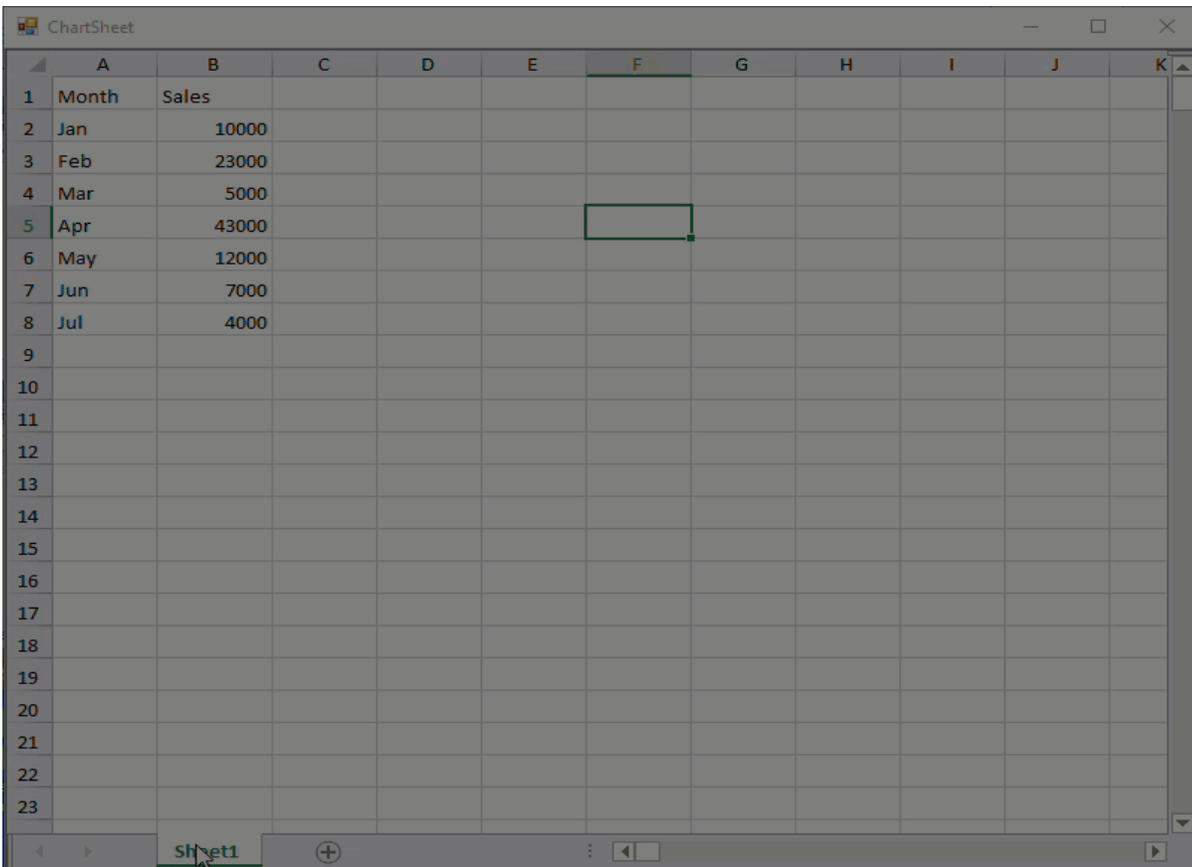
1. Run the code below to load a spread control with **EnhancedShapeEngine** and **TabStrip** editing enabled.

C#

```
fpSpread1.Features.EnhancedShapeEngine = true;  
fpSpread1.TabStrip.Editable = true;
```

1. Right-click on **Sheet1** in tab strip.
2. Choose the **Insert** option from the menu.
3. On the **Insert** dialog, select **Chart** and then click **OK**.
A new ChartSheet has been created as Chart1 before Sheet1.
4. Right-click on the **Chart Area** and choose the **Select Data...** option to select the data source.
5. Now go to Sheet1 containing data and select the required range.
6. The reference will appear in the text box in the **Select Data Source** dialog.
7. Click **OK**. The chart will appear in the ChartSheet.

The following GIF illustrates how to add a ChartSheet in a workbook at runtime.



Move Chart to ChartSheet

If a chart already exists in a worksheet, Spread also lets you move it to a ChartSheet.

The following example code depicts how to move a chart from the worksheet to the ChartSheet.

C#

```
fpSpread1.Features.EnhancedShapeEngine = true;
fpSpread1.AsWorkbook().ActiveSheet.SetValue(1, 1, new int[,] { { 1, 4 }, { 2, 3 }, { 3,
2 } });
fpSpread1.AsWorkbook().Charts.Add(before: 0, count: 2);
fpSpread1.Sheets[0].Charts[0].DataFormula = "Sheet1!C2:C3";
fpSpread1.Sheets[0].Charts[0].ViewType = ChartViewType.View3D;
fpSpread1.Sheets[1].Charts[0].DataFormula = "Sheet1!B2:B4";
IChart chart1 = fpSpread1.AsWorkbook().Charts[1];
IChart chart2 = fpSpread1.AsWorkbook().Charts[0];
chart1.Location(ChartLocation.Object, "Chart2"); // To move Chart1 to Chart2
chart2.Location(ChartLocation.NewSheet, "NewChartSheet"); // To move Chart2 to new
sheet
```

Get ChartSheet from Charts/Sheets Collection

A ChartSheet can also be obtained from a charts collection as well as from the Sheets collection.

The following example code depicts how to retrieve ChartSheet from the Charts collection and from the Sheets collection.

C#

```
fpSpread1.Features.EnhancedShapeEngine = true;
fpSpread1.AsWorkbook().ActiveSheet.SetValue(1, 1, new int[,] { { 1, 4 }, { 2, 5 }, { 3,
6 } });
fpSpread1.AsWorkbook().Charts.Add();
fpSpread1.Sheets[0].Charts[0].DataFormula = "Sheet1!B2:C4";
IChart chart1 = fpSpread1.AsWorkbook().Charts[0]; // To get from charts collection
chart1 = (IChart)fpSpread1.AsWorkbook().Sheets[0]; // To get from sheets collection
```

Removing a Sheet

If you have multiple sheets, you can remove a sheet or remove several sheets from the Spread component. There must always be at least one sheet in the component.

In code, you can simply change the sheet count or you can explicitly remove the sheets by specifying their indexes. The sheet index is zero-based.

Removing an existing sheet does not change the default sheet names provided to the other sheets. For example, a Spread component with three sheets would by default name them Sheet1, Sheet2, and Sheet3. If you remove the second sheet, the names for the remaining sheets are Sheet1 and Sheet3. The indexes for the sheets are 0 and 1, because the sheet index is zero based.

You can also hide a sheet. For more information, refer to **Showing or Hiding a Sheet**.

To remove an existing sheet, complete the following instructions.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet to remove.
5. Click the **Remove** button to remove the sheet from the collection.
6. Click **OK** to close the editor.

Method to set

Call the Sheets shortcut object **Remove ('Remove Method' in the on-line documentation)** method (to remove the sheet from the **SheetViewCollection ('SheetViewCollection Class' in the on-line documentation)** for the component) and specify the sheet to remove.

Example

This example code removes the second sheet from a Spread component that has two or more sheets.

C#

```
// Remove the second sheet.  
fpSpread1.Sheets.Remove(fpSpread1.Sheets[1]);
```

VB

```
' Remove the second sheet.  
fpSpread1.Sheets.Remove(fpSpread1.Sheets(1))
```

Showing or Hiding a Sheet

If you have more than one sheet in the component, you can hide a sheet so that it is not displayed to the user. Even though it is not displayed, it is not removed from the component. There must be at least one sheet in the component. For information on adding a sheet, refer to **Adding a Sheet**.

Hiding a sheet does not change the default sheet names provided for the other sheets. For example, a Spread component with three sheets would by default name them Sheet1, Sheet2, and Sheet3. If you hide the second sheet, the names for the remaining sheets are Sheet1 and Sheet3.

Hiding a sheet does not remove it and does not affect formulas on that sheet or references to that sheet.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet to hide.
5. Select the **Visible** property in the property list, and then select false.
6. Click **OK** to close the editor.

Method to set

Set the Sheets shortcut object **Visible ('Visible Property' in the on-line documentation)** property for the sheet.

Example

This example code hides the second and fourth sheets in a Spread component that has eight sheets.

C#

```
private void Form1_Load(object sender, System.EventArgs e)  
{  
    // Set the Spread to have eight sheets.  
    fpSpread1.Sheets.Count = 8;  
}
```

```
// Hide the second and fourth sheets.  
fpSpread1.Sheets[1].Visible = false;  
fpSpread1.Sheets[3].Visible = false;  
}
```

VB

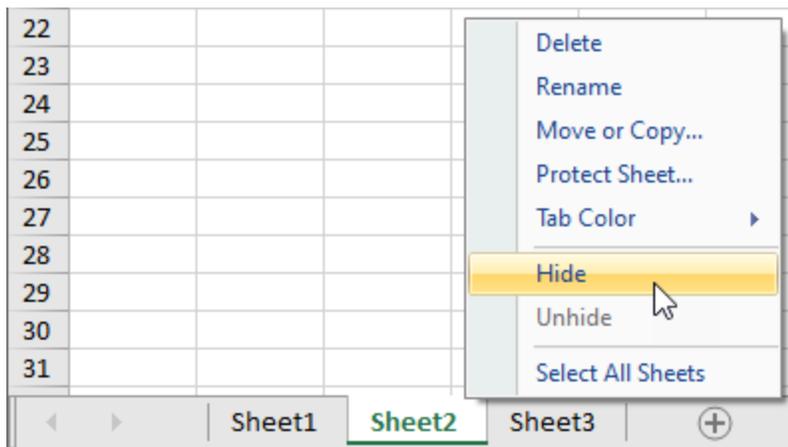
```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles  
MyBase.Load  
    ' Set the Spread to have eight sheets.  
    fpSpread1.Sheets.Count = 8  
    ' Hide the second and fourth sheets.  
    fpSpread1.Sheets(1).Visible = False  
    fpSpread1.Sheets(3).Visible = False  
End Sub
```

Using the Tab Strip Context Menu

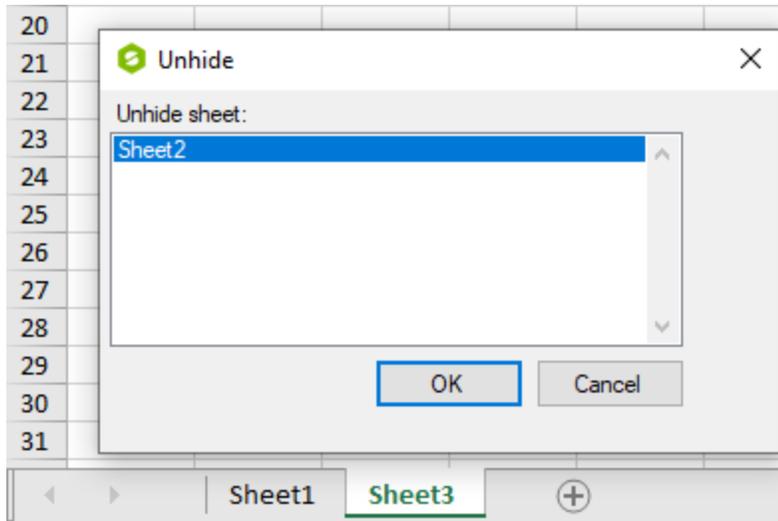
You can hide a sheet by using the tab strip's context menu and choosing 'Hide' option. The tab strip context menu can be enabled by setting the **Editable** option as true.

C#

```
fpSpread1.TabStrip.Editable = true;
```



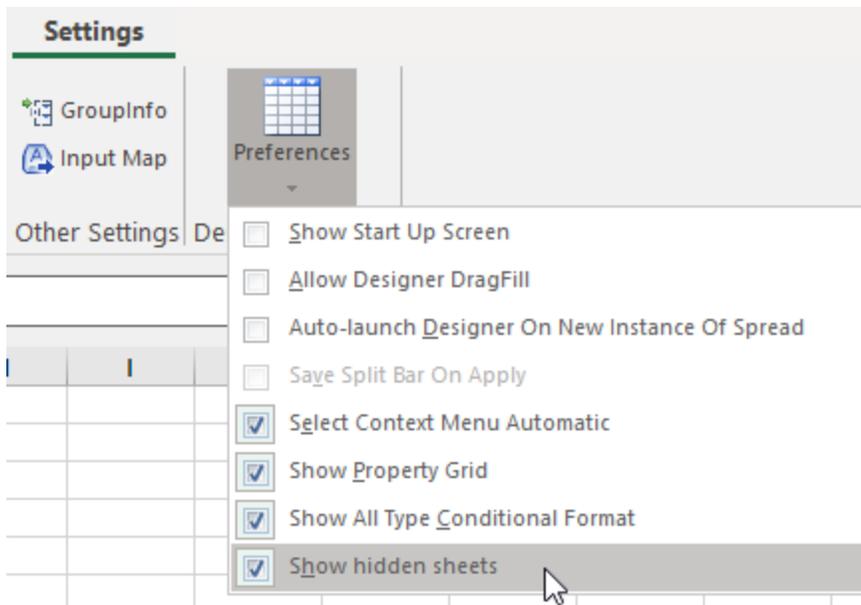
Similarly, you can select 'Unhide' from tab strip context menu to unhide sheets. The 'Unhide' dialog listing all the hidden sheets is opened from which you can select the ones you want to unhide.



Using the Spread Designer

1. Select the sheet tab for the sheet for which you want to set properties.
2. In the property list, select the **Visible** property.
3. Select **False**.
4. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

You can choose 'Hide' and 'Unhide' options from tab strip context menu in Spread Designer as well. However, these options can be disabled by checking the "Show hidden sheets" option in 'Preferences' dropdown list (unchecked by default). This will ensure that the hidden sheets are always shown.



Moving a Sheet

If you have multiple sheets, you can move a sheet. If you move the first sheet location to the last sheet location, then the other sheets are moved to the left. If you move the sheet location to the location of the sheet next to it, then this effectively swaps the sheets.

Specify the from and to location using the sheet index. The sheet index is zero-based.

Moving a sheet does not change the sheet name.

The **AllowSheetMove** ('**AllowSheetMove Property**' in the on-line documentation) property of the **FpSpread** class can be set to true to allow the user to move the sheets using the sheet tabs. You can also hide a sheet. For more information, refer to **Showing or Hiding a Sheet**.

You can prevent a user from moving a specific sheet with the **SheetDragMoving** ('**SheetDragMoving Event**' in the on-line documentation) and **SheetDragMoved** ('**SheetDragMoved Event**' in the on-line documentation) events. Select a sheet tab on the tab strip, drag the sheet tab to another tab, then release the mouse to move the sheet from the old index to the new index. The **SheetDragMoving** ('**SheetDragMoving Event**' in the on-line documentation) event occurs when the user starts dragging the sheet tab name. The **SheetDragMoved** ('**SheetDragMoved Event**' in the on-line documentation) event occurs right after the user moves the sheet. You can prevent specific sheets from being moved by setting the **Cancel** property to true in the **SheetDragMoving** ('**SheetDragMoving Event**' in the on-line documentation) event.

To move an existing sheet, complete the following instructions.

Method to set

Call the Sheets **Move** ('**Move Method**' in the on-line documentation) method (to move the sheet from one location to another).

Example

This example code moves the second sheet to the location of the third sheet.

C#

```
// Move sheet 2 to the location of sheet 3.
fpSpread1.Sheets.Count = 5;
fpSpread1.Sheets.Move(2, 3);
```

VB

```
' Move sheet 2 to the location of sheet 3.
fpSpread1.Sheets.Count = 5
fpSpread1.Sheets.Move(2, 3)
```

Selecting Multiple Sheets

Spread for WinForms allows you to select multiple worksheets at once. This feature is enabled by default and lets you perform various operations on selected sheets, like delete, hide, move, copy, set tab color etc. The below image shows the selected worksheets (Sheet1, 2 and 3) of a workbook.

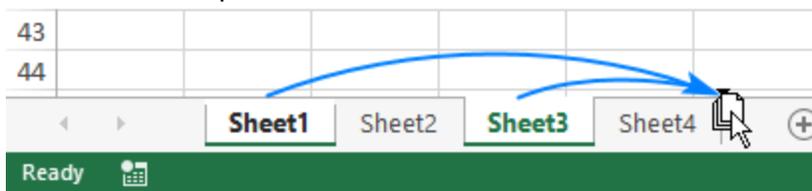


You can change the selected state of a worksheet by using keyboard and mouse click behavior. However, the active sheet is always selected and cannot be deselected.

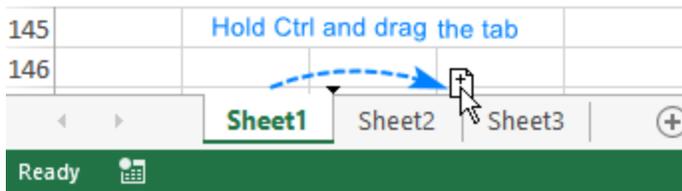
UI Operation	Selected Sheet Tab	Unselected Sheet Tab
Click	Sets the clicked sheet as active sheet.	Sets the clicked sheet as active sheet and deselects all the selected sheets. If all the sheets are selected and non active sheet is clicked, the clicked sheet becomes the active sheet and all other sheets are deselected.
Ctrl or Cmd + Click	Deselects the selected sheet (not in case of active sheet).	The clicked sheet is added as one of the selected sheets.
Shift + Click	In the cases of a selected or unselected sheet, this operation: <ul style="list-style-type: none"> • Selects all the sheets between the active sheet and clicked sheet. • Deselects all the worksheets that are beyond the above range. 	

You can also perform custom operations on multiple selected worksheets. For example:

- Select one or more sheet tabs and drag them to a new location to move them. In below image, Sheet1 and 3 are moved after Sheet4.



- Hold the Ctrl key and drag the sheet tab where you want to copy it. In below image, a copy of Sheet1 is created after Sheet2.



Method to set

Use the **Select** (bool replace = true) method of IWorksheets interface. The method takes an optional parameter replace, which:

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

Use the **SelectedSheets** property of **IWorkbook** interface to get the selected sheets in a workbook.

Example

This example code initializes six sheets in a workbook, selects multiple sheets by retaining or replacing them and gets the count of selected sheets.

C#

```
fpSpread1.Sheets.Count = 6;
```

```
//To select one sheet, you must specify the array string
fpSpread1.AsWorkbook().Worksheets[new string[] { "Sheet2" }].Select();

//select multiple sheets - replacing old sheet selection
fpSpread1.AsWorkbook().Worksheets["Sheet3", "Sheet4"].Select(true);

//select multiple sheets - keeping old sheet selection
fpSpread1.AsWorkbook().Worksheets["Sheet5", "Sheet6"].Select(false);

//get the count of selected sheets
MessageBox.Show(fpSpread1.AsWorkbook().SelectedSheets.Count.ToString());
```

VB

```
fpSpread1.Sheets.Count = 6

'To select one sheet, you must specify the array string
fpSpread1.AsWorkbook().Worksheets((New String() {"Sheet2"})).[Select]()

'select multiple sheets - replacing old sheet selection
fpSpread1.AsWorkbook().Worksheets("Sheet3", "Sheet4").[Select](True)

'select multiple sheets - keeping old sheet selection
fpSpread1.AsWorkbook().Worksheets("Sheet5", "Sheet6").[Select](False)

'get the count of selected sheets
MessageBox.Show(fpSpread1.AsWorkbook().SelectedSheets.Count.ToString())
```

Copying and Inserting a Sheet

You can copy and insert a sheet to the same Spread component or another Spread component. Spread allows you to create copies of sheets using the **Copy ('Copy Method' in the on-line documentation)** method of the **IWorksheets** interface.

Alternatively, you can also create your own custom **CopySheet** method to copy a sheet. In this scenario, to insert a copied sheet into the control, use the **Add ('Add Method' in the on-line documentation)** or **Insert ('Insert Method' in the on-line documentation)** method of the **SheetViewCollection ('SheetViewCollection Class' in the on-line documentation)** class referenced by the **Sheets ('Sheets Property' in the on-line documentation)** property of the **FpSpread** class after calling the **CopySheet** method.

Here is the example code for the **CopySheet** method.

C#

```
public FarPoint.Win.Spread.SheetView CopySheet(FarPoint.Win.Spread.SheetView sheet)
{
    FarPoint.Win.Spread.SheetView newSheet = null;
    if (sheet != null )
    {
        newSheet =
        FarPoint.Win.Serializer.LoadObjectXml(GetType(FarPoint.Win.Spread.SheetView),
        FarPoint.Win.Serializer.GetObjectXml(sheet, "CopySheet"), "CopySheet");
    }
    return newSheet;
}
```

Visual Basic

```
Public Function CopySheet(sheet As FarPoint.Win.Spread.SheetView) As
FarPoint.Win.Spread.SheetView
    Dim newSheet as FarPoint.Win.Spread.SheetView = Nothing
    If Not IsNothing(sheet) Then
        newSheet =
FarPoint.Win.Serializer.LoadObjectXml(GetType(FarPoint.Win.Spread.SheetView),
FarPoint.Win.Serializer.GetObjectXml(sheet, "CopySheet"), "CopySheet")
    End If
    Return newSheet
End Function
```



- The CopySheet method also copies all shapes on that sheet.
- Copying a sheet using the CopySheet method also copies the **NamedStyleCollection ('NamedStyleCollection Class' in the on-line documentation)** in the sheet, and creates separate copies of any **NamedStyle ('NamedStyle Class' in the on-line documentation)** objects in the collection that are private to the copy and not shared with the original NamedStyleCollection in the copied sheet.
- If you want to share the named styles instead of creating separate copies, you can assign the NamedStyleCollection you want to share to the **NamedStyles ('NamedStyles Property' in the on-line documentation)** property of the copy. You might also want to temporarily remove the NamedStyleCollection from the sheet being copied, so that it is not copied unnecessarily. This can be done by assigning the NamedStyleCollection to a variable, then setting the NamedStyles property to Nothing (null in C#), then making the copy, then assigning the variable back to the NamedStyles property.
- The **SpreadActions ('SpreadActions Class' in the on-line documentation)** class has options for the clipboard copy, cut, and paste of a sheet. You can assign copy, cut, and paste operations to any key input. For more information, kindly refer to "**Managing Keyboard Interaction**".

Using Spread Designer

The Spread Designer can be used to copy and paste a sheet at design time. Right-click on the sheet tab icon in the designer to bring up the Copy, Cut, and Paste context menu.

Copying Sheets with Formula References

Spread allows you to create copies of sheets having cell formulas. To do this, you can use the **Copy ('Copy Method' in the on-line documentation)** method of the **IWorksheets** interface. This method allows you to copy both the cell's destination and reference sheets at the same time, which then creates a reference state within the multiple copied sheets.

Using code

The following example code shows how to copy sheets with formula references.

C#

```
var sheet2 = fpSpread1.AsWorkbook().Worksheets.Add();
sheet2.ColumnHeader.RowCount = 2;
sheet2.ColumnHeader.Cells["A1"].Value = 5;
sheet2.Cells["A1"].Value = 4;
IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;
TestActiveSheet.Cells[0, 0].Formula = "Sum(Sheet2!A1,A2)";
TestActiveSheet.Cells[0, 1].Formula = "Sheet1[#Headers, [A1]]";
fpSpread1.AsWorkbook().Worksheets[0, 1].Copy(0);
```

Visual Basic

```
Dim sheet2 = fpSpread1.AsWorkbook().Worksheets.Add()
```

```

sheet2.ColumnHeader.RowCount = 2
sheet2.ColumnHeader.Cells("A1").Value = 5
sheet2.Cells("A1").Value = 4
Dim TestActiveSheet As IWorksheet = fpSpread1.AsWorkbook().ActiveSheet
TestActiveSheet.Cells(0, 0).Formula = "Sum(Sheet2!A1,A2)"
TestActiveSheet.Cells(0, 1).Formula = "Sheet1[#Headers, [A1]]"
fpSpread1.AsWorkbook().Worksheets(0, 1).Copy(0)

```

At Runtime

Follow the below steps to copy the sheets having cell formulas and referenced sheet:

1. Run the code below to load a spread control with TabStrip editing enabled to copy the sheets at runtime.

C#

```

fpSpread1.Width = 800;
fpSpread1.TabStripRatio = 0.8f;
fpSpread1.TabStrip.Editable = true;
var sheet2 = fpSpread1.AsWorkbook().Worksheets.Add();
sheet2.ColumnHeader.RowCount = 2;
sheet2.ColumnHeader.Cells["A1"].Value = 5;
sheet2.Cells["A1"].Value = 4;
IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;
TestActiveSheet.Cells[0, 0].Formula = "Sum(Sheet2!A1,A2)";
TestActiveSheet.Cells[0, 1].Formula = "Sheet2[#Headers, [A1]]";

```

Visual Basic

```

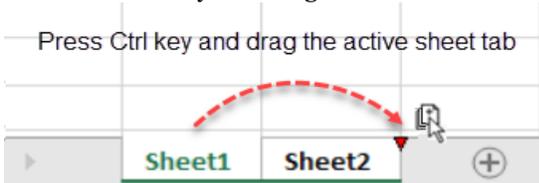
fpSpread1.Width = 800
fpSpread1.TabStripRatio = 0.8F
fpSpread1.TabStrip.Editable = True
Dim sheet2 = fpSpread1.AsWorkbook().Worksheets.Add()
sheet2.ColumnHeader.RowCount = 2
sheet2.ColumnHeader.Cells("A1").Value = 5
sheet2.Cells("A1").Value = 4
Dim TestActiveSheet As IWorksheet = fpSpread1.AsWorkbook().ActiveSheet
TestActiveSheet.Cells(0, 0).Formula = "Sum(Sheet2!A1,A2)"
TestActiveSheet.Cells(0, 1).Formula = "Sheet2[#Headers, [A1]]"

```

2. In the TabStrip, hold the **Ctrl** or **Shift** key to select **Sheet1** and **Sheet2**. Now copy the sheets by performing any of the following procedures.

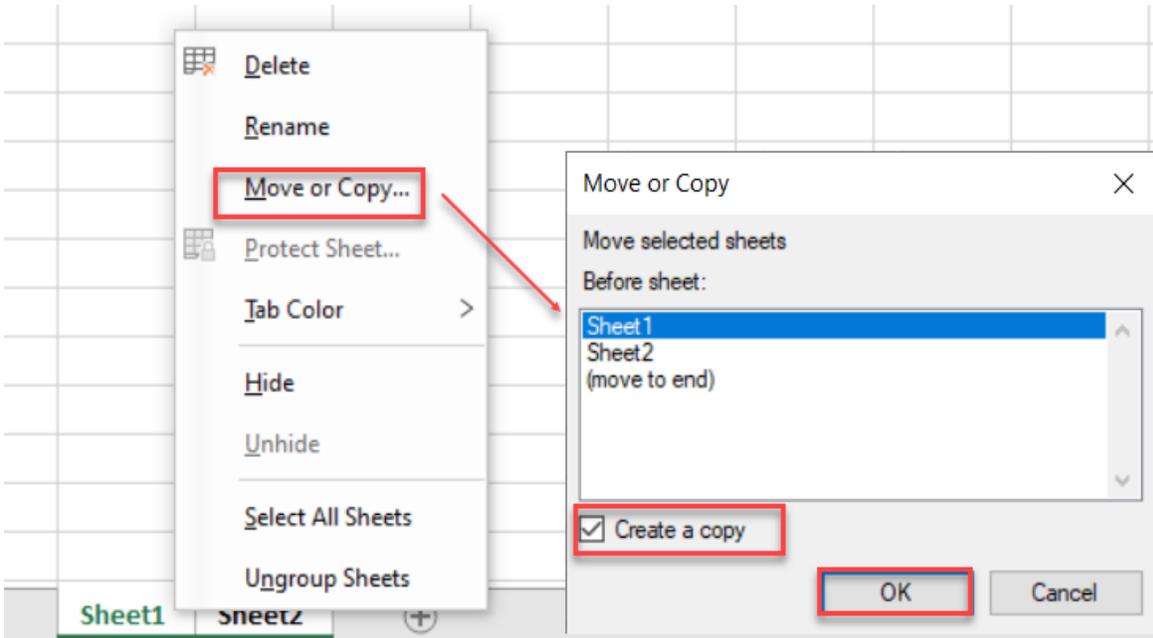
- **Drag the selected sheets tab**

1. Press the **Ctrl** key and drag the sheets tabs with the mouse to the desired location.



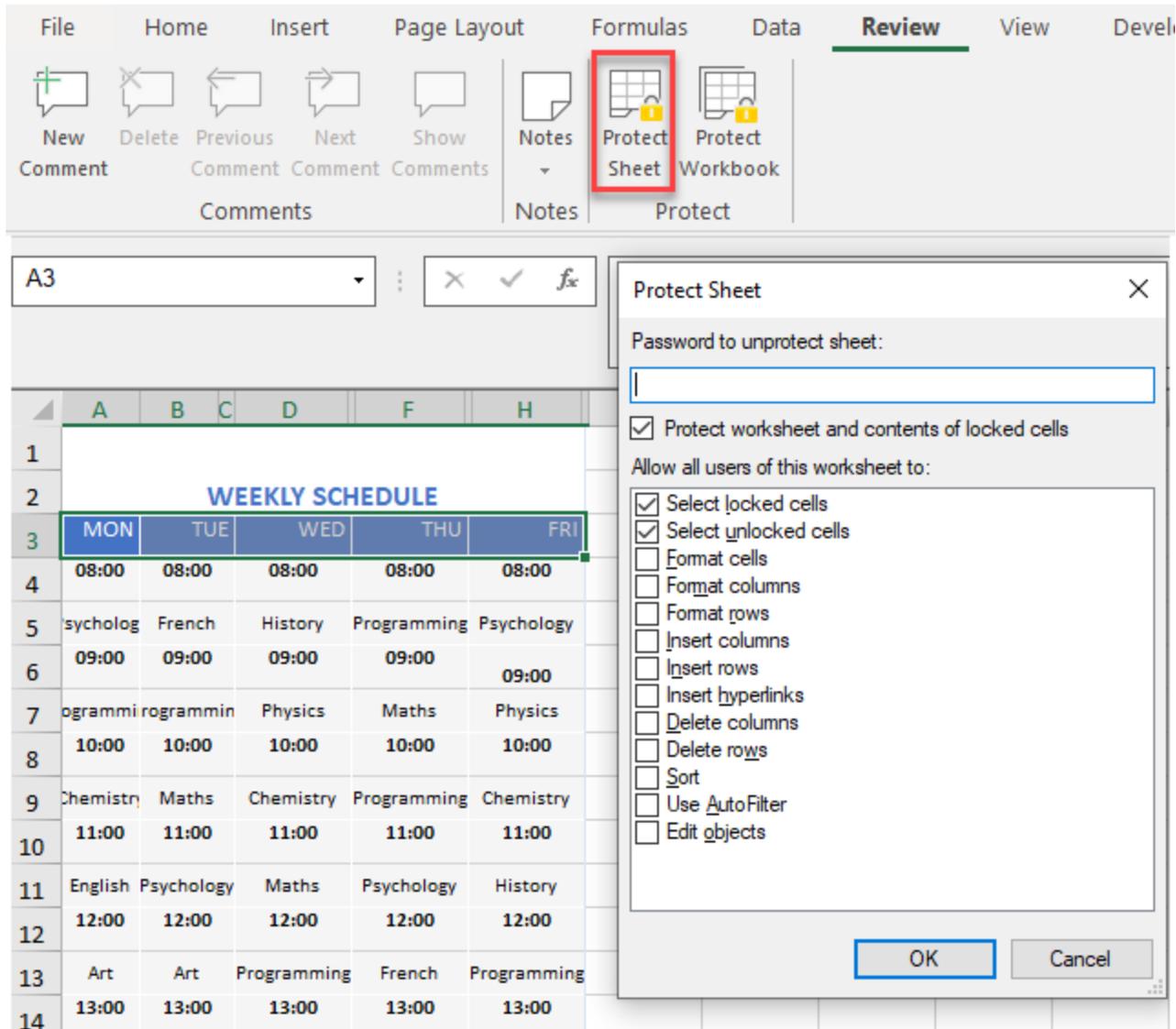
- **Use the Move or Copy dialog**

1. Right-click on the selected sheet tab(s) to open the context menu.
2. Open the **Move or Copy** dialog box.
3. Select the **Create a copy** checkbox and click **OK**, to create copies of selected sheets.



Protecting a Worksheet

You can protect a worksheet with a password in order to prevent other users from accidentally or deliberately changing, moving, or deleting data. In Spread for WinForms, the **Protect** method provides a variety of password protection and lock options that are available in the **Protect Sheet** dialog box.



Using Code

You can choose to protect or unprotect a worksheet using the **Protect** method which passes the **WorksheetLocks** enumeration as the parameter.

C#

```
fpSpread1.LegacyBehaviors = LegacyBehaviors.None;
activeSheet.Cells["A1:A4"].Value = 1;
activeSheet.Cells["B1:B4"].Value = 3;
activeSheet.Cells["C1:C4"].Value = 5;
fpSpread1.AsWorkbook().ActiveSheet.Protect(WorksheetLocks.DeleteColumns |
WorksheetLocks.DeleteRows);
```

VB

```
fpSpread1.LegacyBehaviors = LegacyBehaviors.None
activeSheet.Cells("A1:A4").Value = 1
activeSheet.Cells("B1:B4").Value = 3
activeSheet.Cells("C1:C4").Value = 5
fpSpread1.AsWorkbook().ActiveSheet.Protect(WorksheetLocks.DeleteColumns
Or WorksheetLocks.DeleteRows)
```

You can also set various types of protection options available for a worksheet using the **IProtection** interface. Each property from this interface gets a Boolean value, true or false indicating whether a particular action is allowed on a protected worksheet. For example, you can set the **AllowDeletingColumns** property as shown below:

C#

```
fpSpread1.AsWorkbook().ActiveSheet.Protect(WorksheetLocks.DeleteColumns);
GrapeCity.Spreadsheet.IProtection protection =
fpSpread1.AsWorkbook().ActiveSheet.Protection;
var value = protection.AllowDeletingColumns;
MessageBox.Show(value.ToString());
```

VB

```
fpSpread1.AsWorkbook().ActiveSheet.Protect(WorksheetLocks.DeleteColumns)
Dim protection As GrapeCity.Spreadsheet.IProtection =
fpSpread1.AsWorkbook().ActiveSheet.Protection
Dim value = protection.AllowDeletingColumns
MessageBox.Show(value.ToString())
```

In the above example, the `MessageBox` will show a false value as the worksheet has already been locked and protected to prevent column deletion.

Using Runtime UI

You can implement the **Protect** method in the runtime UI by setting the tab strip as editable. This allows you to open the **Protect Sheet** dialog.

C#

```
fpSpread1.TabStrip.Editable = true;
```

VB

```
fpSpread1.TabStrip.Editable = True
```

The following GIF illustrates opening protect sheet dialog box at runtime.

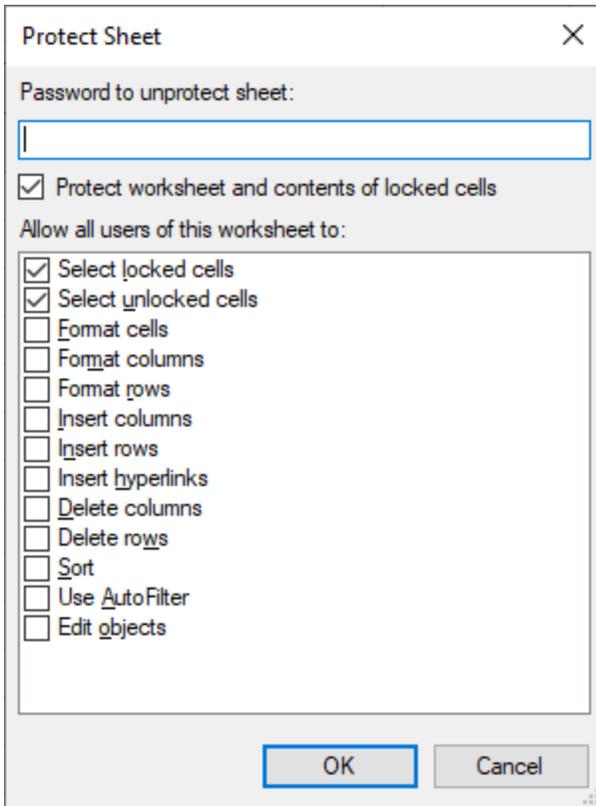
	A	B	C	D	E
1	Revenue (Sales)				
2	Category 1	\$2,30,228	19.9		
3	Category 2	\$1,25,432	10.9		
4	Category 3	\$1,72,289	14.9		
5	Category 4	\$1,70,089	14.7		
6	Category 5	\$1,61,276	14		
7	Category 6	\$1,33,274	11.5		
8	Category 7	\$1,61,447	14		
9	Total Revenue (Sales)	\$11,54,035	100		
10					
11					
12					
13					
14					
15					
16					
17					

Using Spread Designer

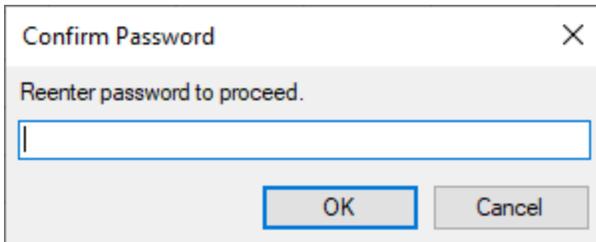
To access the **Protect Sheet** dialog, navigate to the **Protect Sheet** button in the **Protect** group under the **Review** tab of the Spread ribbon.

Steps to activate or deactivate protect sheets options

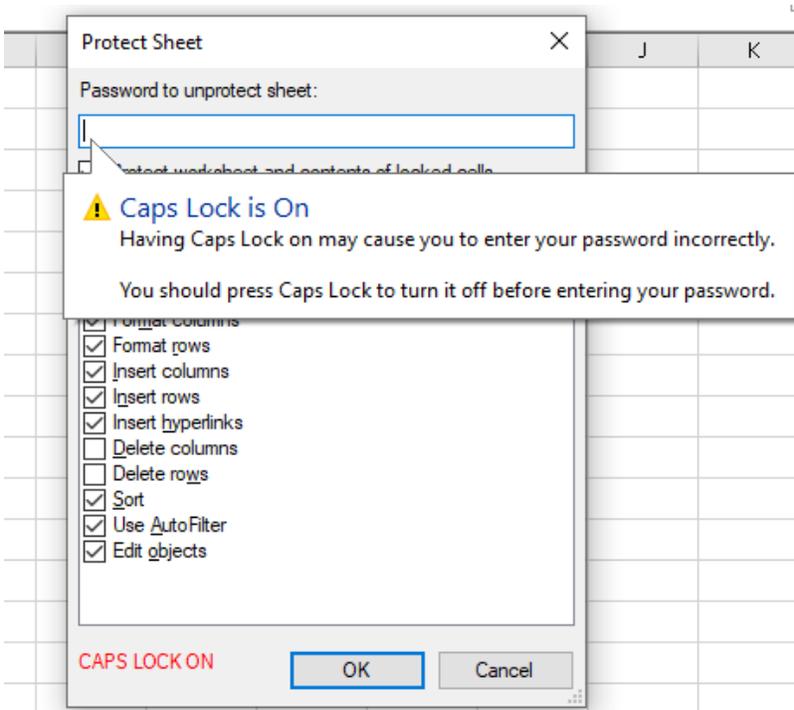
1. Select the multiple options you want to implement while protecting the worksheet from the Protect Sheet dialog box. Enter the password to protect the worksheet.



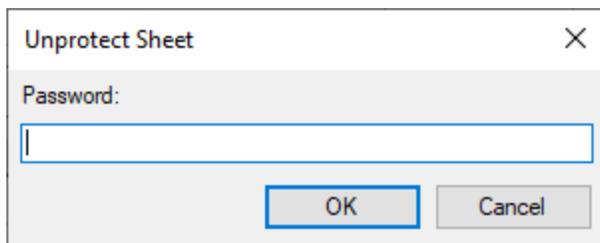
2. When you enter password to protect the worksheet, another dialog appears to confirm the password.



3. If the caps lock is on, a label indicator "**CAPS LOCK ON**" appears on the left side of the dialog box.



Once a password is successfully set, you can access the protected worksheet, but cannot make any changes to the sheet. To unprotect the worksheet enter the set password using the **Unprotect Sheet** dialog.



Limitations

Currently worksheet protection options do not include protection of PivotTable, PivotChart, and Edit Scenarios, but still support import/export for those options. The worksheet protection options work only when LegacyBehaviors.Protect is not set, otherwise the old behavior is applied.

Form Controls

Spread for WinForms provides several form controls for dialog sheets such as buttons, combo boxes, check boxes, spin button, list boxes, option buttons, group boxes, labels and scroll bars. With form controls, users can insert objects into worksheets that can work with data. Also, form controls make it easier for the users to interact with the cell data in the Spread worksheet.

Adding Form Controls using AddFormControl method

You can add a Form Control via the **AddFormControl** method of the **IShape** interface. This method passes the **FormControl** enumeration as a parameter.

Let's take a look at an example on how to add a simple button form control to the worksheet.

C#

```
// Set Enhanced Shape Engine to true
```

```
fpSpread1.Features.EnhancedShapeEngine = true;
// Add Button control
fpSpread1.AsWorkbook().ActiveSheet.Cells[0, 0].Value = "Button";
var btn = fpSpread1.AsWorkbook().ActiveSheet.Shapes.AddFormControl(FormControl.Button, 80, 40,
100, 20);
btn.ControlFormat.Enabled = false;
btn.Name = "Button Shape";
btn.CanMove = Moving.None;
btn.CanSize = Sizing.Width;
```

Similarly you can also add other controls using the **AddFormControl** and passing the suitable enumeration parameter for the specific control.

Adding Form Controls using IWorksheet interface

You can also add form controls to the worksheet using the **IWorksheet** interface, which represents a worksheet. For this purpose, the **IWorksheet** interface provides properties to add form controls.

Form Control UI

Button

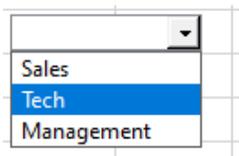


Using the API

C#

```
// Button
var button =
fpSpread1.AsWorkbook().ActiveSheet.Buttons.Add(0, 200,
200, 50);
button.Text = "Click Me";
```

ComboBox



C#

```
var dropDown =
fpSpread1.AsWorkbook().ActiveSheet.DropDowns.Add(70, 170,
105, 20);
dropDown.AddItem("Sales");
dropDown.AddItem("Tech");
dropDown.AddItem("Management");
```

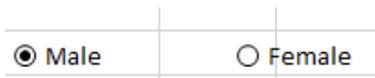
CheckBox



C#

```
var chkBox =
fpSpread1.AsWorkbook().ActiveSheet.CheckBoxes.Add(100,
130, 100, 20);
chkBox.Text = "FootBall";
```

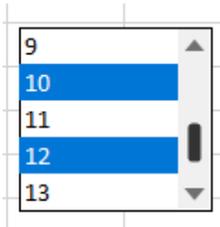
OptionButton



C#

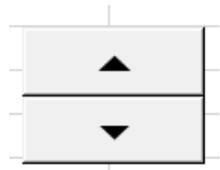
```
var opt1 =
fpSpread1.AsWorkbook().ActiveSheet.OptionButtons.Add(0,
75, 100, 20);
opt1.Text = "Male";
opt1.Checked = true;
var opt2 =
fpSpread1.AsWorkbook().ActiveSheet.OptionButtons.Add(100,
75, 100, 20);
opt2.Text = "Female";
```

ListBox

**C#**

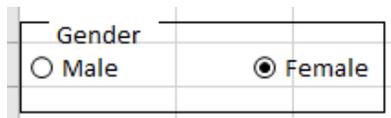
```
//ListBox
fpSpread1.AsWorkbook().ActiveSheet.Cells[7, 0].Value =
"Department:";
var list =
fpSpread1.AsWorkbook().ActiveSheet.ListBoxes.Add(70, 170,
105, 20);
list.MultiSelect =
GrapeCity.Spreadsheet.Drawing.SelectionMode.Simple;
list.ListFillRange = "E1:E14";
list.Enabled = true;
list.Height = 100;
```

SpinButton

**C#**

```
var spinner =
fpSpread1.AsWorkbook().ActiveSheet.Spinners.Add(130, 25,
50, 50);
spinner.Value = 18;
spinner.LinkedCell = "B2";
```

GroupBox

**C#**

```
//OptionBox
var grpGender =
fpSpread1.AsWorkbook().ActiveSheet.GroupBoxes.Add(0, 60,
200, 50);
grpGender.Text = "Gender";
var opt1 =
fpSpread1.AsWorkbook().ActiveSheet.OptionButtons.Add(0,
75, 100, 20);
opt1.Text = "Male";
opt1.Checked = true;
var opt2 =
fpSpread1.AsWorkbook().ActiveSheet.OptionButtons.Add(100,
75, 100, 20);
opt2.Text = "Female";
```

Label

**C#**

```
var lbl =
fpSpread1.AsWorkbook().ActiveSheet.Labels.Add(150, 170,
200, 50);
lbl.Text = "I am a label";
```

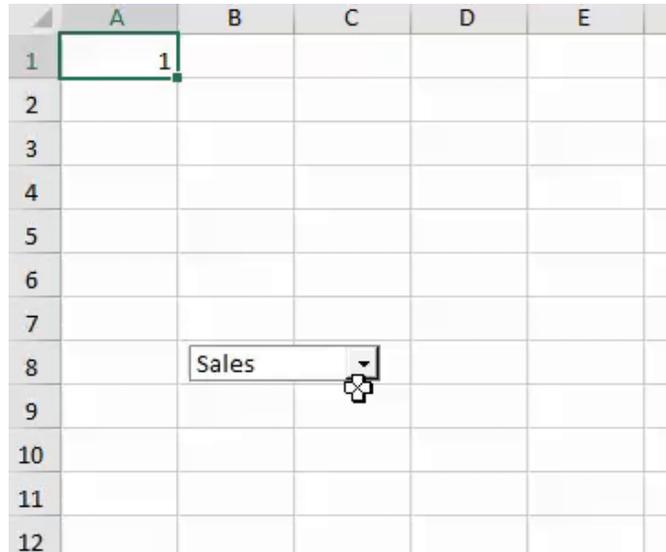
Scrollbar

**C#**

```
var scrollBar =
fpSpread1.AsWorkbook().ActiveSheet.ScrollBars.Add(150,
170, 250, 50);
scrollBar.Display3DShading = true;
```

Using the LinkedCell property

A major functionality of the Spread form control is that it allows you to link its value to a cell using the **LinkedCell** property.



The **LinkedCell** property sets the worksheet range linked to the control's value. For example, the code below depicts how to select an item from combo box, and how it impacts the value of A1 cell.

C#

```
var dropDown = fpSpread1.AsWorkbook().ActiveSheet.DropDowns.Add(70, 170, 105, 20);
dropDown.AddItem("Sales");
dropDown.AddItem("Tech");
dropDown.AddItem("Management");
dropDown.LinkedCell = "A1";
```

Using Events

Click Event

The **IFormControl** interface provides the **Click** event that occurs when the user clicks the control. For example, you can invoke the **DropDown_Click** event for the ComboBox for control.

The following code snippet depicts how you can click a ComboBox control and open a dropdown list of items fetched from a given cell range:

C#

```
public Form1()
{
    InitializeComponent();
    fpSpread1.Features.EnhancedShapeEngine = true;
    // Adding values on cells A1 -> A5
    fpSpread1.AsWorkbook().ActiveSheet.Cells["A1"].Value = 1;
    fpSpread1.AsWorkbook().ActiveSheet.Cells["A2"].Value = 2;
    fpSpread1.AsWorkbook().ActiveSheet.Cells["A3"].Value = 3;
    fpSpread1.AsWorkbook().ActiveSheet.Cells["A4"].Value = 4;
    fpSpread1.AsWorkbook().ActiveSheet.Cells["A5"].Value = 5;
    //Add Dropdown on first cell
    var dropDown = fpSpread1.AsWorkbook().ActiveSheet.DropDowns.Add(70, 0, 100, 25);
    dropDown.ListFillRange = "A1:A5"; //Fill Dropdown list with given range
    dropDown.Click += DropDown_Click; //Handle Click event
}
/// <summary>
/// Click event of dropdown
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
```

```
private void DropDown_Click(object sender, EventArgs e)
{
    MessageBox.Show("Changed");
}
```

Assigned Macros

Even though Spread cannot import and execute VBA code, users can execute their own logic if there is a function, which has the same name with the VBA function. But the function must satisfy the following conditions to be detected.

- The function signature must follow one of the two signatures given below:

```
void Func();
```

OR

```
void Func(object sender, EventArgs e);
```

- The function must be declared in the form, which contains the **FpSpread** instance.

So, once the user opens the XLSX file, the associated VBA function name will be imported. And, when the user interacts with the control and there is an associated function, that function will be invoked.

C#

```
public static void Button3_Click()
{
    MessageBox.Show("Assigned Macro Sample. Button3_Click");
}
public static void Button_click(object sender, EventArgs args)
{
    MessageBox.Show("Assigned Macro Sample Method. Button_click");
}
```

Resolve Macros

Let's say, the macro function doesn't belong to the form, then in that case, the user is able to resolve their own macro function using the **FpSpread.MacroResolve** event. Also, the delegate signature must be declared as mentioned previously. For example, the sample code snippets depicts the resolve macros scenario:

C#

```
private void Form1_Load(object sender, EventArgs e)
{
    //enable enhancedshapeengine for form controls
    fpSpread1.Features.EnhancedShapeEngine = true;
    fpSpread1.OpenExcel("testmacro.xlsm");
    fpSpread1.MacroResolve += FpSpread1_MacroResolve;
}
private void FpSpread1_MacroResolve(object sender, FarPoint.Win.Spread.DelegateResolveEventArgs e)
{
    switch (e.Name)
    {
    case "Button_click":
        e.Delegate = (Action<object, EventArgs>)MacroHndlers.Button_click;
        break;
    case "Button3_Click":
        e.Delegate = (Action)MacroHndlers.Button3_Click;
        break;
    }
}
```

Manipulate Form Controls at Runtime

You can invoke the **Format Control** dialog at runtime using the **FormatDialog** method of the **BuiltInDialogs** class. This is depicted in the code snippet below:

C#

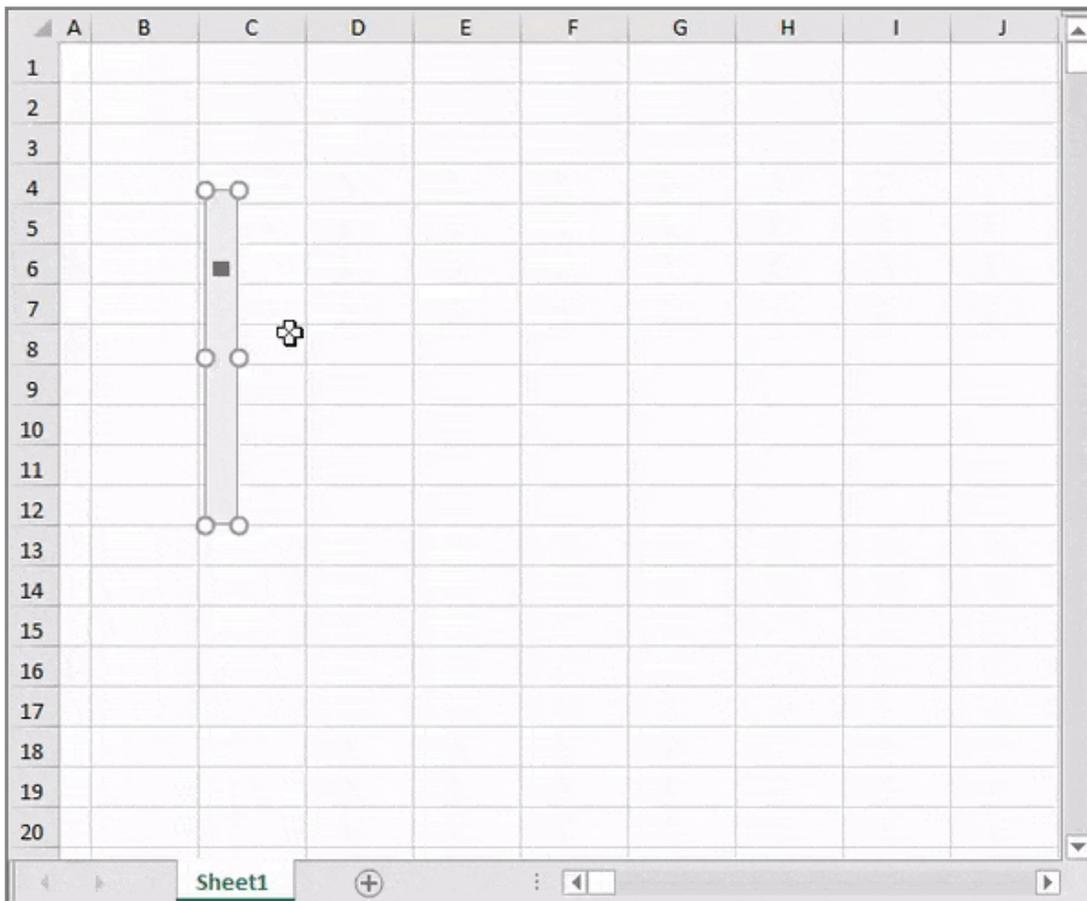
```
GrapeCity.Spreadsheet.FormControls.IScrollBar scrollBar =  
fpSpread1.AsWorkbook().ActiveSheet.ScrollBars.Add(100, 200, 20, 200);  
BuiltInDialogs.FormatControl(fpSpread1, scrollBar);
```

You can invoke the **Format Control** dialog at runtime in the following ways as well:

- Select the form control by using the "Ctrl+Click" shortcut key and press "Ctrl+1".

OR

- Right click the form control and then select the **Format Control** option from the context menu.



The Format Control dialog provides 4 common tabs: Size, Protection, Properties and Alt Text.

- **Size** tab: Change the height and width of the format control.
- **Protection** tab: Protect the form control using the Locked option.
- **Properties** tab: Change the properties of the form control.
- **Alt Text** tab: Add alternative text to the form control.

 **Note:** The **Form Control** Dialog might have more tabs depending on the control type.

Adding a Title and Subtitle to a Sheet

You can add a specially formatted area at the top of the component that includes a title, a subtitle or both. A title is set for the component, and a separate subtitle can be set for each sheet. The following figure illustrates a Spread component with a title and a subtitle set for the sheet.

The screenshot shows a spreadsheet window titled "FarPoint Spread Title". The spreadsheet has a light blue header row with the text "Sheet Only Subtitle". Below the header, the spreadsheet grid starts with columns labeled A through F and rows numbered 1 through 6. The cell at row 1, column A is selected and highlighted with a black border. The spreadsheet has a scroll bar on the right side.

The title is set using the **TitleInfo ('TitleInfo Property' in the on-line documentation)** property at the FpSpread level. The subtitle is set using the **TitleInfo ('TitleInfo Property' in the on-line documentation)** property at the sheet level.

Use the **TitleInfo ('TitleInfo Class' in the on-line documentation)** class and its members to display and customize the title and subtitles.

Method to set

Set the properties of the **TitleInfo ('TitleInfo Class' in the on-line documentation)** class.

Example

This example code sets and displays a title for the component and a subtitle for the sheet.

C#

```
// Show the title for the entire spreadsheet component.
fpSpread1.TitleInfo.Visible = true;
fpSpread1.TitleInfo.Text = "FarPoint Spread Title";
fpSpread1.TitleInfo.HorizontalAlign =
FarPoint.Win.Spread.CellHorizontalAlignment.Center;
// Show the subtitle for the individual sheet.
fpSpread1.Sheets[0].TitleInfo.Visible = true;
fpSpread1.Sheets[0].TitleInfo.Text = "Sheet Only Subtitle";
fpSpread1.Sheets[0].TitleInfo.HorizontalAlign =
FarPoint.Win.Spread.CellHorizontalAlignment.Center;
fpSpread1.Sheets[0].TitleInfo.BackColor = System.Drawing.Color.Aqua;
```

VB

```
' Show the title for the entire spreadsheet component.
fpSpread1.TitleInfo.Visible = True
fpSpread1.TitleInfo.Text = "FarPoint Spread Title"
fpSpread1.TitleInfo.HorizontalAlign =
FarPoint.Win.Spread.CellHorizontalAlignment.Center
' Show the subtitle for the individual sheet.
fpSpread1.Sheets(0).TitleInfo.Visible = True
fpSpread1.Sheets(0).TitleInfo.Text = "Sheet Only Subtitle"
fpSpread1.Sheets(0).TitleInfo.HorizontalAlign =
```

```
FarPoint.Win.Spread.CellHorizontalAlignment.Center
fpSpread1.Sheets(0).TitleInfo.BackColor = System.Drawing.Color.Aqua
```

Using the Spread Designer

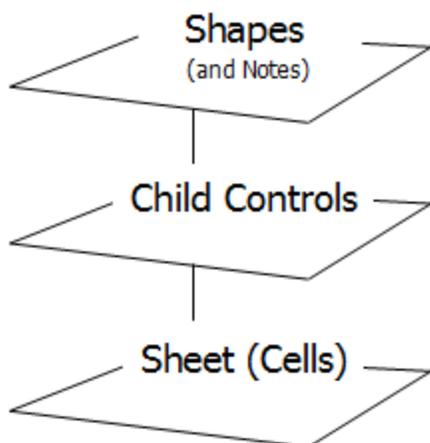
1. From the **Settings** menu, select the **Titles** icon in the **Spread Settings** section.
2. Specify the text for title and subtitle.
3. Select **OK** to close the dialog.
4. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Placing Child Controls on a Sheet

You can place controls on a sheet (not just the FpSpread component but a specific sheet) to provide more interaction with the user. Use the **AddControl** ('**AddControl Method**' in the on-line documentation) method for the Spread component or the sheet. Anything that can be derived from the **Control** class in the .NET framework can be hosted on a sheet in Spread. You must implement the **IEmbeddedControlSupport** ('**IEmbeddedControlSupport Interface**' in the on-line documentation) interface and set the following:

- **ActivationPolicy** ('**ActivationPolicy Property**' in the on-line documentation) - determines how the user can activate the object
- **CanMove** ('**CanMove Property**' in the on-line documentation) - determines whether the user can move the object once placed
- **CanSize** ('**CanSize Property**' in the on-line documentation) - determines whether the user can size the object once placed
- **ControlPaint** ('**ControlPaint Method**' in the on-line documentation) - determines how the object is represented when not active; similar to the paint method for a cell

The child control is placed on a sheet, according to the active cell, but is not anchored to that cell. Once placed, the control has an absolute position that does not change when the sheet is changed or the cell is either moved or removed. The child control is placed on a separate layer, the controls layer, which is separate from the data area (on which cells with data appear) and is separate from the drawing layer (on which shapes and other graphical elements appear).



With the **AllowChildControlDesign** ('**AllowChildControlDesign Property**' in the on-line documentation) property (in **FpSpread**), you can determine whether the user can interact with the child control in design mode. When design is allowed, the control can get focus and displays grab handles to allow moving and sizing, and displays a highlight border to indicate it is in focus. When the design is not allowed, the control simply is a live control that responds to the user clicking on it (or whatever policy is set with the **ActivationPolicy** ('**ActivationPolicy Property**' in the on-line documentation) property of **IEmbeddedControlSupport** ('**IEmbeddedControlSupport**

Interface' in the on-line documentation).

The child control is one of any number of controls that can be placed on the sheet. For the sheet there is a child control container (similar to the shape container for all the shapes on the sheet). You can enumerate through each control and override any property of the interface of that control. You can set events and work with event handlers.

The child controls layer is available only at run time. This feature is not available in the standalone version of the Spread Designer.

Some customization of the underlying **System.Control** class may be needed to make the embedded child controls to appear properly in your application.

For more information on the interface and enumeration of the child control layer, refer to these:

- **IEmbeddedControlSupport ('IEmbeddedControlSupport Interface' in the on-line documentation)**
- **ChildActivationPolicy ('ChildActivationPolicy Enumeration' in the on-line documentation)**

For more information on the methods and properties of the child control layer on the sheet, refer to these:

- **SheetView.AddControl ('AddControl Method' in the on-line documentation)**
- **SheetView.ClearControls ('ClearControls Method' in the on-line documentation)**
- **SheetView.GetControl ('GetControl Method' in the on-line documentation)**
- **SheetView.GetControlContainer ('GetControlContainer Method' in the on-line documentation)**
- **SheetView.RemoveControl ('RemoveControl Method' in the on-line documentation)**

For more information on the methods and properties of the child control layer for the Spread component, refer to these:

- **FpSpread.ChildControlActivated ('ChildControlActivated Event' in the on-line documentation)**
- **FpSpread.ChildControlDeactivated ('ChildControlDeactivated Event' in the on-line documentation)**
- **FpSpread.AddControl ('AddControl Method' in the on-line documentation)**
- **FpSpread.AllowChildControlDesign ('AllowChildControlDesign Property' in the on-line documentation)**
- **FpSpread.RemoveControl ('RemoveControl Method' in the on-line documentation)**

Displaying a Footer for Columns or Groups

You can show a column footer, a group footer, or both for the sheet and put information in the footer such as formulas or text. The column footer is an area at the bottom of the sheet. The group footer is an extra row of footer cells at the bottom of a sheet with grouping, if you are using the grouping feature.

For details on the API, refer to the **ColumnFooter ('ColumnFooter Property' in the on-line documentation)** property of the **SheetView ('SheetView Class' in the on-line documentation)** class and the various members of the **ColumnFooter ('ColumnFooter Class' in the on-line documentation)** class.

To calculate the column footer or group footer result with a formula, set the **SetAggregationType ('SetAggregationType Method' in the on-line documentation)** method of the **ColumnFooter** object to the correct formula type for that column. The following figure displays a group bar and a column footer with a formula in the column:

Double-click a column to group by that column.

	A	B	C	D	E	F
1	0	1	2	3	4	5
2	15	16	17	18	19	20
3	30	31	32	33	34	35
4	45	46	47	48	49	50
5	60	61	62	63	64	65
	Sum	428				

The group footer is an extra row that is displayed below the group after grouping by a column header. The **GroupFooterVisible** ('GroupFooterVisible Property' in the on-line documentation) property must be set to true after the group has been created. The **Grouped** ('Grouped Event' in the on-line documentation) event can be used to put information in the group footer after a user has created the group.

For more information on grouping, refer to **Managing Grouping of Rows of User Data**.

Properties Window

1. At design time, in the **Properties** window, select the **Sheets** property for the FpSpread component.
2. Click the button to display the **SheetView Collection Editor**.
3. Select the **ColumnFooter** property or the **GroupFooter** property or both in the **Property** list and set **Visible** to true.
4. Click **OK** to close the editor.

Using a Shortcut

Set the **Visible** ('Visible Property' in the on-line documentation) property of the ColumnFooter for the sheet.

Example

This example code displays a column footer and sets a span and a text color.

C#

```
fpSpread1.Sheets[0].RowCount = 10;
fpSpread1.Sheets[0].ColumnCount = 15;
// Show the column footer.
fpSpread1.Sheets[0].ColumnFooter.Visible = true;
fpSpread1.Sheets[0].ColumnFooter.RowCount = 2;
fpSpread1.Sheets[0].ColumnFooter.DefaultStyle.ForeColor = Color.Purple;
fpSpread1.Sheets[0].ColumnFooter.Columns[12].HorizontalAlignment =
FarPoint.Win.Spread.CellHorizontalAlignment.Left;
fpSpread1.Sheets[0].ColumnFooter.Cells[0, 12].RowSpan = 2;
fpSpread1.Sheets[0].ColumnFooter.Cells[0, 0].Value = "test";
```

VB

```
fpSpread1.Sheets(0).RowCount = 10
fpSpread1.Sheets(0).ColumnCount = 15
' Show the footer.
fpSpread1.Sheets(0).ColumnFooter.Visible = true
fpSpread1.Sheets(0).ColumnFooter.RowCount = 2
fpSpread1.Sheets(0).ColumnFooter.DefaultStyle.ForeColor = Color.Purple
fpSpread1.Sheets(0).ColumnFooter.Columns(12).HorizontalAlignment =
FarPoint.Win.Spread.CellHorizontalAlignment.Left
fpSpread1.Sheets(0).ColumnFooter.Cells(0, 12).RowSpan = 2
fpSpread1.Sheets(0).ColumnFooter.Cells(0, 0).Value = "test"
```

Method to set

1. Set the **Visible ('Visible Property' in the on-line documentation)** property of the ColumnFooter for the sheet.
2. Set the **SetAggregationType** method for the column.

Example

This example sums the values in the first column and displays them in the column footer. The example also sums the values in the second group and puts them in the group footer.

C#

```
private void Form1_Load(object sender, System.EventArgs e)
{
    fpSpread1.Sheets[0].RowCount=8;
    fpSpread1.Sheets[0].ColumnCount = 15;
    fpSpread1.Sheets[0].GroupBarInfo.Visible = true;
    fpSpread1.Sheets[0].AllowGroup = true;
    fpSpread1.Sheets[0].GroupFooterVisible = true;
    fpSpread1.Sheets[0].ColumnFooter.Visible = true;
    fpSpread1.Sheets[0].ColumnFooter.RowCount = 2;
    fpSpread1.Sheets[0].ColumnFooter.Columns[12].HorizontalAlignment =
FarPoint.Win.Spread.CellHorizontalAlignment.Left;
    fpSpread1.Sheets[0].ColumnFooter.Cells[0, 12].RowSpan = 2;
    //Value
    for (int r = 0; r < fpSpread1.Sheets[0].RowCount; r++)
    {
        for (int j = 0; j < fpSpread1.Sheets[0].ColumnCount; j++)
        {
            fpSpread1.Sheets[0].Models.Data.SetValue(r, j, j + r *
fpSpread1.Sheets[0].ColumnCount);
        }
    }
    int i = 0;
    fpSpread1.Sheets[0].ColumnFooter.SetAggregationType(0,1,
FarPoint.Win.Spread.Model.AggregationType.Sum);
    fpSpread1.Sheets[0].ColumnFooter.Cells[0, i].Value = "Sum";
```

```

}
private void fpSpread1_Grouped(object sender, EventArgs e)
    FarPoint.Win.Spread.Model.GroupDataModel gdm;
gdm =
(FarPoint.Win.Spread.Model.GroupDataModel) fpSpread1.ActiveSheet.Models.Data;
gdm.GroupFooterVisible = true;
FarPoint.Win.Spread.Model.Group g1 =
(FarPoint.Win.Spread.Model.Group) gdm.Groups[1];
((FarPoint.Win.Spread.Model.IAggregationSupport) g1.GroupFooter.DataModel)
.SetCellAggregationType(0, 0,
FarPoint.Win.Spread.Model.AggregationType.Sum);
fpSpread1.ActiveSheet.Models.Data = gdm;
}

```

VB

```

Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
    fpSpread1.Sheets(0).RowCount = 8
    fpSpread1.Sheets(0).ColumnCount = 15
    fpSpread1.Sheets(0).GroupBarInfo.Visible = True
    fpSpread1.Sheets(0).AllowGroup = True
    fpSpread1.Sheets(0).GroupFooterVisible = True
    fpSpread1.Sheets(0).ColumnFooter.Visible = True
    fpSpread1.Sheets(0).ColumnFooter.RowCount = 2
    fpSpread1.Sheets(0).ColumnFooter.Columns(12).HorizontalAlignment =
FarPoint.Win.Spread.CellHorizontalAlignment.Left
    'Value
    Dim r As Integer
    Dim j As Integer
    For r = 0 To fpSpread1.Sheets(0).RowCount
    For j = 0 To fpSpread1.Sheets(0).ColumnCount
    fpSpread1.Sheets(0).Models.Data.SetValue(r, j, j + r *
fpSpread1.Sheets(0).ColumnCount)
    Next j
    Next r
    Dim i As Integer
    i = 0
    fpSpread1.Sheets(0).ColumnFooter.SetAggregationType(0, 1,
FarPoint.Win.Spread.Model.AggregationType.Sum)
    fpSpread1.Sheets(0).ColumnFooter.Cells(0, i).Value = "Sum"
End Sub
Private Sub fpSpread1_Grouped(ByVal sender As Object, ByVal e As
System.EventArgs) Handles fpSpread1.Grouped
    Dim gdm As FarPoint.Win.Spread.Model.GroupDataModel
    Dim g1 As FarPoint.Win.Spread.Model.Group

```

```
gdm = fpSpread1.Sheets(0).Models.Data
gdm.GroupFooterVisible = True
g1 = gdm.Groups(1)
CType(g1.GroupFooter.DataModel,
FarPoint.Win.Spread.Model.IAggregationSupport).SetCellAggregationType(0,
0, FarPoint.Win.Spread.Model.AggregationType.Sum)
fpSpread1.ActiveSheet.Models.Data = gdm
End Sub
```

Example

This example code displays a column footer and adds a formula from a different sheet.

C#

```
fpSpread1.Sheets.Count = 3;
fpSpread1.Sheets[0].RowCount = 8;
fpSpread1.Sheets[0].ColumnCount = 15;
fpSpread1.Sheets[0].GroupBarInfo.Visible = true;
fpSpread1.Sheets[0].AllowGroup = true;
fpSpread1.Sheets[0].GroupFooterVisible = true;
fpSpread1.Sheets[0].ColumnFooter.Visible = true;
fpSpread1.Sheets[0].ColumnFooter.RowCount = 2;
fpSpread1.Sheets[0].ColumnFooter.Columns[12].HorizontalAlignment =
FarPoint.Win.Spread.CellHorizontalAlignment.Left;
fpSpread1.Sheets[0].ColumnFooter.Cells[0, 12].RowSpan = 2;
//Value
for (int r = 0; r < fpSpread1.Sheets[0].RowCount; r++)
{
    for (int j = 0; j < fpSpread1.Sheets[0].ColumnCount; j++)
    {
        fpSpread1.Sheets[0].Models.Data.SetValue(r, j, j + r *
fpSpread1.Sheets[0].ColumnCount);
    }
}
int i = 0;
fpSpread1.Sheets[1].RowCount = 10;
fpSpread1.Sheets[1].ColumnCount = 15;
fpSpread1.Sheets[1].GroupBarInfo.Visible = true;
fpSpread1.Sheets[1].AllowGroup = true;
fpSpread1.Sheets[1].GroupFooterVisible = true;
fpSpread1.Sheets[1].ColumnFooter.Visible = true;
fpSpread1.Sheets[1].ColumnFooter.RowCount = 2;
fpSpread1.Sheets[1].ColumnFooter.Columns[12].HorizontalAlignment =
FarPoint.Win.Spread.CellHorizontalAlignment.Left;
fpSpread1.Sheets[1].ColumnFooter.Cells[0, 12].RowSpan = 2;
//Value
for (int r = 0; r < fpSpread1.Sheets[1].RowCount; r++)
```

```

{
    for (int j = 0; j < fpSpread1.Sheets[1].ColumnCount; j++)
    {
        fpSpread1.Sheets[1].Models.Data.SetValue(r, j, j + r *
fpSpread1.Sheets[1].ColumnCount);
    }
}

fpSpread1.Sheets[0].ColumnFooter.Cells[0, i].Value = "2ndSum";
fpSpread1.Sheets[0].ColumnFooter.Cells[0, 1].Formula =
"SUM(Sheet2!A:A)";

```

VB

```

fpSpread1.Sheets.Count = 3
fpSpread1.Sheets(0).RowCount = 8
fpSpread1.Sheets(0).ColumnCount = 15
fpSpread1.Sheets(0).GroupBarInfo.Visible = True
fpSpread1.Sheets(0).AllowGroup = True
fpSpread1.Sheets(0).GroupFooterVisible = True
fpSpread1.Sheets(0).ColumnFooter.Visible = True
fpSpread1.Sheets(0).ColumnFooter.RowCount = 2
fpSpread1.Sheets(0).ColumnFooter.Columns(12).HorizontalAlignment =
FarPoint.Win.Spread.CellHorizontalAlignment.Left
'Value
Dim r As Integer
Dim j As Integer
For r = 0 To fpSpread1.Sheets(0).RowCount
    For j = 0 To fpSpread1.Sheets(0).ColumnCount
        fpSpread1.Sheets(0).Models.Data.SetValue(r, j, j + r *
fpSpread1.Sheets(0).ColumnCount)
    Next j
Next r
fpSpread1.Sheets(1).RowCount = 10
fpSpread1.Sheets(1).ColumnCount = 15
fpSpread1.Sheets(1).GroupBarInfo.Visible = True
fpSpread1.Sheets(1).AllowGroup = True
fpSpread1.Sheets(1).GroupFooterVisible = True
fpSpread1.Sheets(1).ColumnFooter.Visible = True
fpSpread1.Sheets(1).ColumnFooter.RowCount = 2
fpSpread1.Sheets(1).ColumnFooter.Columns(12).HorizontalAlignment =
FarPoint.Win.Spread.CellHorizontalAlignment.Left
fpSpread1.Sheets(1).ColumnFooter.Cells(0, 12).RowSpan = 2
'Value
For r = 0 To fpSpread1.Sheets(1).RowCount
    For j = 0 To fpSpread1.Sheets(1).ColumnCount
        fpSpread1.Sheets(1).Models.Data.SetValue(r, j, j + r *

```

```

fpSpread1.Sheets(1).ColumnCount)
    Next j
Next r
Dim i As Integer
i = 0
fpSpread1.Sheets(0).ColumnFooter.Cells(0, i).Value = "2ndSum"
fpSpread1.Sheets(0).ColumnFooter.Cells(0, 1).Formula = "SUM(Sheet2!A:A)"

```

Set Column Footer Format

In Spread, you can show the column footer format in flat style using the **NumberFormat** property of the **IRange** interface.

	A	B	C	D	E	F	G
1		1000					
2		1000					
3		1000					
4		1000					
5		1000					
6		1000					
7		1000					
8		1000					
9		1000					
		11,000					

For example, you can display a comma as a thousands separator in the footer number format.

C#

```

// Set flat style in Spread
fpSpread1.LegacyBehaviors = LegacyBehaviors.None;
fpSpread1.Reset();
fpSpread1.Sheets[0].Cells[0, 1, 10, 1].Value = 1000;
fpSpread1.Sheets[0].ColumnFooterVisible = true;
fpSpread1.Sheets[0].ColumnFooterRowCount = 1;
fpSpread1.Sheets[0].ColumnFooter.SetAggregationType(0, 1,
FarPoint.Win.Spread.Model.AggregationType.Sum);
// Set number format in column footer
fpSpread1.AsWorkbook().ActiveSheet.ColumnFooter.Cells[0, 1].NumberFormat
= "#,##0";

```

VB

```

' Set flat style in Spread

```

```

fpSpread1.LegacyBehaviors = LegacyBehaviors.None
fpSpread1.Reset()
fpSpread1.Sheets(0).Cells(0, 1, 10, 1).Value = 1000
fpSpread1.Sheets(0).ColumnFooterVisible = True
fpSpread1.Sheets(0).ColumnFooterRowCount = 1
fpSpread1.Sheets(0).ColumnFooter.SetAggregationType(0, 1,
FarPoint.Win.Spread.Model.AggregationType.Sum)
' Set number format in column footer
fpSpread1.AsWorkbook().ActiveSheet.ColumnFooter.Cells(0, 1).NumberFormat
= "#,##0"

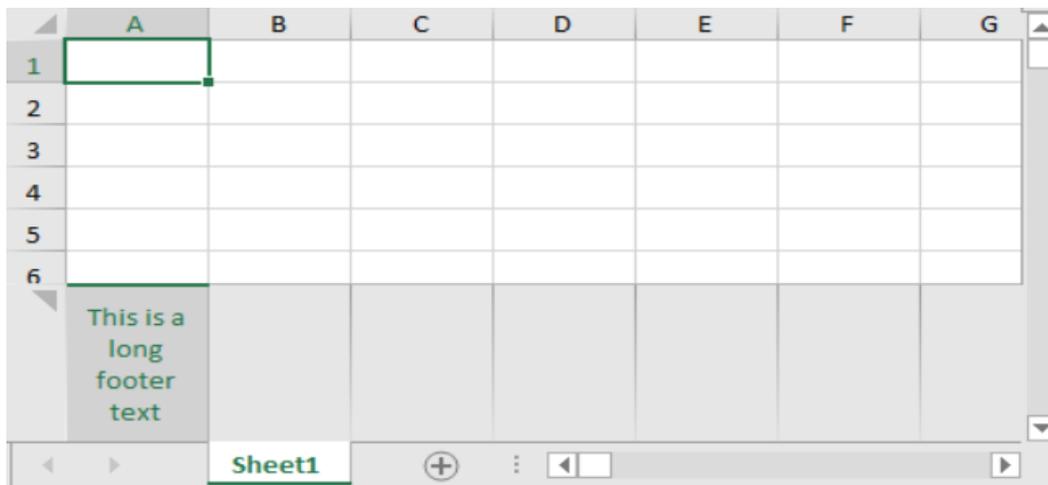
```

Wrap Text in Column Footers

You can customize text wrap in a column footer by setting the **WrapText** property of **IRange** (**IRange Interface** in the on-line documentation) Interface to true. Note that to view the wrapped text properly in the cell, you need to adjust the row height using the **RowHeight** property in the **ColumnFooter** class.

Before applying this **WrapText** property, ensure that the value of **ColumnHeaderRenderer.WordWrap2** property of the [SpreadSkin](#) class is set to null. By default, the **ColumnHeaderRenderer.WordWrap2** value is null for the default skin only.

The following image depicts a preview of the wrapped text in the column footer.



Use the sample codes below to wrap the column footer text using the **IRange.WrapText** property.

C#

```

// Wrap text in column footer
activeSheet.ColumnFooter.Visible = true;
activeSheet.ColumnFooter.Cells.RowHeight = 90;
activeSheet.ColumnFooter.Cells[0, 0].Value = "This is a long footer
text";
activeSheet.ColumnFooter.Cells[0, 0].WrapText = true;

```

VB

```

' Wrap text in column footer

```

```
activeSheet.ColumnFooter.Visible = True
activeSheet.ColumnFooter.Cells.RowHeight = 90
activeSheet.ColumnFooter.Cells(0, 0).Value = "This is a long footer text"
activeSheet.ColumnFooter.Cells(0, 0).WrapText = True
```

Adding a Tag to a Sheet

You can add an application tag to a sheet. If you prefer, you can associate data with any cell in the spreadsheet, or the cells in a column, a row, or the entire spreadsheet. The string data can be used to interact with a cell or to provide information to the application you create. The sheet data, or sheet tag, is similar to item data you can provide for the spreadsheet, columns, or rows.

Since the sheet tag is declared as an object, it is very flexible; it can be a number, a boolean, a string, or an instance of a class.

For more information on tags, refer to the **Tag ('Tag Property' in the on-line documentation)** property and the **GetCellFromTag ('GetCellFromTag Method' in the on-line documentation)** method in the **SheetView ('SheetView Class' in the on-line documentation)** class.

Working with 1-Based Indexing

Spread for WinForms provides extensive support for adding one or more worksheets with 1-based indexing in a workbook. This type of indexing facilitates effective conversion of SpreadCom or Excel VBA to new APIs. Users can add worksheets with 1-based indexing with the help of some wrapper classes. These classes allow users to convert the indexed parameters and forward them to the old implementation.

By default, each workbook possesses a single worksheet named Sheet 1 with sheet index 0.

Inserting worksheets with 1-based indexing is useful especially when the users don't intend to initiate the workbook with the default sheet index as 0 and also don't want the cells nomenclature to be started from zero (for instance - By default, the A1 cell is referenced as (0,0) but using 1-based indexing, the same cell will be referred as (1,1)). In such a scenario, 1-based indexing eliminates all the confusion with the nomenclature of the worksheets and the cells of the worksheet in a workbook.

Example

This example code creates a wrapper class and shows how to add a sheet using 1-based indexing.

C#

```
// CreateBase1Object Wrapper class to support 1 based indexing
IWorkbook workbook = WorkbookSet.CreateBase1Object(fpSpread1.AsWorkbook());
workbook.Worksheets[1].Cells[1, 1].Value = "Test";
```

VB

```
'CreateBase1Object Wrapper class to support 1 based indexing
Dim workbook As IWorkbook = WorkbookSet.CreateBase1Object(fpSpread1.AsWorkbook())
workbook.Worksheets(1).Cells(1, 1).Value = "Test"
```

 **Note:** The **LinkSources ('LinkSources Property' in the on-line documentation)** property uses zero-based indexing and not 1-based indexing because it is an array.

Customizing Clipboard Operation Options

There are several Clipboard operations (such as copy, cut, and paste) that are automatically set for a sheet with default settings; they are built-in to Spread. But Spread also gives you the ability to customize these operations on actual applications that you develop, depending on your specific needs. You can customize how the user can interact with the contents of the Clipboard when users perform copying and pasting actions in cells in the spreadsheet. You can implement those operations at your discretion in code. These customizations include:

- Deactivating clipboard operations
- Excluding headers from clipboard operations
- Obtaining the clipboard contents
- Deactivating pasting
- Changing the scope of pasting
- Performing the clipboard operations in code

The following members are used to determine Clipboard-related interaction:

- **ClipboardOptions** ('ClipboardOptions Property' in the on-line documentation) property (and **ClipboardOptions** ('ClipboardOptions Enumeration' in the on-line documentation) enumeration)
- **AutoClipboard** ('AutoClipboard Property' in the on-line documentation) property
- **ClipboardCopyOptions** ('ClipboardCopyOptions Enumeration' in the on-line documentation) enumeration
- **ClipboardPasteOptions** ('ClipboardPasteOptions Enumeration' in the on-line documentation) enumeration

The spreadsheet methods in the **SheetView** ('SheetView Class' in the on-line documentation) class that involve Clipboard operation are:

- **ClipboardCopy** ('ClipboardCopy Method' in the on-line documentation), which copies the contents from the sheet to the Clipboard
- **ClipboardCut** ('ClipboardCut Method' in the on-line documentation), which cuts the contents from the sheet to the Clipboard
- **ClipboardPaste** ('ClipboardPaste Method' in the on-line documentation), which pastes the contents from the Clipboard to the sheet

The corresponding shape methods in the **SheetView** ('SheetView Class' in the on-line documentation) class that involve Clipboard operation are:

- **ClipboardCopyShape** ('ClipboardCopyShape Method' in the on-line documentation), which copies the active shape to the Clipboard
- **ClipboardCutShape** ('ClipboardCutShape Method' in the on-line documentation), which cuts the active shape to the Clipboard
- **ClipboardPasteShape** ('ClipboardPasteShape Method' in the on-line documentation), which pastes the shape from the Clipboard

If there are locked cells in the range to cut or paste then the Clipboard operation is not performed.



The .NET version of the product handles Clipboard operations differently from the way that the COM version does.

You can also set how some Clipboard-related features perform when the user is in edit mode in a cell on the Spread. You can set whether the pop-up menu appears while in edit mode within a cell using the **AutoMenu** property in **SuperEditBase** class and whether the user can perform Clipboard operations with the shortcut keys with the **AllowClipboardKeys** property in **SuperEditBase** class.

For more information about copying and pasting, see **Copying Data on a Sheet**.

Deactivating Clipboard Operations

You can set whether the shortcut keys are available to the end user for them to use to perform Clipboard operations by

setting the **AutoClipboard** property in the **FpSpread ('FpSpread Class' in the on-line documentation)** class. You can deactivate all the Clipboard operations with components by setting the **AutoClipboard** property to **False** in the **FpSpread ('FpSpread Class' in the on-line documentation)** class. The default setting is **True**. You can deactivate some Clipboard operations such as pasting (Ctrl+V) by deactivating the input map definition for short-cut keys and Clipboard operations (Ctrl+C, Ctrl+V, and Ctrl+X).

Excluding Headers from Clipboard Operations

You can set whether to include headers when using Clipboard operations by setting the **ClipboardOptions** property in the **FpSpread ('FpSpread Class' in the on-line documentation)** class and the **ClipboardOptions** enumeration. The default setting, **AllHeaders**, allows all headers to be included.

Obtaining the Clipboard Contents

Values which are copied on sheets are controlled by the **Clipboard** class which is provided from the .NET Framework. You can obtain Clipboard contents by using respective operations in this **Clipboard** class.

Cell data is copied onto the Clipboard in advance by calling **ClipboardCopy** method in the **SheetView ('SheetView Class' in the on-line documentation)** class at the **Load** event. Then, calling **GetDataObject** method enables you to obtain those Clipboard contents for use, such as for text format determination.

Deactivating Pasting

The **ClipboardPasting** event occurs when pasting is performed (Ctrl+V) on sheets. You can deactivate pasting by canceling this event under certain circumstances. You can prevent pasting by obtaining the timing when pasting is performed, and canceling the action.

Changing the Scope of Pasting

When cells are copied or cut to the Clipboard, all the data aspects including values, formats, and formulas, are available by default for pasting. You can configure to paste values only, for example, by changing the input map definitions. The default setting is **ClipboardPasteAll**, which enables pasting all the aspects of the data. The **ClipboardPasteOptions** enumeration allows you to set the scope of what is pasted when a Clipboard paste is performed by the user.

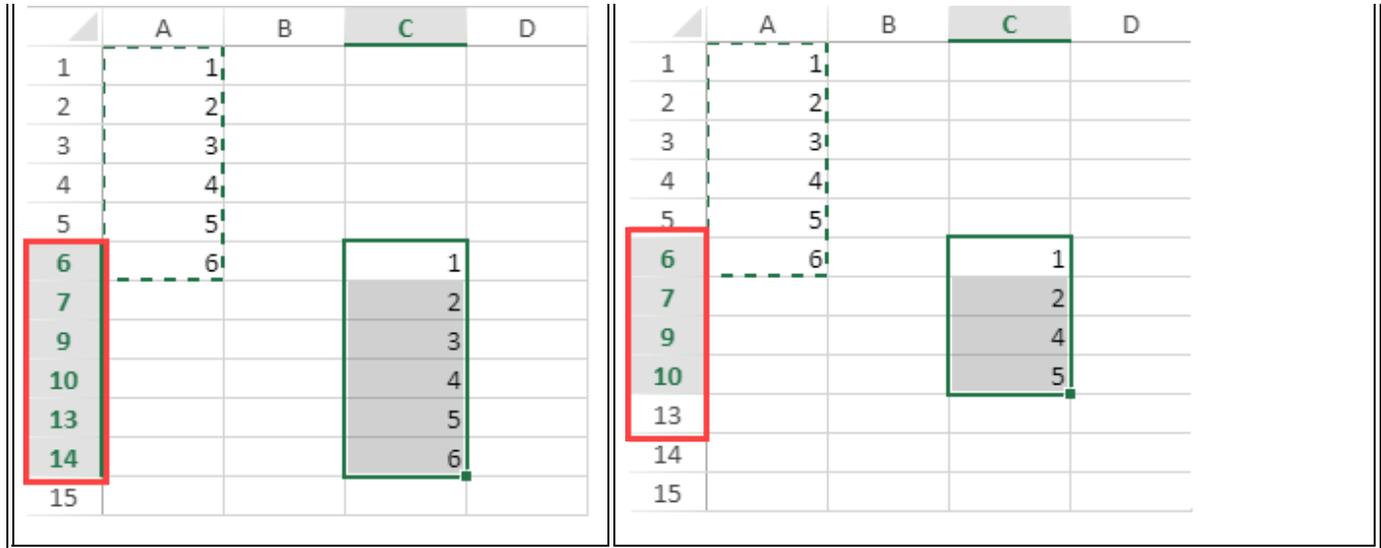
Disabling Pasting in Invisible Cells

You can set whether to paste the clipboard values in invisible cells, row, and columns by using the **PasteSkipInvisibleRange ('PasteSkipInvisibleRange Property' in the on-line documentation)** option from the **Features ('Features Class' in the on-line documentation)** class. It accepts a boolean value and is **False** by default.

This option only takes effect when **RichClipboard ('RichClipboard Property' in the on-line documentation)** is set to **True**.

When PasteSkipInvisibleRange = True
--

When PasteSkipInvisibleRange = False (default value)



The following example allows you to skip pasting the data in an invisible range.

C#

```
fpSpread1.LegacyBehaviors = LegacyBehaviors.None;
fpSpread1.Features.RichClipboard = true;
fpSpread1.ActiveSheet.Rows[7].Visible = false;
fpSpread1.ActiveSheet.Rows[10, 11].Visible = false;
fpSpread1.AsWorkbook().Features.PasteSkipInvisibleRange = true;
```

Visual Basic

```
fpSpread1.LegacyBehaviors = LegacyBehaviors.None
fpSpread1.Features.RichClipboard = true
fpSpread1.ActiveSheet.Rows(7).Visible = false
fpSpread1.ActiveSheet.Rows(10, 11).Visible = false
fpSpread1.AsWorkbook().Features.PasteSkipInvisibleRange = true
```

Performing the Clipboard Operations in Code

Various methods are provided in the **SheetView ('SheetView Class' in the on-line documentation)** class for Clipboard processes. You can run them when you want. These include:

- **ClipboardCopy ('ClipboardCopy Method' in the on-line documentation)**
- **ClipboardCopyShape ('ClipboardCopyShape Method' in the on-line documentation)**
- **ClipboardCut ('ClipboardCut Method' in the on-line documentation)**
- **ClipboardCutShape ('ClipboardCutShape Method' in the on-line documentation)**
- **ClipboardPaste ('ClipboardPaste Method' in the on-line documentation)**
- **ClipboardPasteShape ('ClipboardPasteShape Method' in the on-line documentation)**

Enabling Multi-Range Selection for Copy/Paste

The **RichClipboard** property in the **IFeatures** interface indicates whether you can copy/paste data across multi-ranges. This property can be implemented with flat style mode only.

	A	B	C	D	E	F	G	H	I
1	2		7						
2	4		5	6					
3	4		12						
4									
5									
6									
7									
8									
9									
10									
11									
12									
13									
14									
15									
16									

This multi-range selection for copy/paste option only takes effect when the **RichClipboard** property is set to true.

C#

```
// Flat style
fpSpread1.LegacyBehaviors &= ~LegacyBehaviors.Style;
// Setting RichClipboard property
fpSpread1.Features.RichClipboard= true;
```

Pasting Special Cell Content

Spread allows you to copy and paste cell contents into a worksheet. When you use the **Copy** and **Paste** options, all attributes are copied by default.

Instead of a simple direct paste, you can also use the **Paste Special** to paste the contents of the clipboard in a specific way. It is particularly useful when you want to control how data is transferred from the clipboard into your worksheet. For example, you can use **Paste Special** to paste only values without formulas, to transpose data, or to include conditional formatting during the paste process.

Using code

The following example code shows how to paste cell content and formatting using the **IRange.PasteSpecial** API. Note that the **RichClipboard** feature must be set to True to select various pasting operations.

C#

```
// Add cell content
fpSpread1.Features.RichClipboard = true;
fpSpread1.Features.EnhancedShapeEngine = true;
IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;
TestActiveSheet.SetValue(0, 0, new object[,] { { 1, 2, 3, null, 5 } });
```

```

var rangeA1C1 = TestActiveSheet.Cells["A1:C1"];
rangeA1C1.FormatConditions.AddIconSetCondition();
var cmt = TestActiveSheet.Cells["E2"].AddComment("visible comment");
cmt.Visible = true;
var thread = TestActiveSheet.Cells["D2"].AddCommentThreaded("thread");
thread.AddReply("reply1");
thread.AddReply("reply2");
var cellB2 = TestActiveSheet.Cells["B2"];
cellB2.Borders.LineStyle = GrapeCity.Spreadsheet.BorderLineStyle.Thick;
cellB2.Borders.Color =
GrapeCity.Spreadsheet.Color.FromKnownColor(GrapeCity.Core.KnownColor.Red);
// Copy and paste specific cell content
TestActiveSheet.Cells["A1:E2"].Copy(true);
TestActiveSheet.Cells["E8"].PasteSpecial(GrapeCity.Spreadsheet.PasteType.AllExceptBorders,
GrapeCity.Spreadsheet.PasteSpecialOperation.None, false, false);

```

VB

```

' Add cell content
fpSpread1.Features.RichClipboard = True
fpSpread1.Features.EnhancedShapeEngine = True
Dim TestActiveSheet As IWorksheet = fpSpread1.AsWorkbook().ActiveSheet
TestActiveSheet.SetValue(0, 0, New Object(,) {
{1, 2, 3, Nothing, 5}})
Dim rangeA1C1 = TestActiveSheet.Cells("A1:C1")
rangeA1C1.FormatConditions.AddIconSetCondition()
Dim cmt = TestActiveSheet.Cells("E2").AddComment("visible comment")
cmt.Visible = True
Dim thread = TestActiveSheet.Cells("D2").AddCommentThreaded("thread")
thread.AddReply("reply1")
thread.AddReply("reply2")
Dim cellB2 = TestActiveSheet.Cells("B2")
cellB2.Borders.LineStyle = GrapeCity.Spreadsheet.BorderLineStyle.Thick
cellB2.Borders.Color =
GrapeCity.Spreadsheet.Color.FromKnownColor(GrapeCity.Core.KnownColor.Red)
' Copy and paste specific cell content
TestActiveSheet.Cells("A1:E2").Copy(True)
TestActiveSheet.Cells("E8").PasteSpecial(GrapeCity.Spreadsheet.Paste
eType.AllExceptBorders, GrapeCity.Spreadsheet.PasteSpecialOperation.None, False, False)

```

At Runtime

The following steps show how to use the **Paste Special** functionality at runtime.

1. Run the code below to load a spread control with the ribbonBar having RichClipboard property as True.

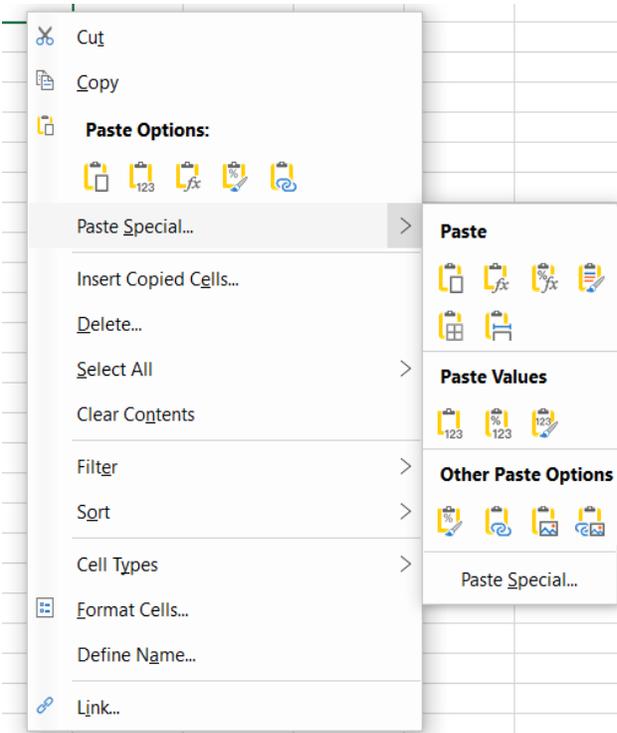
C#

```

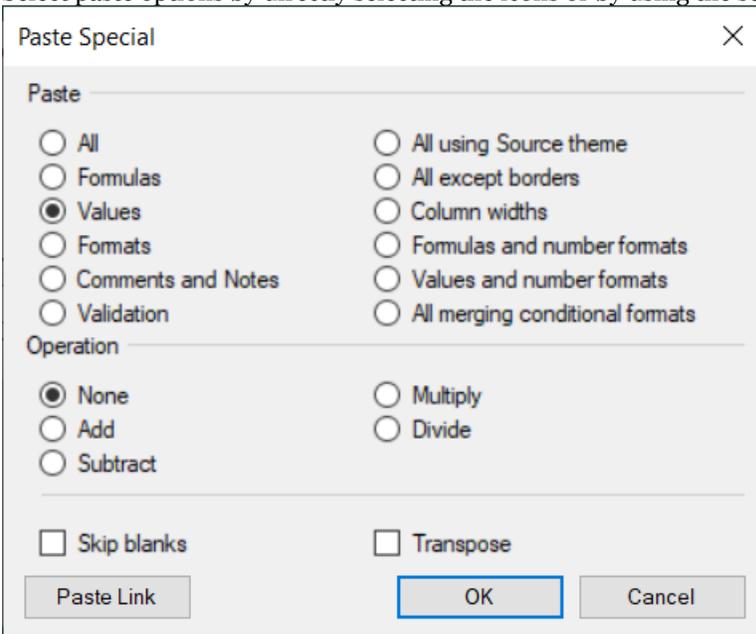
fpSpread1.Features.RichClipboard = true;
ribbonBar1.Attach(fpSpread1);

```

2. Select the cell range that contains the data or other attributes that you want to copy.
3. On the **Home** tab of the ribbonBar, click **Copy** or press **Ctrl + C** key.
4. Right-click on the cell where you want to paste the copied data and select the paste options.



5. Select paste options by directly selecting the icons or by using the settings from the **Paste Special** dialog.



 The options on the Paste menu will depend on the type of selected data. For example, if the target sheet has no conditional formatting, the option “Merge Conditional Formats” is not visible and is disabled on the **Paste Special** dialog.

Paste Menu Options

There are numerous options to paste copied cell data. The following are available paste menu options in Spread.

Select	To paste
Paste	All cell contents and formatting, including linked data.
Formulas	Only the formulas.
Formulas & Number Formatting	Only formulas and number formatting options.
Keep Source Formatting	All cell contents and formatting using the theme that was applied to the source data.
No Borders	All cell contents and formatting except cell borders.
Keep Source Column Widths	All cell contents and formatting and column widths.
Transpose	Reorients the content of copied cells when pasting. Data in rows is pasted into columns and vice versa.
Paste Values	Only the values as displayed in the cells.
Values Number Formatting	Only the values and number formatting.
Values Source Formatting	Only the values and number color and font size formatting.
Formatting	All cell formatting, including number and source formatting.
Paste Link	Link the pasted data to the original data. When you paste a link to the data that you copied, Excel enters an absolute reference to the copied cell or range of cells in the new location.
Paste as Picture	A copy of the image.
Linked Picture	A copy of the image with a link to the original cells (if you make any changes to the original cells those changes are reflected in the pasted image).
Column widths	Paste the width of one column or range of columns to another column or range of columns.
Merge Conditional Formatting	Combine conditional formatting from the copied cells with conditional formatting present in the paste area.

Paste Special Options

Spread provides the following advanced Paste Special options.

Select	To paste
All	All cell contents and formatting, including linked data.
Formulas	Only the formulas.
Values	Only the values as displayed in the cells.
Formats	Cell formatting.
Comments	Only comments attached to the cell.
Validation	Only data validation rules.
All using source theme	All cell contents and formatting using the theme that was applied to the source data.
All except borders	Cell contents and formatting, except cell borders.

Column widths	Width of one column or range of columns to another column or range of columns.
Formulas and number formats	Only formulas and number formatting.
Values and number formats	Only values and number formatting.
All merging conditional formats	All cell contents and formatting. Combine conditional formatting from the copied cells with conditional formatting present in the paste area.

Operation Options

The operation options mathematically combine values between the copy and paste areas. Note that Operations will be disabled when selected **Paste Type** is Formats, Comments and Notes, Validation.

Select	To paste
None	Paste the contents of the copy area without a mathematical operation.
Add	Add the values in the copy area to the values in the paste area.
Subtract	Subtract the values in the copy area from the values in the paste area.
Multiply	Multiply the values in the paste area by the values in the copy area.
Divide	Divide the values in the paste area by the values in the copy area.

Other options

The table below describes additional Paste Special options.

Select	To paste
Skip Blanks	Avoid replacing values or attributes in your paste area when blank cells occur in the copy area.
Transpose	Reorients the content of copied cells when pasting. Data in rows is pasted into columns and vice versa. Transpose is disabled if the copied range contains a sparkline.
Paste Link	If the data is a picture, links to the source picture. If the source picture is changed, this one will change too. Note that the Paste Link button is enabled only if the following conditions are fulfilled: <ul style="list-style-type: none"> • Paste Type is All or All except borders. • Operation is None. • Transpose and Skip Blanks are unchecked.

Limitations

- Paste Transpose, Column Widths, Operation, and Paste Link do not support the **PasteSkipInvisibleRange ('PasteSkipInvisibleRange Property' in the on-line documentation)** property of the Features class.
- The Paste menu on ribbonBar does not support Paste Special.

Rows and Columns

These tasks relate to setting the appearance of columns or rows in the sheet:

- **Customizing the Number of Rows or Columns**
- **Adding a Row or Column**
- **Removing a Row or Column**
- **Showing or Hiding a Row or Column**
- **Setting the Row Height or Column Width**
- **Setting Fixed (Frozen) Rows or Columns**
- **Moving Rows or Columns**
- **Creating Alternating Rows**
- **Setting up Preview Rows**
- **Input Data in Rows or Columns**
- **Finding Rows or Columns That Have Data**
- **Adding a Tag to a Row or Column**

When you work with rows and columns, you can work with the objects using the shortcuts in code (**Row**, **Rows**, **Column**, **Columns**, **AlternatingRow**, and **AlternatingRows** classes) or you can work directly with the model. Most developers who are not creating extensive customizations find it easier to work with the shortcut objects. For information on the underlying model responsible for rows and columns, refer to the **Understanding the Axis Model**.

You can edit properties of the Rows and Columns classes in the **Properties** window (in Spread Designer or in Visual Studio .NET). For more information on the **Cells, Columns, and Rows Editor** that is available from the **Properties** window, refer to the explanation of this editor in the Spread Designer Guide.

Similar to other grid products you might have used, Spread does not allow in-cell editing of the cells in the row and column headers.

For more information about the appearance of rows as a result of filtering, refer to **Setting the Appearance of Filtered Rows**.

Settings applied to a particular row or column override the settings that are set at the sheet level and settings applied at a cell level override the row or column settings. Refer to **Object Parentage**.

For more information, refer to the **Row ('Row Class' in the on-line documentation)**, **Rows ('Rows Class' in the on-line documentation)**, **AlternatingRow ('AlternatingRow Class' in the on-line documentation)**, **AlternatingRows ('AlternatingRows Class' in the on-line documentation)**, **Column ('Column Class' in the on-line documentation)**, and **Columns ('Columns Class' in the on-line documentation)** objects in the Assembly Reference.

Customizing the Number of Rows or Columns

When you create a sheet, it is automatically created with five hundred columns and five hundred rows. You can change the number to zero or up to two billion column and rows.

For more details, refer to the SheetView.**RowCount** (**'RowCount Property' in the on-line documentation**) property and SheetView.**ColumnCount** (**'ColumnCount Property' in the on-line documentation**) property.

You can also restrict the user from accessing various rows and columns.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.

3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet for which to set the number of columns or rows.
5. In the properties list, set the **ColumnCount** property to set the number of columns and the **RowCount** property to set the number of rows.
6. Click **OK** to close the editor.

Using a Shortcut

Set the **ColumnCount** or **RowCount** property for the Sheets shortcut object.

Example

This example code sets the first sheet to have 10 columns and 100 rows.

C#

```
fpSpread1.Sheets[0].ColumnCount = 10;  
fpSpread1.Sheets[0].RowCount = 100;
```

VB

```
fpSpread1.Sheets(0).ColumnCount = 10  
fpSpread1.Sheets(0).RowCount = 100
```

Method to set

Set the **ColumnCount** ('**ColumnCount Property**' in the on-line documentation) or **RowCount** ('**RowCount Property**' in the on-line documentation) property for a **SheetView** ('**SheetView Class**' in the on-line documentation) class.

Example

This example code sets the first sheet to have 10 columns and 100 rows.

C#

```
FarPoint.Win.Spread.SheetView Sheet0;  
Sheet0 = fpSpread1.Sheets[0];  
Sheet0.ColumnCount = 10;  
Sheet0.RowCount = 100;
```

VB

```
Dim Sheet0 As FarPoint.Win.Spread.SheetView  
Sheet0 = fpSpread1.Sheets(0)  
Sheet0.ColumnCount = 10  
Sheet0.RowCount = 100
```

Using the Spread Designer

1. Select the sheet tab for the sheet for which you want to set the number of columns or rows.
2. From the property list for the sheet, in the **Appearance** category, select **Columns** or **Rows** to expand the properties for the columns or rows in the sheet.
3. In the list of properties for the columns or rows, set the **Count** property.
4. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Adding a Row or Column

You can add one or more columns or rows to a sheet, and specify where the column or row is added. You can use the **AddRows ('AddRows Method' in the on-line documentation)** method and **AddColumns ('AddColumns Method' in the on-line documentation)** method of the **SheetView ('SheetView Class' in the on-line documentation)** class to add row and column respectively.

If you set the cell type for the column and a row is inserted, the new cells in the column use the cell type for the entire column. If you set the cell type for individual cells and a new row is inserted, then the new cells use the default cell type, and you would need to set the cell type for each of the new cells.

For more details refer to the **SheetView.AddRows ('AddRows Method' in the on-line documentation)** method or **SheetView.AddColumns ('AddColumns Method' in the on-line documentation)** method.

Using a Shortcut

- Call the **AddColumns** or **AddRows** method for the Sheets shortcut object.
- Set the *column* or *row* parameter to specify the column or row before which to add the columns or rows.
- Set the *count* parameter to specify the number of columns or rows to add.

Example

This example code adds two columns before 7th column.

C#

```
fpSpread1.Sheets[0].AddColumns(6, 2);
```

Visual Basic

```
fpSpread1.Sheets(0).AddColumns(6, 2)
```

Method to set

1. Use the **AddRows ('AddRows Method' in the on-line documentation)** or **AddColumns ('AddColumns Method' in the on-line documentation)** method for a **SheetView ('SheetView Class' in the on-line documentation)** object.
2. Set the *column* or *row* parameter to specify the column or row before which to add the columns or rows.
3. Set the *count* parameter to specify the number of columns or rows to add.

Example

This example code adds two columns before 7th column.

C#

```
FarPoint.Win.Spread.SheetView Sheet0;  
Sheet0 = fpSpread1.Sheets[0];  
Sheet0.AddColumns(6, 2);
```

Visual Basic

```
Dim Sheet0 As FarPoint.Win.Spread.SheetView  
Sheet0 = fpSpread1.Sheets(0)  
Sheet0.AddColumns(6, 2)
```

Using the Spread Designer

1. Select the sheet tab for the sheet for which you want to add a row or column.
2. Select a row above which you want to add a row or a column to the left of which you want to add a column.
3. Right-click on the row or column and choose **Insert**.
An additional row or column is added to the sheet.
4. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Removing a Row or Column

You can remove one or more columns or rows from a sheet. You can use the **RemoveRows ('RemoveRows Method' in the on-line documentation)** method and **RemoveColumns ('RemoveColumns Method' in the on-line documentation)** of the **class to remove row and column in the sheet. ('SheetView Class' in the on-line documentation)**

If you simply want to hide the row or column from the end user, but not remove it from the sheet, kindly refer to **Showing or Hiding a Row or Column**.

Using a Shortcut

- Call the **RemoveRows** or **RemoveColumns** method for the Sheets shortcut object.
- Set the *row* or *column* parameter to specify the row or column before which to remove the rows or columns.
- Set the *count* parameter to specify the number of rows or columns to remove.

Example

This example code removes two columns before 7th column.

C#

```
fpSpread1.Sheets[0].RemoveColumns(6,2);
```

Visual Basic

```
FpSpread1.Sheets(0).RemoveColumns(6,2)
```

Method to set

1. Call the **RemoveRows ('RemoveRows Method' in the on-line documentation)** or **RemoveColumns ('RemoveColumns Method' in the on-line documentation)** method for a **SheetView ('SheetView Class' in the on-line documentation)** object.
2. Set the *row* or *column* parameter to specify the row or column before which to remove the rows or columns.
3. Set the *count* parameter to specify the number of rows or columns to remove.

Example

This example code removes two columns before 7th column.

C#

```
FarPoint.Win.Spread.SheetView Sheet0;  
Sheet0 = fpSpread1.Sheets[0];  
Sheet0.RemoveColumns(6,2);
```

Visual Basic

```
Dim Sheet0 As FarPoint.Win.Spread.SheetView
Sheet0 = fpSpread1.Sheets(0)
Sheet0.RemoveColumns(6, 2)
```

Using the Spread Designer

1. Select the sheet tab for the sheet for which you want to remove a row or column.
2. Select the row(s) or column(s) to remove by selecting the header(s).
3. Right-click on the row or column and choose **Delete**.
4. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Showing or Hiding a Row or Column

You can choose to show or hide rows and columns in a sheet by following one of the methods below:

- Set the **Row.Visible** ('**Visible Property**' in the on-line documentation) property or **Column.Visible** ('**Visible Property**' in the on-line documentation) property. The properties accept boolean values and hide a row or column when set to false.
- Set the **SetColumnVisible** ('**SetColumnVisible Method**' in the on-line documentation) or **SetRowVisible** ('**SetRowVisible Method**' in the on-line documentation) methods of SheetView class to show or hide rows and columns.
- Set the row height or column width to zero by using the **RowHeight** ('**RowHeight Property**' in the on-line documentation) and **ColumnWidth** ('**ColumnWidth Property**' in the on-line documentation) properties.



You can also hide row headers and column headers. For more information, refer to **Showing or Hiding Headers**.

The following code sample demonstrates the use of SheetView methods as well as Rows and Columns properties to hide the first row and the second column.

C#

```
//Using SheetView methods to set visibility
fpSpread1.Sheets[0].SetColumnVisible(1, false);
fpSpread1.Sheets[0].SetRowVisible(0, false);
//Another option - use the Visible property.
fpSpread1.Sheets[0].Columns[1].Visible = false;
fpSpread1.Sheets[0].Rows[0].Visible = false;
//Another option - set row height or column width to zero.
GrapeCity.Spreadsheet.IWorksheet sheet1 = fpSpread1.AsWorkbook().ActiveSheet
sheet1.Columns[1].ColumnWidth = 0;
sheet1.Rows[0].RowHeight = 0;
```

VB

```
'Using SheetView methods to set visibility
fpSpread1.Sheets(0).SetColumnVisible(1, False)
fpSpread1.Sheets(0).SetRowVisible(0, False)
'Another option - use the Visible property.
fpSpread1.Sheets(0).Columns(1).Visible = False
fpSpread1.Sheets(0).Rows(0).Visible = False
'Another option - set row height or column width to zero.
```

```
GrapeCity.Spreadsheet.IWorksheet sheet1 = fpSpread1.AsWorkbook().ActiveSheet
sheet1.Columns(1).ColumnWidth = 0
sheet1.Rows(0).RowHeight = 0
```

 **Note:** To know more about showing or hiding rows and columns by setting row height and column width, refer to **Setting the Row Height or Column Width**.

By default, the width and height of hidden columns and rows are displayed by about 1 pixel when set to hidden. To eliminate this display, you need to set **ResizeZeroIndicator** (**'ResizeZeroIndicator Enumeration' in the on-line documentation**) to Default.

ResizeZeroIndicator = Default

4
5
7

ResizeZeroIndicator = Enhanced

4
5
7

When you hide a row or column, the size of the row or column is remembered by the Spread component. If you re-display the row or column, it is displayed at the size it was before it was hidden.

Conversely, you can also prevent the user to re-display the hidden rows or columns by setting the Row.**Resizable** (**'Resizable Property' in the on-line documentation**) or Column.**Resizable** (**'Resizable Property' in the on-line documentation**) properties to false.

If you want to determine if a row or column is presently visible to the user, that is, whether it appears in the viewport that is currently being displayed, you can use the [FpSpread class methods](#). For more information refer to **Customizing Viewports**.

Using the Properties Window

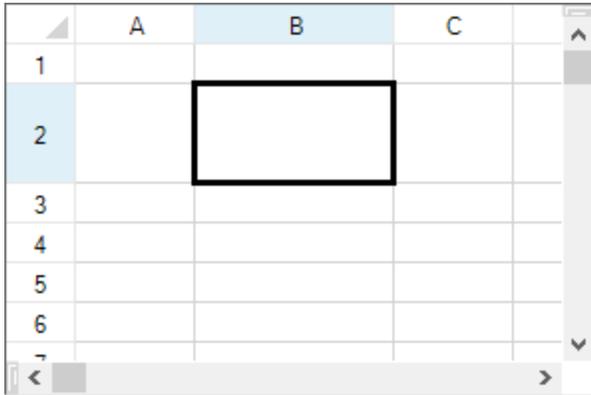
1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheet** collection.
3. Select the **Row** or **Column** collection.
4. Click on a row or column to select it and then select an option from the drop-down combo list for the **Visible** property.

Using the Spread Designer

1. Select the row or column by clicking on the header.
2. Select an option from the **Visible** drop-down combo list.
3. From the **File** menu, select **Save and Exit** to save the changes.

Setting the Row Height or Column Width

You can set the row height or column width as a specified number of pixels. Each sheet uses and lets you set a default size, making all rows or columns in the sheet the same size. You can override that setting by setting the value for individual rows or columns.



Users can change the row height or column width by dragging the header lines between rows or columns.

For more details refer to the **Column.Width ('Width Property' in the on-line documentation)** method or **Row.Height ('Height Property' in the on-line documentation)** method. For more information, kindly refer to "**Allowing the User to Resize Rows or Columns**".

Method to set

Get a collection of rows and columns in the **Rows ('Rows Property' in the on-line documentation)** and **Columns ('Columns Property' in the on-line documentation)** property of the **SheetView ('SheetView Class' in the on-line documentation)** class, and specify the target row and column by index. The row height is set by the **Height ('Height Property' in the on-line documentation)** property of the **Row ('Row Class' in the on-line documentation)** class, and the column width is set by the **Width ('Width Property' in the on-line documentation)** property of the **Column ('Column Class' in the on-line documentation)** class.

Example

This example code sets the second column's width to 100 pixels.

C#

```
// Set second column width to 100.
fpSpread1.Sheets[0].Columns[1].Width = 100;
```

VB

```
' Set second column width to 100.
fpSpread1.Sheets(0).Columns(1).Width = 100
```

Note: When the **Column.Width ('Width Property' in the on-line documentation)** or **Row.Height ('Height Property' in the on-line documentation)** methods are set to zero, the row height and column width are set to zero and they appear as hidden.

You can also set the row height and column width to zero to hide the rows and columns by using **RowHeight ('RowHeight Property' in the on-line documentation)** and **ColumnWidth ('ColumnWidth Property' in the on-line documentation)** properties. However, in this case, the originally set row height or column width is retained when they are re-displayed. The below example code explains this scenario. You can also use the Axis model to change the width and height which is restored on being re-displayed.

C#

```
GrapeCity.Spreadsheet.IWorksheet sheet1 = fpSpread1.AsWorkbook().ActiveSheet;
sheet1.Rows[5].RowHeight = 50; // Initial row height is set to 50
```

```

sheet1.Rows[5].RowHeight = 0;    // The row is now hidden, the row height is still 50
internally
sheet1.Rows[5].Hidden = false;  // The row is visible and the row height is 50
sheet1.Rows[5].Hidden = true;   // The row height is still 50 internally
fpSpread1.ActiveSheet.Models.RowAxis.SetSize(5, 100); // Use AxisModel to change row
height
sheet1.Rows[5].Hidden = false;  // The row height is 100

```

VB

```

GrapeCity.Spreadsheet.IWorksheet sheet1 = fpSpread1.AsWorkbook().ActiveSheet
sheet1.Rows(5).RowHeight = 50 'Initial row height is set to 50
sheet1.Rows(5).RowHeight = 0  'The row is now hidden, the row height is still 50
internally
sheet1.Rows(5).Hidden = false 'The row is visible and the row height is 50
sheet1.Rows(5).Hidden = true  'The row height is still 50 internally
fpSpread1.ActiveSheet.Models.RowAxis.SetSize(5, 100) 'Use AxisModel to change row
height
sheet1.Rows(5).Hidden = false 'The row height is 100

```

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet for which to set the column width for a column.
5. In the properties list, set the **Columns** property (or **Row** property) and then click the button to display the **Cell, Column, and Row Editor**.
6. Click the column heading of the column for which you want to set the width (or row for the height).
7. In the properties list, set the **Width** property (**Height** property for the row).
8. Click **OK** to close the **Cell, Column, and Row Editor**.
9. Click **OK** to close the **SheetView Collection Editor**.

Using the Spread Designer

1. To set the default column width,
 - a. Select the sheet tab for the sheet for which you want to set the default column width.
 - b. In the property list, in the **Appearance** category, select **Columns** to expand the list of properties for the columns.
 - c. In the list of properties for columns, select **Default** to expand the list of default column settings.
 - d. In the list of default settings, change the **Width** setting.
 - e. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.
2. To set a specific column width,
 - a. Select the column for which you want to change the width.
 - b. In the properties list for that column, change the **Width** property.
 - c. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Resizing the Row or Column to Fit the Data

Spread for WinForms also allows you to resize the column width or row height based on the length or breadth of data in the cells in that column or row. The size of the row or column with the largest data is referred to as the preferred size.

The methods that make use of the preferred size are:

- **Row ('Row Class' in the on-line documentation) class, GetPreferredHeight ('GetPreferredHeight Method' in the on-line documentation) method**
- **Column ('Column Class' in the on-line documentation) class, GetPreferredWidth ('GetPreferredWidth Method' in the on-line documentation) method**
- **SheetView ('SheetView Class' in the on-line documentation) class, GetPreferredRowHeight ('GetPreferredRowHeight Method' in the on-line documentation) method**
- **SheetView ('SheetView Class' in the on-line documentation) class, GetPreferredColumnWidth ('GetPreferredColumnWidth Method' in the on-line documentation) method**
- **SheetView ('SheetView Class' in the on-line documentation) class, GetPreferredCellSize ('GetPreferredCellSize Method' in the on-line documentation) method**

The **GetPreferredHeight ('GetPreferredHeight Method' in the on-line documentation)** method of the **Row ('Row Class' in the on-line documentation) class** and **GetPreferredWidth ('GetPreferredWidth Method' in the on-line documentation)** method of the **Column ('Column Class' in the on-line documentation) class** always include the header cells. The overloaded **GetPreferredColumnWidth ('GetPreferredColumnWidth Method' in the on-line documentation)** method of the **SheetView ('SheetView Class' in the on-line documentation) class** has one overload that always includes the header cells while another overload allows you to choose whether to include or exclude header cells. In the following code, `width1` and `width2` include the header cells but `width3` excludes the header cells.

C#

```
float width1 = fpspread.Sheets[0].Columns[0].GetPreferredWidth();
float width2 = fpspread.Sheets[0].GetPreferredColumnWidth(0);
float width3 = fpspread.Sheets[0].GetPreferredColumnWidth(0, true);
```

For information on setting the cell size based on the size of the data, refer to **Resizing a Cell to Fit the Data**.

For information on allowing the user to resize the data, refer to **Allowing the User to Resize Rows or Columns**.



- By default, the `WordWrap` of the header cell is set to `true`. If the header text is longer than the cell text, you should disable the `WordWrap` like the following example code to make the maximum length (the width of the widest text) match the header text.

C#

```
//Disable WordWrap for all column header labels
FarPoint.Win.Spread.CellType.ColumnHeaderRenderer ch = new
FarPoint.Win.Spread.CellType.EnhancedColumnHeaderRenderer();
ch.WordWrap = false;
fpSpread1.ActiveSheet.ColumnHeader.DefaultStyle.Renderer = ch;
```

Visual Basic

```
'Disable WordWrap for all column header labels
Dim ch As New FarPoint.Win.Spread.CellType.EnhancedColumnHeaderRenderer
ch.WordWrap = False
FpSpread1.ActiveSheet.ColumnHeader.DefaultStyle.Renderer = ch
```

- The column size automatically adjusted by double-clicking the border of column header, or the preferred size fetched by calling the `GetPreferredColumnWidth` method, etc. slightly differs depending on the setting font (font name, size, style) of the target cell.
- In case of rich text type cell, auto-size is enabled, and when using multiple fonts, the column width is

shifted by several pixels. The automatically adjusted column widths also differ depending on the used font, size, and style (bold, italic, and underline). (The text width fetched by the `GetPreferredColumnWidth` method is also different.)

- The height of the row fetched by calling the `GetPreferredRowHeight` method in a rich text type cell differs depending on the used font, size, and style (bold, italic, underline). Also, when the **WordWrap ('WordWrap Property' in the on-line documentation)** property or **Multiline ('Multiline Property' in the on-line documentation)** property is set to `True`, the required row height cannot be fetched if different font information is mixed in the same cell, so in this case, you need to make fine adjustments.

Example

This example sets the row height and column width.

C#

```
FarPoint.Win.Spread.Row row;
FarPoint.Win.Spread.Column col;
float sizerow;
float sizercol;
row = fpSpread1.ActiveSheet.Rows[0];
col = fpSpread1.ActiveSheet.Columns[0];
fpSpread1.ActiveSheet.Cells[0, 0].Text = "This text is used to determine the height and width.";
sizerow = row.GetPreferredHeight();
sizercol = col.GetPreferredWidth();
row.Height = sizerow;
col.Width = sizercol;
```

VB

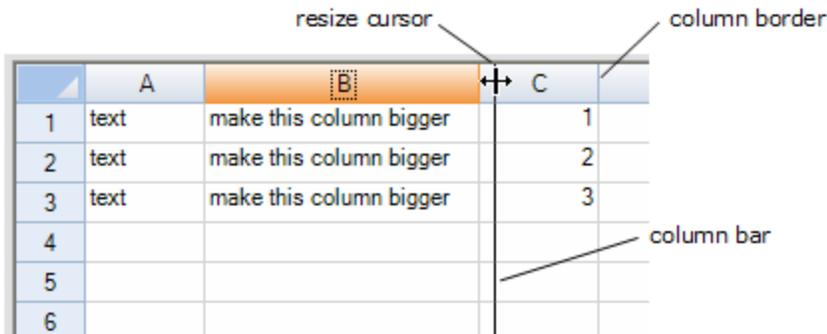
```
Dim row As FarPoint.Win.Spread.Row
Dim col As FarPoint.Win.Spread.Column
Dim sizerow As Single
Dim sizercol As Single
row = fpSpread1.ActiveSheet.Rows(0)
col = fpSpread1.ActiveSheet.Columns(0)
fpSpread1.ActiveSheet.Cells(0, 0).Text = "This text is used to determine the height and width."
sizerow = row.GetPreferredHeight()
sizercol = col.GetPreferredWidth()
row.Height = sizerow
col.Width = sizercol
```

Allowing the User to Resize Rows or Columns

You can allow the user to adjust the size of a row or column in the sheet. Set the **Resizable ('Resizable Property' in the on-line documentation)** property for the **Row ('Row Class' in the on-line documentation)** class to allow the user to resize rows and the **Resizable ('Resizable Property' in the on-line documentation)** property for the **Column ('Column Class' in the on-line documentation)** class to allow the user to resize columns. The user can also double-click on the border between the column headers to resize the column to match the width of the header text.

For users to resize rows or columns, they left click on the edge of the header of the row or column to resize and drag the side of the header and release the mouse at the desired size. While the left mouse button is down, a bar is displayed along with the resize pointer as shown in the following figure. Be sure to click on the right edge of the column and

bottom edge of the row.



Automatic Adjustment by Double Click

Double-clicking on the row or column edge resizes the row or column to fit the tallest or widest content of that row or column (preferred height or width). Wrapped text in the column is ignored when double-clicking on a column divider. Unwrapped text is ignored when double-clicking on a row divider. You can specify the automatic fit behavior with the **AutoFitColumnOptions** ('**AutoFitColumnOptions Property**' in the **on-line documentation**) or **AutoFitRowOptions** ('**AutoFitRowOptions Property**' in the **on-line documentation**) property.

Header Area Resize

By default, user resizing of rows or columns is allowed for rows and columns in the data area and not allowed for the header area. In code, you can resize row and column headers, not just data area rows and columns. You can override the default behavior using the **Resizable** ('**Resizable Property**' in the **on-line documentation**) property and prevent the user from resizing.

The following code turns on resizing for a single column in the row header:

```
spread.Sheets[0].RowHeader.Columns[0].Resizable = true;
```

The following code turns on resizing for all columns in the row header:

```
spread.Sheets[0].RowHeader.Columns.Default.Resizable = true;
```

You can determine if a row or column can be resized by the user with these methods in the SheetView class:

- **GetColumnSizeable** ('**GetColumnSizeable Method**' in the **on-line documentation**)
- **SetColumnSizeable** ('**SetColumnSizeable Method**' in the **on-line documentation**)
- **GetRowSizeable** ('**GetRowSizeable Method**' in the **on-line documentation**)
- **SetRowSizeable** ('**SetRowSizeable Method**' in the **on-line documentation**)

To resize the rows or column based on the size of the data, refer to **Resizing the Row or Column to Fit the Data**.

Using the Properties Window

1. At design time, in the **Properties** window, select **Sheet**.
2. In the **Properties** window, select **Column** or select **Row**, and click on the button for the **Cell, Column, and Row Editor**.
3. In the Editor, select a column or a row.
4. Select the **Resizable** property.
5. Click the drop-down arrow to display the choices and select the value (True or False). Repeat this for each property.

Method to set

Set the **Resizable** ('**Resizable Property**' in the on-line documentation) property for the row or **Resizable** ('**Resizable Property**' in the on-line documentation) property for the column.

Example

The following example sets the sheet to allow the first row and first column to be resizable.

C#

```
fpSpread1.Sheets[0].Columns[0].Resizable = true;  
fpSpread1.Sheets[0].Rows[0].Resizable = true;
```

VB

```
fpSpread1.Sheets(0).Columns(0).Resizable = True  
fpSpread1.Sheets(0).Rows(0).Resizable = True
```

Using the Spread Designer

1. Select **Sheet** from the drop-down combo list located on the top right side of the Designer.
2. From the **Appearance** category, select **Column** or select **Row**, and click on the button for the **Cell, Column, and Row Editor**.
3. In the Editor, select a column or a row.
4. Select the **Resizable** property.
5. Select the value from the drop-down list (either True or False).
6. From the **File** menu, select **Save and Exit** to save the changes.

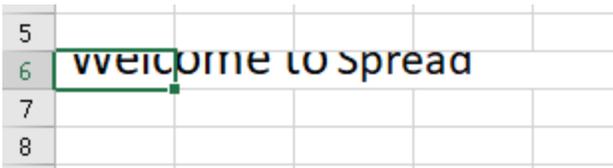
Using Auto Row Height

Spread for WinForms supports automatic adjustment of row height based on the data present in the row.

The auto row height feature works only when the following conditions are met:

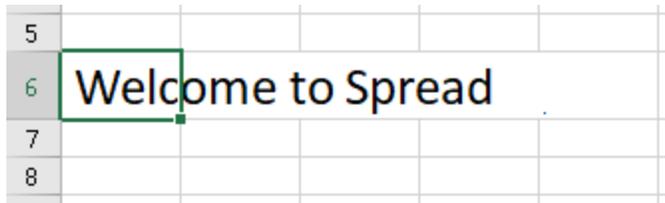
- The LegacyBehaviors property should not include AutoRowHeight.
- The custom height for the row containing the cell should not be set.
- The cell shouldn't belong to a span.

When the auto row height feature is disabled, the cell containing text with larger font size will appear clipped as shown in the example screenshot shared below.



5					
6	welcome to spread				
7					
8					

When the auto row height feature is enabled, the row will automatically adjust its height in accordance to the font size of the text as shown in the example screenshot shared below.



Users can enable the auto row height feature in the workbook in order to allow the Spread component to automatically calculate row height. The row height will be calculated automatically whenever the cut, copy, paste, move, drag drop or drag fill operations are executed on the spreadsheet, the value of a cell is changed directly or when the edit mode of the cell is exited.

In order to enable the auto row height feature, refer to the following code.

C#

```
// Enable all new behaviors by default including auto row height
fpSpread1 = new FarPoint.Win.Spread.FpSpread(FarPoint.Win.Spread.LegacyBehaviors.None);

// Enable the AutoRowHeight feature only
fpSpread1 = new FarPoint.Win.Spread.FpSpread(FarPoint.Win.Spread.LegacyBehaviors.All &
~FarPoint.Win.Spread.LegacyBehaviors.AutoRowHeight);
```

VB

```
'Enable all new behaviors by default including auto row height
fpSpread1 = New FarPoint.Win.Spread.FpSpread(FarPoint.Win.Spread.LegacyBehaviors.None)

'Enable the Auto Row Height feature only
fpSpread1 = New FarPoint.Win.Spread.FpSpread(FarPoint.Win.Spread.LegacyBehaviors.All &
~FarPoint.Win.Spread.LegacyBehaviors.AutoRowHeight)
```

In order to disable the auto row height feature, users need to set the **LegacyBehaviors** property to **AutoRowHeight** as shown in the code snippet shared below.

C#

```
// Disable all new behaviors by default including auto row height.
fpSpread1 = new FarPoint.Win.Spread.FpSpread(FarPoint.Win.Spread.LegacyBehaviors.All);

// Disable the Auto Row Height feature
fpSpread1 = new FarPoint.Win.Spread.FpSpread(FarPoint.Win.Spread.LegacyBehaviors.None |
FarPoint.Win.Spread.LegacyBehaviors.AutoRowHeight);
```

VB

```
'Disable all new behaviors by default including auto row height.
fpSpread1 = New FarPoint.Win.Spread.FpSpread(FarPoint.Win.Spread.LegacyBehaviors.All)

'Disable the Auto Row Height feature
fpSpread1 = New FarPoint.Win.Spread.FpSpread(FarPoint.Win.Spread.LegacyBehaviors.None |
FarPoint.Win.Spread.LegacyBehaviors.AutoRowHeight)
```

Setting Fixed (Frozen) Rows or Columns

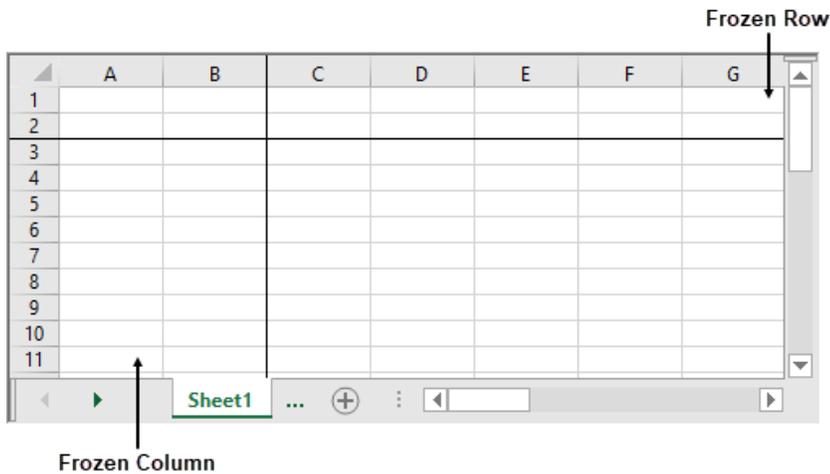
You can freeze rows or columns or both in a sheet to make them unscrollable.

 **Note:** Frozen rows or columns are not scrollable at run time but they are scrollable during design time.

Set Frozen Rows or Columns

The frozen top rows are called leading rows and the frozen left-most columns are called leading columns. The frozen leading rows and columns stay at the top and left of the view regardless of the scrolling.

You can set the number of frozen rows or columns by using the **FrozenRowCount** (**'FrozenRowCount Property' in the on-line documentation**) or **FrozenColumnCount** (**'FrozenColumnCount Property' in the on-line documentation**) properties.



C#

```
// freeze rows and columns
FpSpread1.Sheets[0].FrozenColumnCount = 2;
FpSpread1.Sheets[0].FrozenRowCount = 2;
```

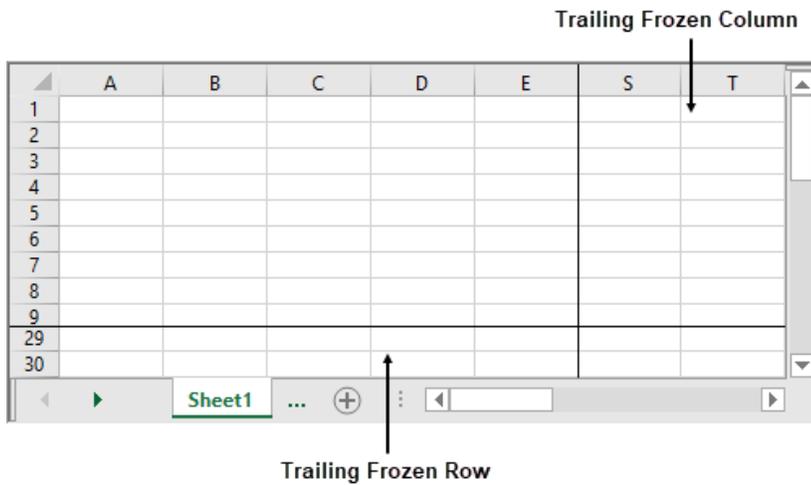
Visual Basic

```
'freeze rows And columns
FpSpread1.Sheets(0).FrozenColumnCount = 2
FpSpread1.Sheets(0).FrozenRowCount = 2
```

Set Trailing Frozen Rows or Columns

The frozen trailing rows and columns stay at the bottom and right of the view regardless of the scrolling.

You can set the number of frozen trailing bottom rows or trailing right-most columns by using **FrozenTrailingColumnCount** (**'FrozenTrailingColumnCount Property' in the on-line documentation**) or **FrozenTrailingRowCount** (**'FrozenTrailingRowCount Property' in the on-line documentation**) properties.



When the height of data rows and frozen trailing rows is less than that of the viewport area, you can choose whether to display the blank space between the data rows and frozen trailing rows or not. The **FrozenTrailingStickToEdge** ('FrozenTrailingStickToEdge Property' in the on-line documentation) option can be used to control this behavior and accepts RowCol ('RowCol Enumeration' in the on-line documentation) enumeration values such as Both (default), Rows, Columns, None.

When FrozenTrailingStickToEdge = Row						When FrozenTrailingStickToEdge = None					
17	I Love You Phi	Comedy	Independence	57	1.34	19	Jane Eyre	Romance	Universal	77	20.12
18	It's Complicated	Comedy	Universal	63	2.64	20	Just Wright	Comedy	Fox	58	1.79
19	Jane Eyre	Romance	Universal	77		21	Killers	Action	Lionsgate	45	1.24
20	Just Wright	Comedy	Fox	58	1.79	22	Knocked Up	Comedy	Universal	83	6.63
						23	Leap Year	Comedy	Universal	49	1.71
21	Killers	Action	Lionsgate	45	1.24	24	Letters to Juliet	Comedy	Summit	62	2.639333
22	Knocked Up	Comedy	Universal	83	6.63	25	License to Wed	Comedy	Warner Br	55	1.98
23	Leap Year	Comedy	Universal	49	1.71						
24	Letters to Juliet	Comedy	Summit	62	2.639333						
25	License to Wed	Comedy	Warner Br	55	1.98						

C#

```
fpSpread1.ActiveSheet.Rows.Count = 25;
fpSpread1.ActiveSheet.Columns.Count = 20;
fpSpread1.Sheets[0].FrozenTrailingRowCount = 5;
fpSpread1.ActiveSheet.FrozenTrailingStickToEdge = FarPoint.Win.Spread.RowCol.Rows;
```

Visual Basic

```
FpSpread1.ActiveSheet.Rows.Count = 25
FpSpread1.ActiveSheet.Columns.Count = 20
FpSpread1.Sheets(0).FrozenTrailingRowCount = 5
FpSpread1.ActiveSheet.FrozenTrailingStickToEdge = FarPoint.Win.Spread.RowCol.Rows
```

When using the print option, trailing frozen rows and columns are not printed repeatedly at the bottom and right of every page, but print only once as the last row and column.

Leading frozen rows and columns can be repeatedly printed at the top and left of every page. For more information about repeating rows and columns, refer to **Repeating Rows or Columns on Printed Pages**.

Set Frozen Line Color

You can specify the color of the line displayed between the frozen and non-frozen areas of the worksheet with the **FrozenLineColor** ('FrozenLineColor Property' in the on-line documentation) property in the **WorksheetOptions** ('WorksheetOptions Class' in the on-line documentation) class. By default, no color is set to show the frozen line.

The preferred frozen line color can be chosen by using [Color methods](#).

	A	B	C	D	E	F	S	T
1								
2								
3								
4								
5								
6								
7								
9								
10								
13								
14								
29								
30								

Frozen Line Color
RGB (0,0,255)

C#

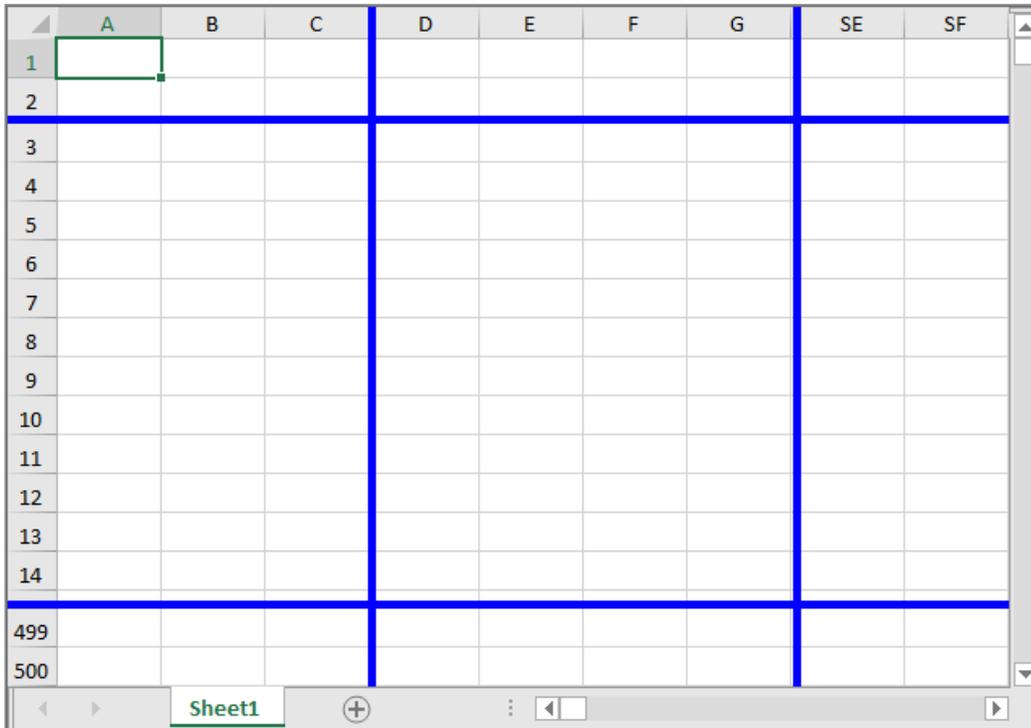
```
// freeze rows and columns
fpSpread1.Sheets[0].FrozenColumnCount = 2;
fpSpread1.Sheets[0].FrozenRowCount = 2;
fpSpread1.Sheets[0].FrozenTrailingColumnCount = 2;
fpSpread1.Sheets[0].FrozenTrailingRowCount = 2;
// set frozen line color
fpSpread1.AsWorkbook().Worksheets[0].Options.FrozenLineColor = GrapeCity.Spreadsheet.Color.FromArgb(0, 0, 255);
```

Visual Basic

```
'freeze rows And columns
FpSpread1.Sheets(0).FrozenColumnCount = 2
FpSpread1.Sheets(0).FrozenRowCount = 2
FpSpread1.Sheets(0).FrozenTrailingColumnCount = 2
FpSpread1.Sheets(0).FrozenTrailingRowCount = 2
'set frozen line color
FpSpread1.AsWorkbook().Worksheets(0).Options.FrozenLineColor = GrapeCity.Spreadsheet.Color.FromArgb(0, 0, 255)
```

Set Frozen Line Width

You can specify or change the width of the displayed line between the frozen and non-frozen areas of the worksheet using **FrozenLineThickness** property from the **WorksheetOptions** class.

**C#**

```
fpSpread1.ActiveSheet.FrozenColumnCount = 3;
fpSpread1.ActiveSheet.AsWorksheet().Options.FrozenLineThickness = 5;
```

VB

```
fpSpread1.ActiveSheet.FrozenColumnCount = 3
fpSpread1.ActiveSheet.AsWorksheet().Options.FrozenLineThickness = 5
```

Using the Properties Window

1. At design time, in the **Properties** window, select the Sheet.
2. In the **SheetView Collection Editor (Appearance section)**, set the **FrozenRowCount**, **FrozenColumnCount**, **FrozenTrailingRowCount**, **FrozenTrailingColumnCount**, or the **FrozenTrailingStickToEdge** property.

Using the Spread Designer

1. Select **Sheet** from the drop-down combo list located on the top right side of the Designer.
2. From the **SheetView Collection Editor**, set the **FrozenColumnCount**, **FrozenRowCount**, **FrozenTrailingColumnCount**, **FrozenTrailingRowCount**, **FrozenTrailingStickToEdge**.
3. From the **File** menu, select **Save and Exit** to save the changes.

Moving Rows or Columns

You can allow the user to drag and move rows or columns. Set the **AllowRowMove** ('**AllowRowMove Property**' in the on-line documentation) property to allow the user to move rows and the **AllowColumnMove** ('**AllowColumnMove Property**' in the on-line documentation) property to allow the user to move columns. If you wish to allow the user to move multiple rows or columns, also set the **AllowRowMoveMultiple** ('**AllowRowMoveMultiple Property**' in the on-line documentation) or **AllowColumnMoveMultiple** ('**AllowColumnMoveMultiple Property**' in the on-line documentation) property.

You can hide the drag band while moving the row or column with the **ShowDragBandOnMoving** ('**ShowDragBandOnMoving Property**' in the on-line documentation) property.

For the user to move rows or columns, they left click on the header of the row or column to move and drag the header back or forth over the header area and release the mouse over the header of the desired destination (for multiple rows or columns, select them first). The row or column that is being moved is shown in a transparent clone attached to the pointer, as shown in this figure, where the fourth column is being moved to the left.

	ID	CompanyName	ContactTitle	ContactTitle
1	1	Alfreds Futterkiste	Maria Anders	Sales Representative
2	2	Ana Trujillo Emparedados y heladerías	Ana Trujillo	Owner
3	3	Antonio Moreno Taquería	Antonio Moreno	Owner
4	4	Around the Horn	Thomas Hardy	Sales Representative
5	5	Berglunds snabbköp	Christina Berglund	Order Administrator
6	6	Blauer See Delikatessen	Hanna Christodoulou	Sales Representative
7	7	Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager
8	8	Bólido Comidas preparadas	Martín Chocón	Owner
9	9	Bon app'	Lauree Coates	Owner
10	10	Bottom-Dollar Markets	Elizabeth Hench	Accounting Manager
11	11	B's Beverages	Victor Jones	Sales Representative
12	12	Cactus Comidas para llevar	Patricio Simons	Sales Agent
13	13	Centro comercial Moctezuma	Francoise Guillot	Marketing Manager
14	14	Chop-suey Chinese	Yang Weng	Owner

Moving the Row or Column in Code

To relocate a row, use the `SheetView.SheetView` ('[SheetView Class](#)' in the on-line documentation) method; to remove multiple rows at a time, use the `SheetView.RemoveRows` ('[RemoveRows Method](#)' in the on-line documentation) method. To relocate a column, use the `SheetView.MoveColumn` ('[MoveColumn Method](#)' in the on-line documentation) method; to remove multiple columns at a time, use the `SheetView.RemoveColumns` ('[RemoveColumns Method](#)' in the on-line documentation) method. Alternatively, you can use the `Move` ('[Move Method](#)' in the on-line documentation) method of the `DefaultSheetAxisModel` ('[DefaultSheetAxisModel Class](#)' in the on-line documentation) class.

You can use the `GetColumnFromTag` ('[GetColumnFromTag Method](#)' in the on-line documentation) method to find columns based on their `Tag` ('[Tag Property](#)' in the on-line documentation) property, which you can set programmatically to whatever you want. You can do the same with the `Tag` ('[Tag Property](#)' in the on-line documentation) method for rows.

You can also delete multiple rows with the `Remove` ('[Remove Method](#)' in the on-line documentation) method of the `Rows` ('[Rows Class](#)' in the on-line documentation) class which represents a range of rows. Similarly, you can also remove multiple columns with the `Remove` ('[Remove Method](#)' in the on-line documentation) method of the `Columns` ('[Columns Class](#)' in the on-line documentation) class. For example,

C#

```
fpSpread1.Sheets[0].Columns[1, 5].Remove();
```

Visual Basic

```
FpSpread1.Sheets(0).Columns(1, 5).Remove()
```

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.

2. In the **Misc** category, select the **AllowRowMove** or the **AllowColumnMove** property.
3. Click the drop-down arrow to display the choices and select the value (True or False). Repeat this for each property.

Method to set

Set the **AllowRowMove** ('**AllowRowMove Property**' in the on-line documentation) or **AllowColumnMove** ('**AllowColumnMove Property**' in the on-line documentation) property for the **FpSpread** ('**FpSpread Class**' in the on-line documentation) component.

Example

This example allows the user to move columns or rows.

C#

```
fpSpread1.AllowRowMove = true;  
fpSpread1.AllowColumnMove = true;
```

VB

```
fpSpread1.AllowRowMove = True  
fpSpread1.AllowColumnMove = True
```

Using the Spread Designer

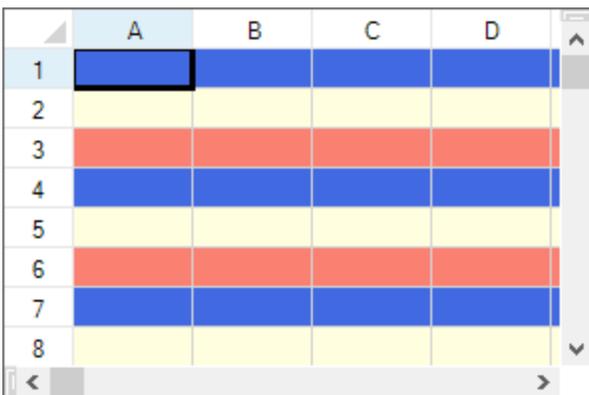
1. Select **Spread** from the drop-down combo list located on the top right side of the Designer.
2. From the **Misc** section, select **True** or **False** for **AllowRowMove** or **AllowColumnMove**.
3. From the **File** menu, select **Save and Exit** to save the changes.

Creating Alternating Rows

You might want to set up your sheet so that alternating rows have a different appearance. For example, in a ledger, alternating rows often have a green background. In Spread, you can set up multiple alternating row appearances, which are applied in sequence, starting with the first row.

Set up the alternating rows using an index into the alternating row appearances. It might help to think of the default row appearance as the first alternating row style (or style zero, because the index is zero-based). Set the other alternating row appearances to subsequent indexes.

The figure here shows the results for the following example code for setting up alternating rows for every three rows.



	A	B	C	D
1	Blue	Blue	Blue	Blue
2	Yellow	Yellow	Yellow	Yellow
3	Red	Red	Red	Red
4	Blue	Blue	Blue	Blue
5	Yellow	Yellow	Yellow	Yellow
6	Red	Red	Red	Red
7	Blue	Blue	Blue	Blue
8	Yellow	Yellow	Yellow	Yellow

For more details, refer to the **AlternatingRow** ('**AlternatingRow Class**' in the on-line documentation) class.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. Select the sheet for which you want to create alternating rows from the collection list.
5. Select the **AlternatingRows** property from the property list for that sheet.
6. If you want to add additional alternating rows patterns, set the **Count** property to the number of patterns you want.
7. Click the **AlternatingRows** property button to display the **AlternatingRow Collection Editor**.
8. Select alternating row pattern for which to set properties.
9. Set properties for the selected pattern using the property list.
10. Click **OK** to close the **AlternatingRow Collection Editor**.
11. Click **OK** to close the **SheetView Collection Editor**.

Method to set

Set using the index to the **AlternatingRows** property of **SheetView** class. The default row style is the first style (with index "0") of the appearance of every other row. When applying the default row style to the first row, set the index after "1" to other row style.

Example

This example code creates a sheet that has three different appearance settings for rows. The first row uses the default appearance. The second row has a light blue background with navy text, and the third row has a light yellow background with navy text. This pattern repeats for all subsequent rows.

C#

```
fpSpread1.Sheets[0].AlternatingRows.Count = 3;
fpSpread1.Sheets[0].AlternatingRows[0].BackColor = Color.RoyalBlue;
fpSpread1.Sheets[0].AlternatingRows[0].ForeColor = Color.Navy;
fpSpread1.Sheets[0].AlternatingRows[1].BackColor = Color.LightYellow;
fpSpread1.Sheets[0].AlternatingRows[1].ForeColor = Color.Navy;
fpSpread1.Sheets[0].AlternatingRows[2].BackColor = Color.Salmon;
fpSpread1.Sheets[0].AlternatingRows[2].ForeColor = Color.Navy;
```

VB

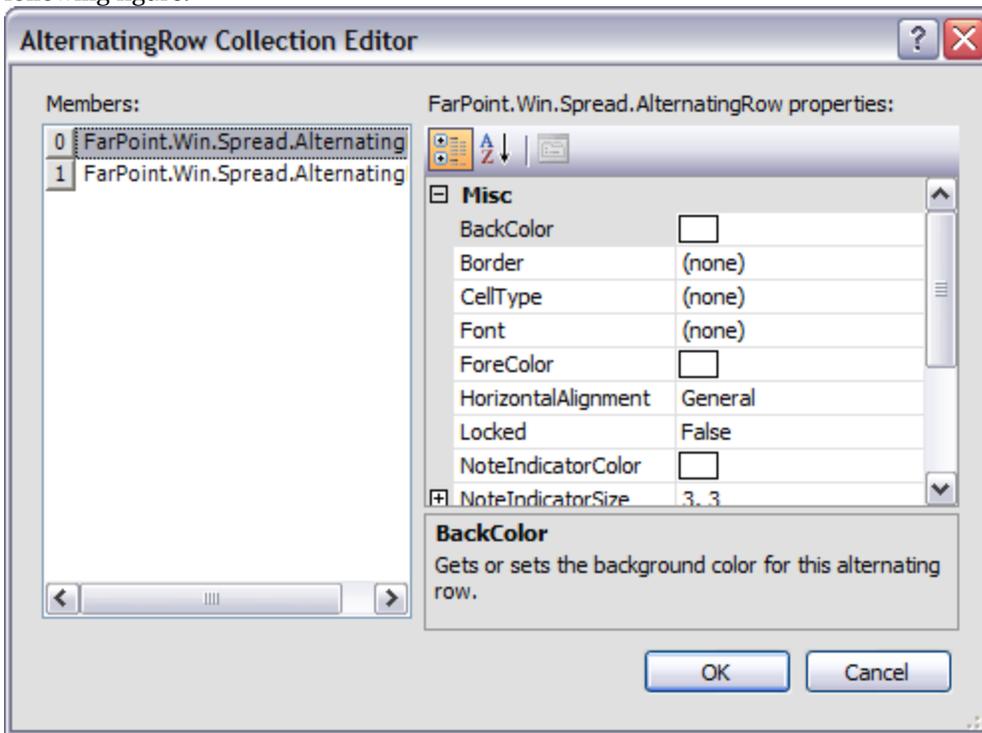
```
fpSpread1.Sheets(0).AlternatingRows.Count = 3
fpSpread1.Sheets(0).AlternatingRows(0).BackColor = Color.RoyalBlue
fpSpread1.Sheets(0).AlternatingRows(0).ForeColor = Color.Navy
fpSpread1.Sheets(0).AlternatingRows(1).BackColor = Color.LightYellow
fpSpread1.Sheets(0).AlternatingRows(1).ForeColor = Color.Navy
fpSpread1.Sheets(0).AlternatingRows(2).BackColor = Color.Salmon
fpSpread1.Sheets(0).AlternatingRows(2).ForeColor = Color.Navy
```

Using the Spread Designer

1. Select the sheet tab for the sheet for which you want to set the alternating rows.
2. From the property list for that sheet, in the **Appearance** category, select the **AlternatingRows** property.
3. If you want to add additional alternating rows patterns, set the **Count** property to the number of patterns you

want.

- Click the **AlternatingRows** button to display the **AlternatingRow Collection Editor** as shown in the following figure.



- Select alternating row pattern for which to set properties.
- Set properties for the selected pattern using the property list.
- Click **OK** to close the **AlternatingRow Collection Editor**.
- From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting up Preview Rows

You can display a preview row to provide more information about a record. The preview row is displayed below the row for which it provides information. You can specify colors and other formatting for the preview row as well. The following figure shows preview rows with text (the rows without row header numbers):

	A	B	C	D
1		Preview Ro		
	The preview row content is empty			
2		Preview Ro		
	Preview Row Content is: The preview row conte			
3		Preview Ro		
	The preview row content is empty			

Set the **Visible** ('Visible Property' in the on-line documentation) property of the **PreviewRowInfo** ('PreviewRowInfo Class' in the on-line documentation) class to true to see the preview row. Use the **ColumnIndex** ('ColumnIndex Property' in the on-line documentation) property of the **PreviewRowInfo** ('PreviewRowInfo Class' in the on-line documentation) class ('ColumnIndex Property' in the on-line documentation) to specify which column's text you want to see in the preview row. If the cell text is null, then the preview row content is null. You can also use the **PreviewRowFetch** ('PreviewRowFetch Event' in the on-line documentation) event to specify the preview row text. You can set various properties for the **PreviewRowInfo**

(**'PreviewRowInfo Class' in the on-line documentation**) class such as **BackColor** (**'BackColor Property' in the on-line documentation**), **Border** (**'Border Property' in the on-line documentation**), **Font** (**'Font Property' in the on-line documentation**), and so on.

The column header or footer does not display a preview row. A child sheet in a hierarchy does not inherit the preview row settings from the parent sheet. If the sheet with the preview row has child sheets, the preview row is shown below the rows with plus symbols. If a row that has a preview row has spanned rows, the span is not displayed. The preview row is read-only (no keyboard or mouse events, focus, and/or selections).

The preview row is printed and exported to PDF when printing or printing to PDF. The height of the preview row can be increased to show all the preview row text. If there are multiple horizontal pages, the preview row content is displayed in the left-most page.

The API members involved in this feature include (see the **PreviewRowInfo** (**'PreviewRowInfo Class' in the on-line documentation**) class for a complete list):

- **PreviewRowInfo** (**'PreviewRowInfo Class' in the on-line documentation**) class
- **ColumnIndex** (**'ColumnIndex Property' in the on-line documentation**) property
- **BackColor** (**'BackColor Property' in the on-line documentation**) property
- **Visible** (**'Visible Property' in the on-line documentation**) property

Using the Properties Window

1. At design time, in the **Properties** window, select the Sheet.
2. In the **SheetView Collection Editor** (**Misc** section), select the **PreviewRowInfo** option and set properties.

Method to set

Set the **PreviewRowInfo** class properties for the sheet.

Example

This example sets preview row properties.

C#

```
private void Form1_Load(object sender, EventArgs e)
{
    FarPoint.Win.BevelBorder bord = new
    FarPoint.Win.BevelBorder(FarPoint.Win.BevelBorderType.Raised, Color.Red, Color.Blue);
    fpSpread1.Sheets[0].Cells[0, 1, 10, 1].Text = "Preview Row";
    fpSpread1.Sheets[0].PreviewRowInfo.Visible = true;
    fpSpread1.Sheets[0].PreviewRowInfo.BackColor = Color.BurlyWood;
    fpSpread1.Sheets[0].PreviewRowInfo.ForeColor = Color.Black;
    fpSpread1.Sheets[0].PreviewRowInfo.Border = bord;
}

private void fpSpread1_PreviewRowFetch(object sender,
FarPoint.Win.Spread.PreviewRowFetchEventArgs e)
{
    FarPoint.Win.Spread.SheetView sheetView = e.View.GetSheetView();
    if (sheetView.SheetName == "Sheet1")
    {
        if (e.PreviewRowContent == string.Empty)
            e.PreviewRowContent = "The preview row content is empty";
        if ((e.Row + 1) % 2 == 0)
            e.PreviewRowContent = string.Format("Preview Row Content is: {0}",
            e.PreviewRowContent);
    }
}
```

```
}  
}
```

VB

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)  
Handles MyBase.Load  
Dim bord As New FarPoint.Win.BevelBorder(FarPoint.Win.BevelBorderType.Raised,  
Color.DarkBlue, Color.Blue)  
fpSpread1.Sheets(0).Cells(0, 1, 10, 1).Text = "Preview Row"  
fpSpread1.Sheets(0).PreviewRowInfo.Visible = True  
fpSpread1.Sheets(0).PreviewRowInfo.BackColor = Color.BurlyWood  
fpSpread1.Sheets(0).PreviewRowInfo.ForeColor = Color.Black  
fpSpread1.Sheets(0).PreviewRowInfo.Border = bord  
End Sub  
  
Private Sub fpSpread1_PreviewRowFetch(ByVal sender As Object, ByVal e As  
FarPoint.Win.Spread.PreviewRowFetchEventArgs) Handles  
fpSpread1.PreviewRowFetch  
Dim sheetView As FarPoint.Win.Spread.SheetView  
sheetView = e.View.GetSheetView()  
If sheetView.SheetName = "Sheet1" Then  
If (e.PreviewRowContent = String.Empty) Then  
e.PreviewRowContent = "The preview row content is empty"  
End If  
If ((e.Row + 1) / 2 = 0) Then  
e.PreviewRowContent = String.Format("Preview Row Content is: {0}", e.PreviewRowContent)  
End If  
End If  
End Sub
```

Using the Spread Designer

1. Select **Sheet** from the drop-down combo list located on the top right side of the Designer.
2. From the **Misc** section, select the **PreviewRowInfo** option to set properties.
3. From the **File** menu, select **Save and Exit** to save the changes.

Input Data in Rows or Columns

You can restrict the portion of the sheet in which the user can enter data. You can set the size of the sheet in terms of rows and columns. You can hide rows or columns so the user does not have access to them at all, or set them as frozen so that users can see them but cannot move them out of view by scrolling. For more information on hidden rows and columns, refer to **Showing or Hiding a Row or Column**. For information on frozen rows and columns, refer to **Setting Fixed (Frozen) Rows or Columns**. For information on setting the number of rows and columns in a sheet, refer to **Customizing the Number of Rows or Columns**.

You can restrict the user from entering data beyond the data already in the sheet. Set the **RestrictRows** (**'RestrictRows Property' in the on-line documentation**) property and **RestrictColumns** (**'RestrictColumns Property' in the on-line documentation**) property to restrict or allow the user to enter data in a row or column on a sheet that is more than one row or column beyond the last row or column that contains data.

Using the Properties Window

1. At design time, in the **Properties** window, select the sheet.
2. In the **Behavior** category, select the **RestrictRows** (**'RestrictRows Property' in the on-line**

documentation) or the **RestrictColumns** ('**RestrictColumns Property**' in the on-line **documentation**) property.

3. Click the drop-down arrow to display the choices and select the value (True or False). Repeat this for each property.

Method to set

Set the **RestrictRows** ('**RestrictRows Property**' in the on-line **documentation**) or **RestrictColumns** ('**RestrictColumns Property**' in the on-line **documentation**) property for the sheet.

Example

C#

```
fpSpread1.ActiveSheet.RestrictColumns = true;  
fpSpread1.ActiveSheet.RestrictRows = true;
```

VB

```
fpSpread1.ActiveSheet.RestrictColumns = True  
fpSpread1.ActiveSheet.RestrictRows = True
```

Using the Spread Designer

1. Select **Sheet** from the drop-down combo list located on the top right side of the Designer.
2. From the **Behavior** section, select **True** or **False** for **RestrictRows** or **RestrictColumns**.
3. From the **File** menu, select **Save and Exit** to save the changes.

Rows or Columns That Have Data

You can work with the rows and columns of the sheet and distinguish which ones have data by using various members of the **SheetView** ('**SheetView Class**' in the on-line **documentation**) class. You can use the following methods available on the sheet:

- **GetLastNonEmptyColumn** ('**GetLastNonEmptyColumn Method**' in the on-line **documentation**) method
- **GetLastNonEmptyRow** ('**GetLastNonEmptyRow Method**' in the on-line **documentation**) method

You can return the number of rows and columns that have data using these properties:

- **NonEmptyColumnCount** ('**NonEmptyColumnCount Property**' in the on-line **documentation**) property
- **NonEmptyRowCount** ('**NonEmptyRowCount Property**' in the on-line **documentation**) property

Row/Column with Source Range

It is also possible to return the last row and column within a specified range of a sheet using the **IRange.End** method. This method accepts two parameters i.e. **direction enum** to specify the direction and **includeHidden** parameter to specify whether to include hidden rows and columns.

Refer to the following code to return the cell at the end of specified source range in left direction.

C#

```
// IRange.End(Direction.Left)
```

```

IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;
TestActiveSheet.Cells["A1"].Value = 1;
TestActiveSheet.Cells["C1:F1"].Value = 1;
TestActiveSheet.Columns[1].Hidden = true;
TestActiveSheet.Columns[5].Hidden = true;
// No data left source range
Debug.WriteLine(TestActiveSheet.Cells["B5"].End(Direction.Left, false).Address());
//or true. return A5
// Source range inside data range
Debug.WriteLine(TestActiveSheet.Cells["D1"].End(Direction.Left, false).Address());
//return A1
Debug.WriteLine(TestActiveSheet.Cells["D1"].End(Direction.Left, true).Address());
//return C1
// There is data left source range
Debug.WriteLine(TestActiveSheet.Cells["I1"].End(Direction.Left, false).Address());
//return E1
Debug.WriteLine(TestActiveSheet.Cells["I1"].End(Direction.Left, true).Address());
//return F1

```

Refer to the following code to return the cell at the end of specified source range in right direction.

C#

```

// IRange.End(Direction.Right)
IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;
TestActiveSheet.Cells["C1:F1"].Value = 1;
TestActiveSheet.Cells["H1"].Value = 1;
TestActiveSheet.Columns[2].Hidden = true;
TestActiveSheet.Columns[6].Hidden = true;
// No data right source range
Debug.WriteLine(TestActiveSheet.Cells["B5"].End(Direction.Right, false).Address());
//or true. return SF5
// Source range inside data range
Debug.WriteLine(TestActiveSheet.Cells["D1"].End(Direction.Right, false).Address());
//return H1
Debug.WriteLine(TestActiveSheet.Cells["D1"].End(Direction.Right, true).Address());
//return F1
//There is data right source range
Debug.WriteLine(TestActiveSheet.Cells["A1"].End(Direction.Right, false).Address());
//return D1
Debug.WriteLine(TestActiveSheet.Cells["A1"].End(Direction.Right, true).Address());
//return C1

```

Refer to the following code to return the cell at the end of specified source range in upward direction.

C#

```

// IRange.End(Direction.Up)
IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;
TestActiveSheet.Cells["C1"].Value = 1;
TestActiveSheet.Cells["C3:C6"].Value = 1;
TestActiveSheet.Rows[1].Hidden = true;
TestActiveSheet.Rows[5].Hidden = true;
// No data above source range
Debug.WriteLine(TestActiveSheet.Cells["B5"].End(Direction.Up, false).Address()); //or
true. return B1
// Source range inside data range
Debug.WriteLine(TestActiveSheet.Cells["C4"].End(Direction.Up, false).Address());

```

```
//return C1
Debug.WriteLine(TestActiveSheet.Cells["C4"].End(Direction.Up, true).Address());
//return C3
// There is data above source range
Debug.WriteLine(TestActiveSheet.Cells["C10"].End(Direction.Up, false).Address());
//return C5
Debug.WriteLine(TestActiveSheet.Cells["C10"].End(Direction.Up, true).Address());
//return C6
```

Refer to the following code to return the cell at the end of specified source range in downward direction.

C#

```
// IRange.End(Direction.Down)
IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;
TestActiveSheet.Cells["C3:C6"].Value = 1;
TestActiveSheet.Cells["C8"].Value = 1;
TestActiveSheet.Rows[2].Hidden = true;
TestActiveSheet.Rows[6].Hidden = true;
// No data below source range
Debug.WriteLine(TestActiveSheet.Cells["B5"].End(Direction.Down, false).Address());
//or true. return B500
// Source range inside data range
Debug.WriteLine(TestActiveSheet.Cells["C4"].End(Direction.Down, false).Address());
//return C8
Debug.WriteLine(TestActiveSheet.Cells["C4"].End(Direction.Down, true).Address());
//return C6
// There is data below source range
Debug.WriteLine(TestActiveSheet.Cells["C1"].End(Direction.Down, false).Address());
//return C4
Debug.WriteLine(TestActiveSheet.Cells["C1"].End(Direction.Down, true).Address());
//return C3
```

 The `IRange.End` method returns the first cell inside the source range.

Adding a Tag to a Row or Column

You can add an application tag to a row or column. If you prefer, you can associate data with any cell in the spreadsheet, or the cells in a column, a row, or the entire spreadsheet. The string data can be used to interact with a row or column, or to provide information to the application you create. The row data (column data), or row tag (column tag), is similar to item data you can provide for the spreadsheet or cells.

Since the row tag (column tag) is declared as an object, it is very flexible; it can be a number, a boolean, a string, or an instance of a class.

For more information on tags, refer to the **Tag** ('**Tag Property**' in the on-line documentation) property in the **Row** ('**Row Property**' in the on-line documentation) class or **Column** ('**Column Property**' in the on-line documentation) class and the **GetCellFromTag** ('**GetCellFromTag Method**' in the on-line documentation) method in the **SheetView** ('**SheetView Class**' in the on-line documentation) class.

Headers

You can customize the appearance of header cells. These tasks relate to setting the appearance of headers for rows or columns in the sheet:

- **Understanding Headers**
- **Showing or Hiding Headers**
- **Setting the Height or Width of Header**
- **Setting the Header Text**
- **Customizing the Appearance of Headers**
- **Creating a Header with Multiple Rows or Columns**
- **Creating a Span in a Header**
- **Customizing the Sheet Corner Appearance**

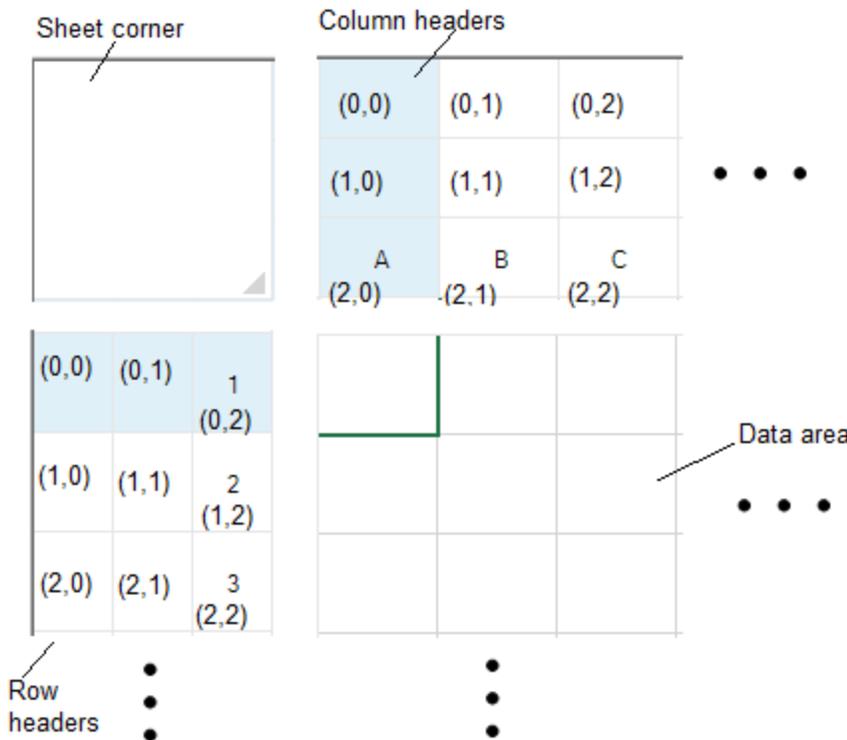
You can also customize header appearance in other ways.

- You can set the starting number for the default header labels.
- If you have multiple rows in the column headers, you can select which row displays the sort indicator and which row displays the automatic text.

For more information, refer to **Customizing the Appearance of Headers** and the **Cell ('Cell Class' in the on-line documentation)** and **Cells ('Cells Class' in the on-line documentation)** objects.

Understanding Headers

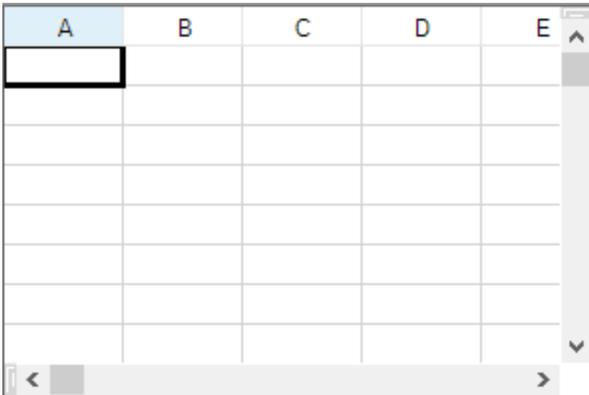
The following figure shows the sheet corner, headers, and cells and illustrates the cell coordinates in headers with multiple rows and columns. The coordinates are shown in parentheses.



When you work with row headers and column headers, you can work with the objects using the shortcut objects in code (**RowHeader ('RowHeader Class' in the on-line documentation)** and **ColumnHeader ('ColumnHeader Class' in the on-line documentation)** classes), or you can work directly with the model. Most developers who are not creating extensive customizations find it easier to work with the shortcut objects.

Showing or Hiding Headers

By default, Spread displays column headers and row headers. If you prefer, you can turn them off, and hide from view the row headers or column headers or both. The following figure shows a sheet that displays only column headers and hides the row headers.



If the sheet has multiple headers, using these instructions to hide the headers hides all header rows or header columns or both. If you want to hide specific rows or columns within a header, you must specify the row or column. For more details on hiding specific rows or columns, refer to **Showing or Hiding a Row or Column**.

The display of headers is done by simply setting a visible property of the header. This can be done in code with any of these properties:

Property

Visible ('Visible Property' in the on-line documentation) property of **RowHeader ('RowHeader Class' in the on-line documentation)** class
or

RowHeaderVisible ('RowHeaderVisible Property' in the on-line documentation) property of **SheetView ('SheetView Class' in the on-line documentation)** class

Visible ('Visible Property' in the on-line documentation) property of **ColumnHeader ('ColumnHeader Class' in the on-line documentation)** class
or

ColumnHeaderVisible ('ColumnHeaderVisible Property' in the on-line documentation) property of **SheetView ('SheetView Class' in the on-line documentation)** class

Descriptions

Set True to show the row header and False to hide the row header.

Set True to show the column header and False to hide the column header.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. Click the sheet for which you want to change the header display.
5. Set the **ColumnHeaderVisible** or **RowHeaderVisible** property to false to turn off the display of the header.

6. Click **OK** to close the editor.

Using a Shortcut

Set the **ColumnHeaderVisible** ('**ColumnHeaderVisible Property**' in the on-line documentation) or **RowHeaderVisible** ('**RowHeaderVisible Property**' in the on-line documentation) property for a Sheets object or the **Visible** ('**Visible Property**' in the on-line documentation) property of the ColumnHeader or RowHeader object.

Example

This example turns off the display of the column header. You can use either line of code.

C#

```
// Turn off the display of column headers.  
fpSpread1.Sheets[0].ColumnHeaderVisible = false;  
fpSpread1.Sheets[0].ColumnHeader.Visible = false;
```

VB

```
' Turn off the display of column headers.  
fpSpread1.Sheets(0).ColumnHeaderVisible = False  
fpSpread1.Sheets(0).ColumnHeader.Visible = False
```

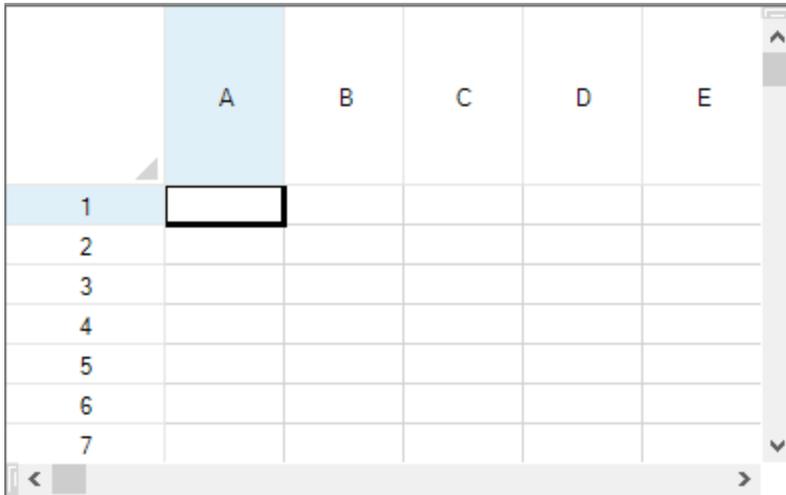
Using the Spread Designer

1. Select the sheet tab for the sheet for which you want to turn off header display.
2. In the properties list, in the **Appearance** category, double-click the **ColumnHeader** or **RowHeader** property to display the properties for the column or row header.
3. Set the **Visible** property to False to turn off the header display.
4. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting the Height or Width of Header

You can customize the appearance of header cells by changing the row height or column width, or both, in any of the rows or columns of headers.

You can change the size by accessing properties in the **RowHeader** ('**RowHeader Class**' in the on-line documentation) class for the row header or the **ColumnHeader** ('**ColumnHeader Class**' in the on-line documentation) class for the column header or both.



Method to set

Use the **Height ('Height Property' in the on-line documentation)** property in the **ColumnHeader ('ColumnHeader Class' in the on-line documentation)** class and the **Width ('Width Property' in the on-line documentation)** property in the **RowHeader ('RowHeader Class' in the on-line documentation)** class.

Example

This example sets the column header height and row header width.

C#

```
private void Form1_Load(object sender, System.EventArgs e)
{
    // Change the column header height to 90 pixels.
    fpSpread1.ActiveSheet.ColumnHeader.Rows[0].Height = 90;
    // Change the row header width to 80 pixels.
    fpSpread1.ActiveSheet.RowHeader.Columns[0].Width = 80;
}
```

VB

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.Load
    ' Change the column header height to 90 pixels.
    fpSpread1.ActiveSheet.ColumnHeader.Rows(0).Height = 90
    ' Change the row header width to 80 pixels.
    fpSpread1.ActiveSheet.RowHeader.Columns(0).Width = 80
End Sub
```

Setting the Header Text

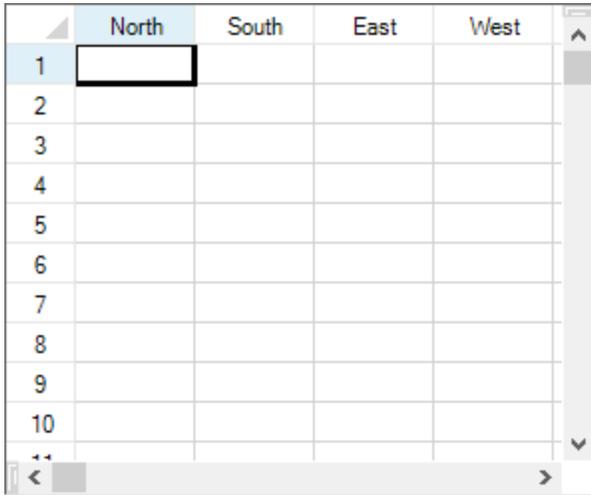
By default the Spread component displays sequential letters in the column headers and numbers in the row headers. This text is displayed in the bottom row if your sheet displays multiple rows in column header, and displayed in the right-most column if sheet displays multiple columns in row header.

Besides this automatic text, you can set any character in the column header and row header, or set the rule for automatic serial number. These methods are explained in the following topics.

- **Customizing Header Label Text**
- **Customizing the Default Header Labels**

Customizing Header Label Text

By default the Spread component displays letters in the column headers and numbers in the row headers. Besides this automatic text, you can add labels to any or all of the header cells. You can add customize the header label text, as shown in the following figure where the first four columns have custom labels.



The image shows a spreadsheet grid with 10 rows and 5 columns. The first four columns have custom header labels: 'North', 'South', 'East', and 'West'. The first row is numbered '1' in the first column, and the rest of the rows are numbered '2' through '10'. The grid has a scroll bar on the right and a scroll bar at the bottom.

	North	South	East	West
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				

To specify the custom text for a header label, you can use the **Label** property of the **ColumnHeader.Column** or **RowHeader.Row** shortcut objects or you can use the **Text** (**'Text Property' in the on-line documentation**) property of the **Cells** shortcut objects. For headers with multiple columns and multiple rows, you use the **Text** property of the **Cells** shortcut objects. Refer to the example in **Creating a Header with Multiple Rows or Columns**. For more information on the individual properties, refer to the **Label** (**'Label Property' in the on-line documentation**) property in the **Column** (**'Column Class' in the on-line documentation**) class or the **Label** (**'Label Property' in the on-line documentation**) property in the **Row** (**'Row Class' in the on-line documentation**) class.

Cells in the headers are separate from the cells in the data area, so the coordinates for cells in the headers start at 0,0 and count up from upper left to lower right within the header. The sheet corner cell is separate and is not counted when specifying header cell coordinates.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. Click the sheet for which you want to change the header labels.
You cannot add or change custom text in cells other than changing the labels displayed when using the **Properties** window.
5. In the property list, select the **Cells** property and click the button to display the **Cell, Column, and Row Editor**.
6. Select the column for which you want to change the labels displayed to custom text.
7. Set the **Label** property to set the custom text.
8. Click **OK** to close the **Cell, Column, and Row Editor**.
9. Click **OK** to close the **SheetView Collection Editor**.

Method to set

If you want to change the label display of the column header, refer to the target column in the Columns property of the ColumnHeader class, and after that, set custom text in **Label ('Label Property' in the on-line documentation)** property of **Column ('Column Class' in the on-line documentation)** class. If you want to change the label display of the row header, refer to the target row in the Rows property of RowHeader class and set custom text to **Label ('Label Property' in the on-line documentation)** property of **Row ('Row Class' in the on-line documentation)** class.

Example

This example code sets custom text for the labels in the first four column headers.

C#

```
// Set custom text for columns A through D.
fpSpread1.Sheets[0].ColumnHeader.Columns[0].Label = "North";
fpSpread1.Sheets[0].ColumnHeader.Columns[1].Label = "South";
fpSpread1.Sheets[0].ColumnHeader.Columns[2].Label = "East";
fpSpread1.Sheets[0].ColumnHeader.Columns[3].Label = "West";
```

VB

```
' Set custom text for columns A through D.
fpSpread1.Sheets(0).ColumnHeader.Columns(0).Label = "North"
fpSpread1.Sheets(0).ColumnHeader.Columns(1).Label = "South"
fpSpread1.Sheets(0).ColumnHeader.Columns(2).Label = "East"
fpSpread1.Sheets(0).ColumnHeader.Columns(3).Label = "West"
```

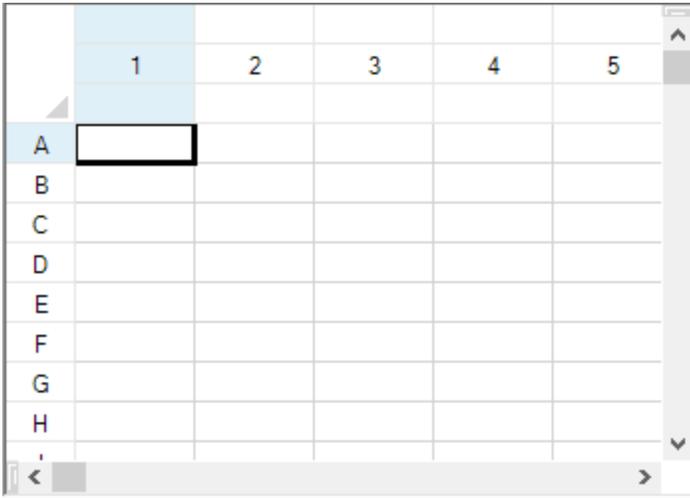
Using the Spread Designer

1. Select the sheet tab for the sheet for which you want to set custom header text.
2. Select the row or column for which you want to set custom header text, then right-click and choose **Headers**.
3. In the **Header Editor**, double-click the header for which you want to display custom text. If there is only one header row or column, click the one displayed cell with the text "<Default>".
4. Edit the text to be the custom text you want, and then press **Enter** to stop editing.
5. When you have edited all the header text you want to edit, click **OK** to close the **Header Editor**, then click **Yes** to apply your changes to the selection.
6. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Customizing the Default Header Labels

By default the Spread component displays sequential letters in the bottom row of the column header and sequentially increasing numbers in the right-most column of the row header. If your sheet displays multiple column header rows or row header columns, you can specify which column or row displays these default labels.

In the following figure, the column headers show numbers instead of letters and the labels are shown in the second row instead of the bottom row. You can also choose not to display the default labels.



You can also set the number (or letter) at which to start the sequential numbering (or lettering) of the labels using a property of the sheet. Use the **StartingColumnNumber** ('StartingColumnNumber Property' in the on-line documentation) property or **StartingRowNumber** ('StartingRowNumber Property' in the on-line documentation) property of the **SheetView** ('SheetView Class' in the on-line documentation) object to set the number or letter displayed in the first column header or first row header respectively on the sheet. The starting number or letter is used only for display purposes and has no effect on the actual row and column coordinates.

HeaderAutoText's Value

Blank
Letters
Numbers

Description

Do not display anything in the header
Display letters in header
Display numbers in header

In case of multiple rows in column header, you can set the row index to display the automatic serial number with the **SheetView** ('SheetView Class' in the on-line documentation) class's **ColumnHeaderAutoTextIndex** ('ColumnHeaderAutoTextIndex Property' in the on-line documentation) property. And, in case of multiple columns in row header, you can set the column index to display the automatic serial number with the **RowHeaderAutoTextIndex** ('RowHeaderAutoTextIndex Property' in the on-line documentation) property. Similar settings can be made using the **AutoTextIndex** ('AutoTextIndex Property' in the on-line documentation) property of the **ColumnHeader** class or the **AutoTextIndex** ('AutoTextIndex Property' in the on-line documentation) property of the **RowHeader** class.

You can also choose to display custom text in the headers instead of or in addition to the automatic label text. For instructions, see **Customizing Header Label Text**.

 **Note:** The value of a starting number or letter is an integer, so if the header displays letters and set the starting letter to 10, the first header cell contains the letter J.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. Click the sheet for which you want to change the header labels.
5. To change the header labels displayed, change the setting of the **ColumnHeaderAutoText** or **RowHeaderAutoText** property.
6. To change the row or column in the header in which the label is displayed, change the setting of the

ColumnHeaderAutoTextIndex or **RowHeaderAutoTextIndex** property.

7. Click **OK** to close the editor.

Method to set

1. To change the settings for the column header, set the Sheet object's **ColumnHeaderAutoText** (**'ColumnHeaderAutoText Property' in the on-line documentation**) and **ColumnHeaderAutoTextIndex** (**'ColumnHeaderAutoTextIndex Property' in the on-line documentation**) properties.
2. To change the settings for the row header, set the Sheet object's **RowHeaderAutoText** (**'RowHeaderAutoText Property' in the on-line documentation**) and **RowHeaderAutoTextIndex** (**'RowHeaderAutoTextIndex Property' in the on-line documentation**) properties.

Example

This example code sets the column header to display numbers instead of letters.

C#

```
// Set the column header to display numbers instead of letters.
fpSpread1.Sheets[0].ColumnHeader.RowCount = 3;
fpSpread1.Sheets[0].ColumnHeaderAutoTextIndex = 1;
fpSpread1.Sheets[0].ColumnHeaderAutoText = FarPoint.Win.Spread.HeaderAutoText.Numbers;
fpSpread1.Sheets[0].RowHeaderAutoText = FarPoint.Win.Spread.HeaderAutoText.Letters;
```

VB

```
' Set the column header to display numbers instead of letters.
fpSpread1.Sheets(0).ColumnHeader.RowCount = 3
fpSpread1.Sheets(0).ColumnHeaderAutoTextIndex = 1
fpSpread1.Sheets(0).ColumnHeaderAutoText = FarPoint.Win.Spread.HeaderAutoText.Numbers
fpSpread1.Sheets(0).RowHeaderAutoText = FarPoint.Win.Spread.HeaderAutoText.Letters
```

Using the Spread Designer

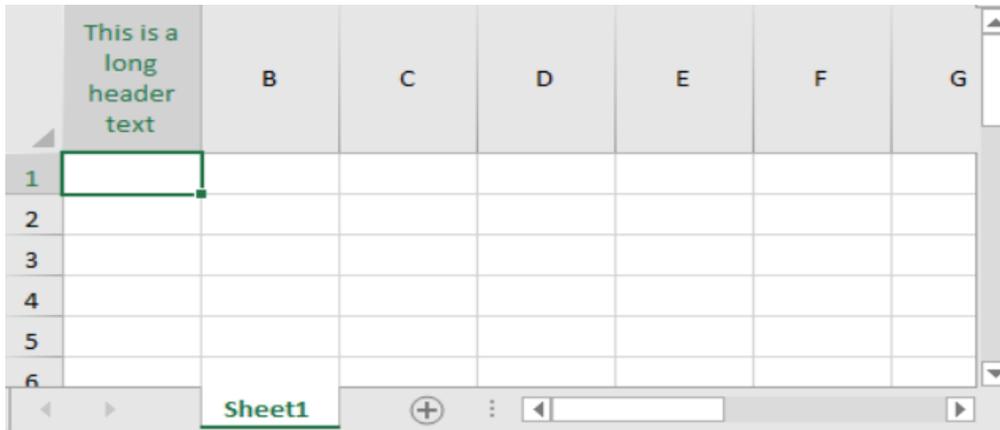
1. Select the sheet tab for the sheet for which you want to modify the header label (automatic text) settings.
2. In the properties list, in the **Appearance** category, double-click the **ColumnHeader** or **RowHeader** property to display the properties for the column or row header.
3. Change the settings of the **AutoText** and **AutoTextIndex** properties to specify the header label to display and which column or row in the header should display the automatic text.
4. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Wrapping the Header Text

In Spread, you can customize text wrap in a column header by setting the **WrapText** property of **IRange** (**'IRange Interface' in the on-line documentation**) Interface to true. Note that to view the wrapped text properly in the cell, you need to adjust the row height using the **RowHeight** property in the **ColumnHeader** class.

Before applying this **WrapText** property, ensure that the value of **ColumnHeaderRenderer.WordWrap2** property of the [SpreadSkin](#) class is set to null. If not, Spread uses the specified value of the **ColumnHeaderRenderer** (**'ColumnHeaderRenderer Property' in the on-line documentation**) property as the default renderer to paint cells in the **ColumnHeader** area. By default, the **ColumnHeaderRenderer.WordWrap2** value is null for the Default skin only.

The following image depicts a preview of the wrapped text in the column header.



Use the sample codes below to wrap the column header text using the **IRange.WrapText** property.

C#

```
// Wrap text in column header
activeSheet.ColumnHeader.Cells.RowHeight = 90;
activeSheet.ColumnHeader.Cells[0, 0].Value = "This is a long header text";
activeSheet.ColumnHeader.Cells[0, 0].WrapText = true;
```

VB

```
' Wrap text in column header
activeSheet.ColumnHeader.Cells.RowHeight = 90
activeSheet.ColumnHeader.Cells(0, 0).Value = "This is a long header text"
activeSheet.ColumnHeader.Cells(0, 0).WrapText = True
```

Customizing the Appearance of Headers

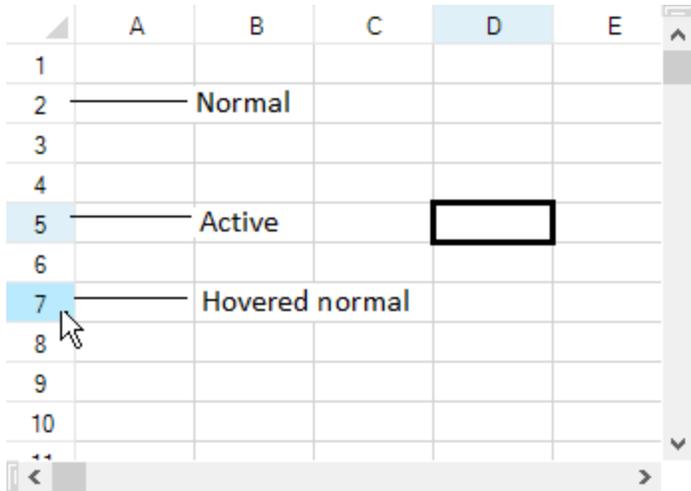
You can customize the appearance of header cells. These tasks relate to setting the appearance of headers for rows or columns in the sheet:

- **Customizing the Style of Header Cells**
- **Customizing the Header Grid Lines**
- **Adding a Gradient to Header Cells**
- **Auto Expand Row Headers**

You can also customize header appearance in other ways.

- You can set the starting number for the default header labels.
- If you have multiple rows in the column headers, you can select which row displays the sort indicator and which row displays the automatic text.
- You can show a row selector icon for the selected row (**ShowRowSelector** (**ShowRowSelector Property** in the on-line documentation)).

The appearance of the header cell is determined also by the state, whether it is selected (active) or not (normal), as shown in the figure below.



- For information on the appearance of the sheet corner, refer to **Customizing the Sheet Corner Appearance**
- For more information on cell-level properties of header cells, refer to the **Cell ('Cell Class' in the on-line documentation)** and **Cells ('Cells Class' in the on-line documentation)** objects.
- For more information on the painting of the header cells, refer to the **ColumnHeaderRenderer ('ColumnHeaderRenderer Class' in the on-line documentation)** class and **RowHeaderRenderer ('RowHeaderRenderer Class' in the on-line documentation)** class.

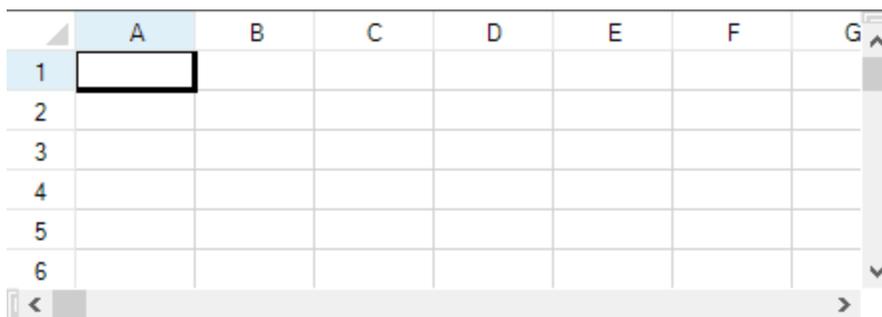
Customizing the Style of Header Cells

You can customize the style of header cells if you want to change the default appearance. You can set or customize many features, including:

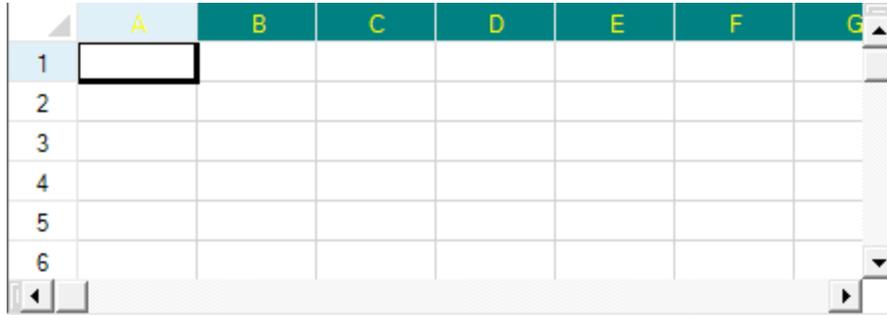
- style properties of the header classes
- members of the renderers
- properties of the default header classes

To customize style properties for the header classes, set the default style of the header cells by setting the **DefaultStyle ('DefaultStyle Property' in the on-line documentation)** property of the **RowHeader ('RowHeader Class' in the on-line documentation)** (**DefaultStyle Property' in the on-line documentation**) or the **DefaultStyle ('DefaultStyle Property' in the on-line documentation)** property of the **ColumnHeader ('ColumnHeader Class' in the on-line documentation)**. For more information on what can be set, refer to the **StyleInfo ('StyleInfo Class' in the on-line documentation)** object and the **RowHeader ('RowHeader Class' in the on-line documentation)** and **ColumnHeader ('ColumnHeader Class' in the on-line documentation)** objects.

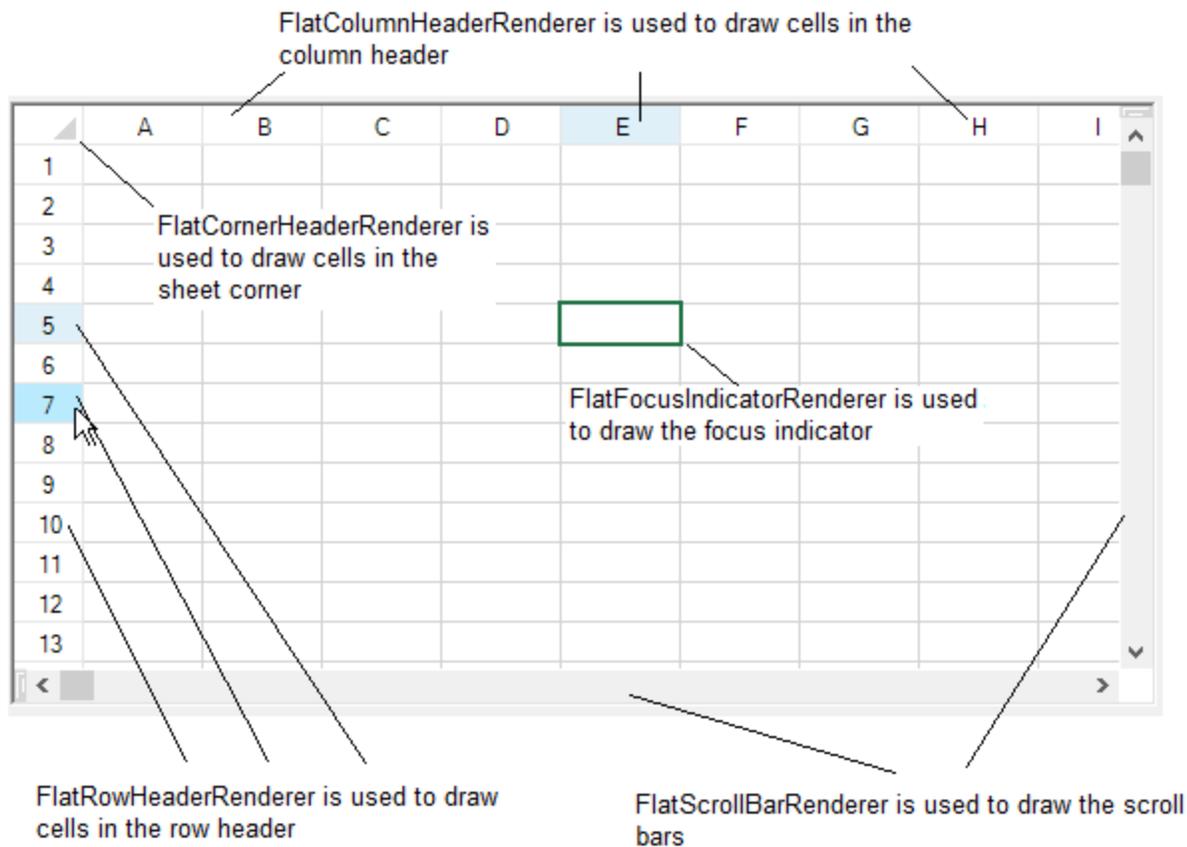
Before



After



To customize using the renderers, set their members to customize the appearance of headers. The renderers used in the Office2013 or Office 2016 style are shown and listed in the following figure.



You can also use default classes to customize many of the header appearance properties by setting the properties of the **Columns.DefaultColumn** ('Columns.DefaultColumn Class' in the on-line documentation) class and the **Rows.DefaultRow** ('Rows.DefaultRow Class' in the on-line documentation) class.

You can also set the grid lines around the header cells to change the three-dimensional appearance. Refer to **Customizing the Header Grid Lines**.

You can also add gradients to the header cells. Refer to **Adding a Gradient to Header Cells**.

Method to set

1. To change the style for the column header, define a style and then set the ColumnHeader object **DefaultStyle**

(**'DefaultStyle Property' in the on-line documentation**) property.

- To change the settings for the row header, define a style and then set the RowHeader object **DefaultStyle** (**'DefaultStyle Property' in the on-line documentation**) property.

Example

This example code defines a style with new colors and applies it to the column header as shown in the figure in this topic.

C#

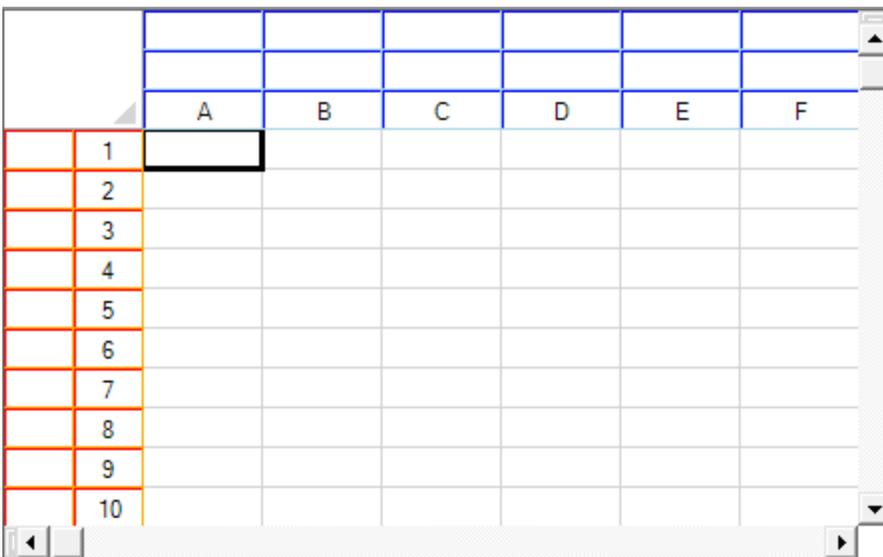
```
fpSpread1.VisualStyles = FarPoint.Win.VisualStyles.Off;
// Define a new style.
FarPoint.Win.Spread.StyleInfo darkstyle = new FarPoint.Win.Spread.StyleInfo();
darkstyle.BackColor = Color.Teal;
darkstyle.ForeColor = Color.Yellow;
// Apply the new style to the column header of a sheet.
fpSpread1.ActiveSheet.ColumnHeader.DefaultStyle = darkstyle;
```

VB

```
fpSpread1.VisualStyles = FarPoint.Win.VisualStyles.Off
' Define a new style.
Dim darkstyle As New FarPoint.Win.Spread.StyleInfo()
darkstyle.BackColor = Color.Teal
darkstyle.ForeColor = Color.Yellow
' Apply the new style to the column header of a sheet.
fpSpread1.ActiveSheet.ColumnHeader.DefaultStyle = darkstyle
```

Customizing the Header Grid Lines

You can specify the grid lines for a header in ways similar to setting the grid lines for a sheet. For more information on grid lines, refer to **Displaying Grid Lines on a Sheet**. In the following figure, the row header grid lines are three-dimensional with red hues and the column header grid lines are three-dimensional with blue hues.

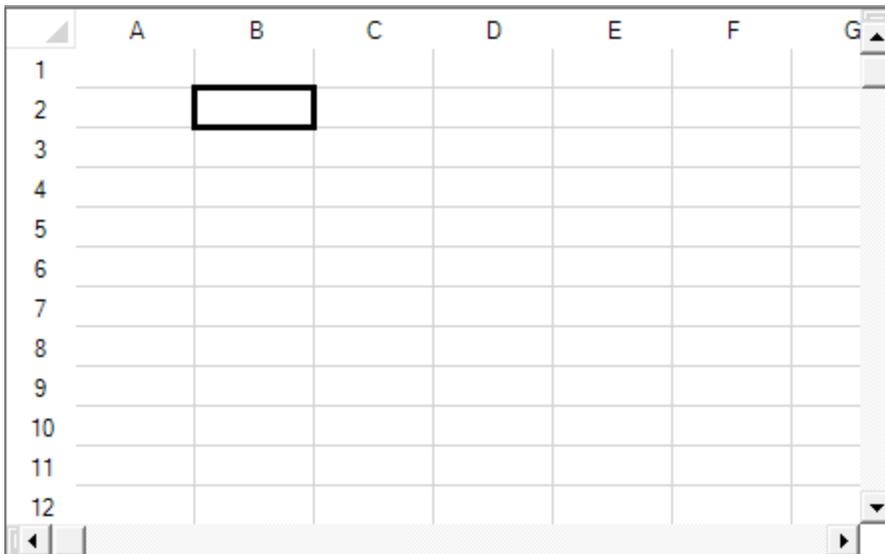


You can specify the horizontal grid lines separately from the vertical grid lines. You can specify the column header grid lines separately from the row header grid lines.

In code, you can customize the grid lines in a header either in the row or column header classes or in the **SheetView** (**'SheetView Class' in the on-line documentation**) class. You can use any of the following:

- **RowHeader** (**'RowHeader Class' in the on-line documentation**) class, **HorizontalGridLine** (**'HorizontalGridLine Property' in the on-line documentation**) property
- **RowHeader** (**'RowHeader Class' in the on-line documentation**) class, **VerticalGridLine** (**'VerticalGridLine Property' in the on-line documentation**) property
- **ColumnHeader** (**'ColumnHeader Class' in the on-line documentation**) class, **HorizontalGridLine** (**'HorizontalGridLine Property' in the on-line documentation**) property
- **ColumnHeader** (**'ColumnHeader Class' in the on-line documentation**) class, **VerticalGridLine** (**'VerticalGridLine Property' in the on-line documentation**) property
- **SheetView** (**'SheetView Class' in the on-line documentation**) class
RowHeaderHorizontalGridLine (**'RowHeaderHorizontalGridLine Property' in the on-line documentation**) property
- **SheetView** (**'SheetView Class' in the on-line documentation**) class **RowHeaderVerticalGridLine** (**'RowHeaderVerticalGridLine Property' in the on-line documentation**) property
- **SheetView** (**'SheetView Class' in the on-line documentation**) class
ColumnHeaderHorizontalGridLine (**'ColumnHeaderHorizontalGridLine Property' in the on-line documentation**) property
- **SheetView** (**'SheetView Class' in the on-line documentation**) class
ColumnHeaderVerticalGridLine (**'ColumnHeaderVerticalGridLine Property' in the on-line documentation**) property
- **GridLine** (**'GridLine Class' in the on-line documentation**) class

You can also set the headers to not show any grid lines, as shown in the following figure, by setting the grid line type to None.



Using the Properties Window

1. At design time, in the **Properties** window, select the particular sheet from the selection drop-down list.
2. Another way to select the sheet is to select the Spread component, select the **Sheets** property, and click the button to display the **SheetView Collection Editor**, and in the **Members** list, select the sheet.
3. To set the row header (or column header) horizontal grid line color or vertical grid line color,
 - a. Select the RowHeader object (or ColumnHeader object) in the property list, and expand the list of properties under that object.

- b. Select the **HorizontalGridLine** property or the **VerticalGridLine** property, and expand the list of properties under that object.
 - c. If you want to display three-dimensional grid lines, set the **Type** property to Lowered or Raised.
 - d. If the grid lines are not three-dimensional, select the **Color** property, and then click the drop-down button to display the color picker. Select a color in the color picker. If the grid lines are three-dimensional, select the **HighlightColor** property, and then click the drop-down button to display the color picker. Select a color in the color picker. Do the same for the **ShadowColor** property.
4. If you selected the sheet using the **Sheets Editor**, click **OK** to close the editor.

Method to set

1. Create a **GridLine ('GridLine Class' in the on-line documentation)** object, setting the color or colors and the style for the grid line in the constructor.
2. Assign the **GridLine ('GridLine Class' in the on-line documentation)** object to the specified header by setting the **RowHeader ('RowHeader Class' in the on-line documentation)** object (or **ColumnHeader ('ColumnHeader Class' in the on-line documentation)** object) **HorizontalGridLine ('HorizontalGridLine Property' in the on-line documentation)** or **VerticalGridLine ('VerticalGridLine Property' in the on-line documentation)** property to the **GridLine ('GridLine Class' in the on-line documentation)** object you created in step 1.
3. You can alternatively assign the **GridLine ('GridLine Class' in the on-line documentation)** object to the **RowHeaderHorizontalGridLine ('RowHeaderHorizontalGridLine Property' in the on-line documentation)**, **RowHeaderVerticalGridLine ('RowHeaderVerticalGridLine Property' in the on-line documentation)**, **ColumnHeaderHorizontalGridLine ('ColumnHeaderHorizontalGridLine Property' in the on-line documentation)**, or **ColumnHeaderVerticalGridLine ('ColumnHeaderVerticalGridLine Property' in the on-line documentation)** properties in the **SheetView** object.

Example

This example code sets the row header grid lines to be three-dimensional with red hues and the column header grid lines to be three-dimensional with blue hues as shown in the preceding figure.

C#

```
fpSpread1.Sheets[0].VisualStyles = FarPoint.Win.VisualStyles.Off;
fpSpread1.ActiveSheet.ColumnHeader.DefaultStyle.Renderer = new
FarPoint.Win.Spread.CellType.ColumnHeaderRenderer();
fpSpread1.ActiveSheet.RowHeader.DefaultStyle.Renderer = new
FarPoint.Win.Spread.CellType.RowHeaderRenderer();
FarPoint.Win.Spread.GridLine rgdln = new
FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Raised, Color.Purple,
Color.Red, Color.Orange);
FarPoint.Win.Spread.GridLine cgdln = new
FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Raised, Color.Purple,
Color.Blue, Color.LightBlue);
fpSpread1.Sheets[0].ColumnHeader.RowCount = 3;
fpSpread1.Sheets[0].RowHeader.ColumnCount = 2;
fpSpread1.Sheets[0].ColumnHeader.HorizontalGridLine = cgdln;
fpSpread1.Sheets[0].RowHeader.HorizontalGridLine = rgdln;
fpSpread1.Sheets[0].ColumnHeader.VerticalGridLine = cgdln;
fpSpread1.Sheets[0].RowHeader.VerticalGridLine = rgdln;
```

VB

```
fpSpread1.Sheets(0).VisualStyles = FarPoint.Win.VisualStyles.Off
fpSpread1.ActiveSheet.ColumnHeader.DefaultStyle.Renderer = New
```

```
FarPoint.Win.Spread.CellType.ColumnHeaderRenderer
fpSpread1.ActiveSheet.RowHeader.DefaultStyle.Renderer = New
FarPoint.Win.Spread.CellType.RowHeaderRenderer
Dim rgdln As New FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Raised,
Color.Purple, Color.Red, Color.Orange)
Dim cgdln As New FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Raised,
Color.Purple, Color.Blue, Color.LightBlue)
fpSpread1.Sheets(0).ColumnHeader.RowCount = 3
fpSpread1.Sheets(0).RowHeader.ColumnCount = 2
fpSpread1.Sheets(0).ColumnHeader.HorizontalGridLine = cgdln
fpSpread1.Sheets(0).RowHeader.HorizontalGridLine = rgdln
fpSpread1.Sheets(0).ColumnHeader.VerticalGridLine = cgdln
fpSpread1.Sheets(0).RowHeader.VerticalGridLine = rgdln
```

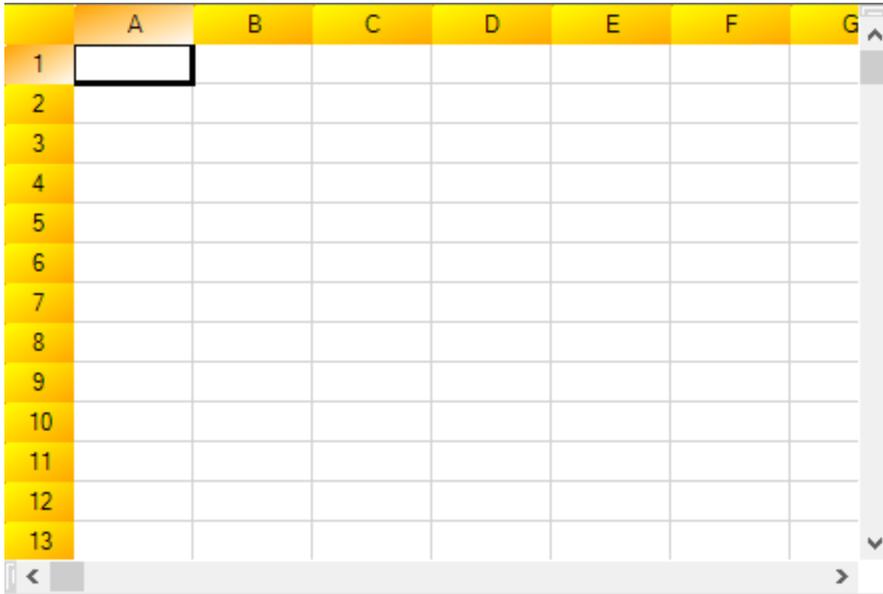
Using the Spread Designer

1. Select the sheet tab for the sheet for which you want to set the grid line colors. You can select the sheet from the drop-down list or select the sheet corner or select the Spread component, then select the **Sheets** property and display the **Sheets Editor**.
2. To set the row header (or column header) horizontal grid line color or vertical grid line color,
 - a. In the **Appearance** category, select the **RowHeaderHorizontalGridLine** or **RowHeaderVerticalGridLine** object (or **ColumnHeaderHorizontalGridLine** or **ColumnHeaderVerticalGridLine** object) in the property list, and expand the list of properties.
 - b. If you want to display three-dimensional grid lines, set the **Type** property to Lowered or Raised.
 - c. If the grid lines are not three-dimensional, select the **Color** property, and then click the drop-down button to display the color picker. Select a color in the color picker.
 - d. If the grid lines are three-dimensional, select the **HighlightColor** property, and then click the drop-down button to display the color picker. Select a color in the color picker. Do the same for the **ShadowColor** property.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Adding a Gradient to Header Cells

You can change the appearance of header cells by adding a color gradient. You can have a gradient from one color to another color.

You can implement a gradient appearance by creating a custom class that inherits existing cell classes (a general cell, in this case) and change the display to a gradient. You can also use the **GradientHeaderRenderer** (**'GradientHeaderRenderer Class' in the on-line documentation**) class.



Method to set

Use the **GradientHeaderRenderer** ('GradientHeaderRenderer Class' in the on-line documentation) class and the **Renderer** ('Renderer Property' in the on-line documentation) property for the column header row and row header column.

Example

This example sets a gradient for the headers and the sheet corner.

C#

```
FarPoint.Win.Spread.CellType.GradientHeaderRenderer gr = new
FarPoint.Win.Spread.CellType.GradientHeaderRenderer (Color.Yellow, Color.Orange,
Color.YellowGreen, Color.Bisque, Drawing2D.LinearGradientMode.ForwardDiagonal);
fpSpread1.ActiveSheet.ColumnHeader.Rows[0].Renderer = gr;
fpSpread1.ActiveSheet.RowHeader.Columns[0].Renderer = gr;
fpSpread1.ActiveSheet.SheetCorner.DefaultStyle.Renderer = gr;
```

VB

```
Dim gr As New FarPoint.Win.Spread.CellType.GradientHeaderRenderer (Color.Yellow,
Color.Orange, Color.YellowGreen, Color.Bisque,
Drawing2D.LinearGradientMode.ForwardDiagonal)
fpSpread1.ActiveSheet.ColumnHeader.Rows(0).Renderer = gr
fpSpread1.ActiveSheet.RowHeader.Columns(0).Renderer = gr
fpSpread1.ActiveSheet.SheetCorner.DefaultStyle.Renderer = gr
```

Auto Expand Row Headers

To prevent a header from truncating, users can configure the header to auto increase the row width just like in Excel.

Spread for Winforms provides support for automatic expansion of the width of the row header labels in order to accommodate the text when users scroll down past the rows (1,000, 10,000, etc.) in a spreadsheet.

While scrolling across the worksheet, Spread control extends the width of row header to display the row labels completely as per the following rules.

- The **RowHeaderAutoWidthMax ('RowHeaderAutoWidthMax Property' in the on-line documentation)** property of the **FpSpread ('FpSpread Class' in the on-line documentation)** class can be used to enable the auto expand feature in the spreadsheet. This property indicates the maximum width that the row header can be extended to, as per the following points:
 - By default, the value of the **RowHeaderAutoWidthMax** property is set to -1.
 - If the value of this property is 0, it means that the auto expand feature is disabled.
 - If the value of this property is set to a positive number, then the row header width must be less than equal to the **RowHeaderAutoWidthMax**.
 - And, if the value of this property is set to a negative number, then the row header width will be automatically calculated by the Spread control.
- When the auto expand row header feature is enabled, the width of rowHeader.Columns [SheetView.RowHeaderAutoTextIndex] must be -1.
- If the row header contains multiple columns, there is only one column which shows the row index automatically (indicated by the **RowHeaderAutoTextIndex ('RowHeaderAutoTextIndex Property' in the on-line documentation)** property of the **SheetView ('SheetView Class' in the on-line documentation)** class). The auto expand feature works with that particular column only. And, to make this feature work, users must ensure that its width is set to -1.

Creating a Header with Multiple Rows or Columns

You can provide multiple rows in the column header and multiple columns in the row header. As shown in the following figure, the headers may have different numbers of columns and rows.

		Fiscal Year 2004							
		1st Quarter		2nd Quarter		3rd Quarter		4th Quarter	
		East	West	East	West	East	West	East	West
Branch #	1								
	2								
	3								
	4								
	5								
	6								
	7								
	8								
	9								
	10								
	11								

The rows or columns in the header can also contain spans, for example, if you want to have a header cell that explains two cells beneath it (or subheaders). For instructions for creating a span in a header, see **Creating a Span in a Header**.

You can customize the labels in these headers. For instructions for customizing the labels, see **Customizing Header Label Text**. For more information on the individual properties, refer to the **Column ('Column Class' in the on-line documentation) Label ('Label Property' in the on-line documentation)** property or the **Row ('Row Class' in the on-line documentation) Label ('Label Property' in the on-line documentation)** property.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. Click the sheet for which you want to change the header display.
5. Set the **ColumnHeaderRowCount** property to the number of rows you want in the column header or the **RowHeaderColumnCount** property to the number of columns you want in the row header.
6. Click **OK** to close the editor.

Using a Shortcut

Set the **ColumnHeaderRowCount** ('**ColumnHeaderRowCount Property**' in the on-line documentation) property or the **RowHeaderColumnCount** ('**RowHeaderColumnCount Property**' in the on-line documentation) property for a Sheets object. Use the **AddColumnHeaderSpanCell** ('**AddColumnHeaderSpanCell Method**' in the on-line documentation) method to span the cells in the header. Use the **Label** and **Text** properties to add the labels to the header cells.

Example

This example code creates a spreadsheet shown in the figure above, with two columns in the row header and three rows in the column header.

C#

```
// Set the number of rows and columns in the headers.
fpSpread1.Sheets[0].ColumnHeaderRowCount = 3;
fpSpread1.Sheets[0].RowHeaderColumnCount = 2;
// Span the header cells as needed.
fpSpread1.Sheets[0].AddColumnHeaderSpanCell(1, 0, 1, 2);
fpSpread1.Sheets[0].AddColumnHeaderSpanCell(1, 2, 1, 2);
fpSpread1.Sheets[0].AddColumnHeaderSpanCell(1, 4, 1, 2);
fpSpread1.Sheets[0].AddColumnHeaderSpanCell(1, 6, 1, 2);
fpSpread1.Sheets[0].AddColumnHeaderSpanCell(0, 0, 1, 8);
fpSpread1.Sheets[0].AddRowHeaderSpanCell(0, 0, 12, 1);
// Set the labels as needed -- using the Label property or
// the cell Text property.
fpSpread1.Sheets[0].ColumnHeader.Columns[0].Label = "East";
fpSpread1.Sheets[0].ColumnHeader.Columns[1].Label = "West";
fpSpread1.Sheets[0].ColumnHeader.Columns[2].Label = "East";
fpSpread1.Sheets[0].ColumnHeader.Columns[3].Label = "West";
fpSpread1.Sheets[0].ColumnHeader.Columns[4].Label = "East";
fpSpread1.Sheets[0].ColumnHeader.Columns[5].Label = "West";
fpSpread1.Sheets[0].ColumnHeader.Columns[6].Label = "East";
fpSpread1.Sheets[0].ColumnHeader.Columns[7].Label = "West";
fpSpread1.Sheets[0].ColumnHeader.Cells[0,0].Text = "Fiscal Year 2004";
fpSpread1.Sheets[0].ColumnHeader.Cells[1,0].Text = "1st Quarter";
fpSpread1.Sheets[0].ColumnHeader.Cells[1,2].Text = "2nd Quarter";
fpSpread1.Sheets[0].ColumnHeader.Cells[1,4].Text = "3rd Quarter";
fpSpread1.Sheets[0].ColumnHeader.Cells[1,6].Text = "4th Quarter";
// Set the row header so that the label displays.
fpSpread1.Sheets[0].RowHeader.Columns[0].Width = 45;
fpSpread1.Sheets[0].RowHeader.Cells[0,0].Text = "Branch #";
```

VB

```
' Set the number or rows and columns in the headers.
fpSpread1.Sheets(0).ColumnHeaderRowCount = 3
fpSpread1.Sheets(0).RowHeaderColumnCount = 2
' Span the header cells as needed.
fpSpread1.Sheets(0).AddColumnHeaderSpanCell(1, 0, 1, 2)
fpSpread1.Sheets(0).AddColumnHeaderSpanCell(1, 2, 1, 2)
fpSpread1.Sheets(0).AddColumnHeaderSpanCell(1, 4, 1, 2)
fpSpread1.Sheets(0).AddColumnHeaderSpanCell(1, 6, 1, 2)
fpSpread1.Sheets(0).AddColumnHeaderSpanCell(0, 0, 1, 8)
fpSpread1.Sheets(0).AddRowHeaderSpanCell(0, 0, 12, 1)
' Set the labels as needed -- using the Label property or
' the cell Text property.
fpSpread1.Sheets(0).ColumnHeader.Columns(0).Label = "East"
fpSpread1.Sheets(0).ColumnHeader.Columns(1).Label = "West"
fpSpread1.Sheets(0).ColumnHeader.Columns(2).Label = "East"
fpSpread1.Sheets(0).ColumnHeader.Columns(3).Label = "West"
fpSpread1.Sheets(0).ColumnHeader.Columns(4).Label = "East"
fpSpread1.Sheets(0).ColumnHeader.Columns(5).Label = "West"
fpSpread1.Sheets(0).ColumnHeader.Columns(6).Label = "East"
fpSpread1.Sheets(0).ColumnHeader.Columns(7).Label = "West"
fpSpread1.Sheets(0).ColumnHeader.Cells(0,0).Text = "Fiscal Year 2004"
fpSpread1.Sheets(0).ColumnHeader.Cells(1,0).Text = "1st Quarter"
fpSpread1.Sheets(0).ColumnHeader.Cells(1,2).Text = "2nd Quarter"
fpSpread1.Sheets(0).ColumnHeader.Cells(1,4).Text = "3rd Quarter"
fpSpread1.Sheets(0).ColumnHeader.Cells(1,6).Text = "4th Quarter"
' Set the row header so that the label displays.
fpSpread1.Sheets(0).RowHeader.Columns(0).Width = 45
fpSpread1.Sheets(0).RowHeader.Cells(0,0).Text = "Branch #"
```

Using the Spread Designer

1. Select the sheet tab for the sheet for which you want to display multiple header rows or columns.
2. In the properties list, in the **Appearance** category, double-click the **ColumnHeader** or **RowHeader** property to display the properties for the column or row header.
3. Set the **RowCount** property to the number of rows you want in the column header or the **ColumnCount** property to the number of columns you want in the row header.
4. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Creating a Span in a Header

You can create cell spans in a header, for example, to make a header for multiple columns or rows, or both, as shown in the figure below.

		Fiscal Year 2004							
		1st Quarter		2nd Quarter		3rd Quarter		4th Quarter	
		East	West	East	West	East	West	East	West
Branch #	1								
	2								
	3								
	4								
	5								
	6								
	7								
	8								
	9								
	10								
	11								

You can create cell spans in either the column headers or row headers or both. For more background about creating cell spans, refer to **Creating a Span of Cells**.

Use these methods when creating a span in a header:

- **AddColumnHeaderSpanCell** ('AddColumnHeaderSpanCell Method' in the on-line documentation)
- **AddRowHeaderSpanCell** ('AddRowHeaderSpanCell Method' in the on-line documentation)

You can specify how the header spans are selected with the **CellSpanSelectionPolicy** ('CellSpanSelectionPolicy Property' in the on-line documentation) property.

For information on creating multiple rows in the column headers or multiple columns in the row headers, refer to **Creating a Header with Multiple Rows or Columns**.

You can customize the labels in these headers. For instructions for customizing the labels, see **Setting the Header Text**.

Using a Shortcut

Call the Sheets object **AddColumnHeaderSpanCell** ('AddColumnHeaderSpanCell Method' in the on-line documentation) or **AddRowHeaderSpanCell** ('AddRowHeaderSpanCell Method' in the on-line documentation) method.

Example

This example code sets the first cell in the column header to span across two columns.

C#

```
// Set the number of rows in the column header.
fpSpread1.ActiveSheet.ColumnHeader.RowCount = 3;
// Set the number of columns in the row header.
fpSpread1.ActiveSheet.RowHeader.ColumnCount = 2;
// Define the labels for the spanned column header cells.
fpSpread1.ActiveSheet.ColumnHeader.Cells[2, 0].Text = "East";
fpSpread1.ActiveSheet.ColumnHeader.Cells[2, 1].Text = "West";
fpSpread1.ActiveSheet.ColumnHeader.Cells[2, 2].Text = "East";
fpSpread1.ActiveSheet.ColumnHeader.Cells[2, 3].Text = "West";
```

```

fpSpread1.ActiveSheet.ColumnHeader.Cells[2, 4].Text = "East";
fpSpread1.ActiveSheet.ColumnHeader.Cells[2, 5].Text = "West";
fpSpread1.ActiveSheet.ColumnHeader.Cells[2, 6].Text = "East";
fpSpread1.ActiveSheet.ColumnHeader.Cells[2, 7].Text = "West";
fpSpread1.ActiveSheet.ColumnHeader.Cells[1, 0].Text = "1st Quarter";
fpSpread1.ActiveSheet.ColumnHeader.Cells[1, 2].Text = "2nd Quarter";
fpSpread1.ActiveSheet.ColumnHeader.Cells[1, 4].Text = "3rd Quarter";
fpSpread1.ActiveSheet.ColumnHeader.Cells[1, 6].Text = "4th Quarter";
fpSpread1.ActiveSheet.ColumnHeader.Cells[0, 0].Text = "Fiscal Year 2004";
// Define the column header cell spans.
fpSpread1.ActiveSheet.AddColumnHeaderSpanCell(1, 0, 1, 2);
fpSpread1.ActiveSheet.AddColumnHeaderSpanCell(1, 2, 1, 2);
fpSpread1.ActiveSheet.AddColumnHeaderSpanCell(1, 4, 1, 2);
fpSpread1.ActiveSheet.AddColumnHeaderSpanCell(1, 6, 1, 2);
fpSpread1.ActiveSheet.AddColumnHeaderSpanCell(0, 0, 1, 8);
// Define the label for the spanned row header cells.
fpSpread1.ActiveSheet.RowHeader.Cells[0,0].Text ="Branch #";
// Define the row header cell span.
fpSpread1.ActiveSheet.AddRowHeaderSpanCell(0, 0, 12, 1);

```

VB

```

' Set the number of rows in the column header.
fpSpread1.ActiveSheet.ColumnHeader.RowCount = 3
' Set the number of columns in the row header.
fpSpread1.ActiveSheet.RowHeader.ColumnCount = 2
' Define the labels for the spanned column header cells.
fpSpread1.ActiveSheet.ColumnHeader.Cells(2, 0).Text = "East"
fpSpread1.ActiveSheet.ColumnHeader.Cells(2, 1).Text = "West"
fpSpread1.ActiveSheet.ColumnHeader.Cells(2, 2).Text = "East"
fpSpread1.ActiveSheet.ColumnHeader.Cells(2, 3).Text = "West"
fpSpread1.ActiveSheet.ColumnHeader.Cells(2, 4).Text = "East"
fpSpread1.ActiveSheet.ColumnHeader.Cells(2, 5).Text = "West"
fpSpread1.ActiveSheet.ColumnHeader.Cells(2, 6).Text = "East"
fpSpread1.ActiveSheet.ColumnHeader.Cells(2, 7).Text = "West"
fpSpread1.ActiveSheet.ColumnHeader.Cells(1, 0).Text = "1st Quarter"
fpSpread1.ActiveSheet.ColumnHeader.Cells(1, 2).Text = "2nd Quarter"
fpSpread1.ActiveSheet.ColumnHeader.Cells(1, 4).Text = "3rd Quarter"
fpSpread1.ActiveSheet.ColumnHeader.Cells(1, 6).Text = "4th Quarter"
fpSpread1.ActiveSheet.ColumnHeader.Cells(0, 0).Text = "Fiscal Year 2004"
' Define the column header cell spans.
fpSpread1.ActiveSheet.AddColumnHeaderSpanCell(1, 0, 1, 2)
fpSpread1.ActiveSheet.AddColumnHeaderSpanCell(1, 2, 1, 2)
fpSpread1.ActiveSheet.AddColumnHeaderSpanCell(1, 4, 1, 2)
fpSpread1.ActiveSheet.AddColumnHeaderSpanCell(1, 6, 1, 2)
fpSpread1.ActiveSheet.AddColumnHeaderSpanCell(0, 0, 1, 8)
' Define the label for the spanned row header cells.
fpSpread1.ActiveSheet.RowHeader.Cells(0,0).Text ="Branch #"
' Define the row header cell span.
fpSpread1.ActiveSheet.AddRowHeaderSpanCell(0, 0, 12, 1)

```

Using the Spread Designer

1. Select the sheet tab for the sheet for which you want to span header cells.
2. Select the row or column for which you want to set custom header text, then right-click and choose **Headers**.
3. In the **Header Editor**, click the left-most header cell in the set of cells for which you want to create a span.

4. In the property list for that header cell, set the **ColumnSpan** or **RowSpan** property to the number of cells to span starting from the selected header cell.
5. Click **OK** to close the **Header Editor**.
6. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Customizing the Sheet Corner Appearance

You can customize the appearance of the sheet corner, the header cell in the upper left corner of the sheet, for each sheet. In many ways, customizing the sheet corner is similar to customizing cells or sheets.

These topics discuss the sheet corner features:

- **General Style of the Sheet Corner**
- **Text Display in the Sheet Corner**
- **Table Display in the Sheet Corner**
- **Customizable Cell in the Sheet Corner**
- **Cell Spans in the Sheet Corner**
- **Header Count Synchronization in the Sheet Corner**
- **Drawing (Rendering) Style**

Operational Support

The sheet corner supports XML serialization and deserialization along with sheet. Anything changed in the sheet corner is saved with the Spread when you save Spread.

The sheet corner supports copy and pasting cells between sheets. You can select a sheet by clicking in sheet corner, then press Ctrl +C to copy it and then go to another sheet and press Ctrl +V to paste the sheet and the sheet corner.

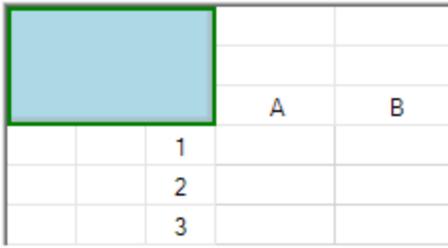
The sheet corner supports PDF printing and normal printing so you can print the sheet corner along with the sheet.

The underlying models of the sheet corner are exposed through the sheet model property and you can change this model. The following classes are involved in the sheet corner.

- FarPoint.Win.Spread Namespace: **SheetCorner ('SheetCorner Class' in the on-line documentation)** Class
- FarPoint.Win.Spread.CellType Namespace: **IRenderer ('IRenderer Interface' in the on-line documentation)** Interface
- FarPoint.Win.Spread.CellType Namespace: **CornerRenderer ('CornerRenderer Class' in the on-line documentation)** Class
- FarPoint.Win.Spread.CellType Namespace: **EnhancedCornerRenderer ('EnhancedCornerRenderer Class' in the on-line documentation)** Class

General Style of the Sheet Corner

Sheet corners can display grid lines, have a different background color from the rest of the headers, and more. There are several different ways to set properties in the sheet corner. One way is with the **SheetCorner ('SheetCorner Class' in the on-line documentation)** class. Another option is to set the sheet corner properties for the **SheetView ('SheetView Class' in the on-line documentation)** class. In the following figure, the sheet corner is displayed with a two-pixel wide, green border and a light blue background.



The sheet corner supports right-to-left orientation. When you set Right-to-Left mode in Spread for the entire spreadsheet, the sheet corner also displays with right-to-left orientation as well.

General Customization

Several of the properties of a **StyleInfo ('StyleInfo Class' in the on-line documentation)** class can be set for the sheet corner cell. These properties include:

Property

Border ('Border Property' in the on-line documentation)

CellType ('CellType Property' in the on-line documentation)

HorizontalAlignment ('HorizontalAlignment Property' in the on-line documentation) and **VerticalAlignment ('VerticalAlignment Property' in the on-line documentation)**

Description

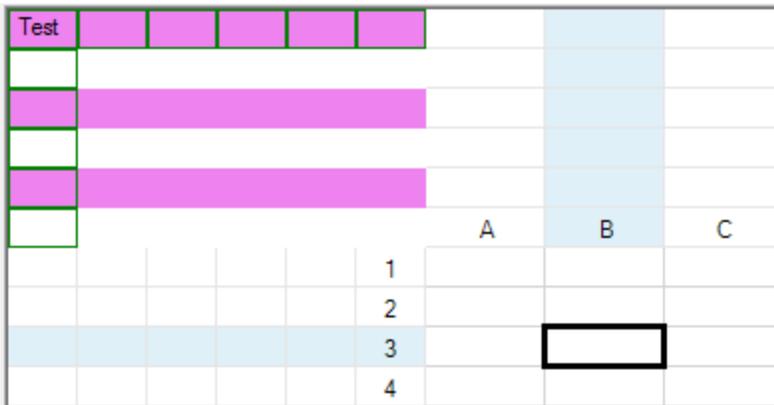
The border around the cell

The type of cell

The alignment of text in the cell (horizontal and vertical)

Using SheetCorner class

The following figure displays a sheet corner that has been customized with the **SheetCorner ('SheetCorner Class' in the on-line documentation)** class.



The following sections provide instructions and example code for customizing many aspects of the sheet corner.

Using Code

You can set the sheet corner style using the properties of the **SheetCorner ('SheetCorner Class' in the on-line documentation)** class.

Example

This example code sets the background color, sets borders, puts text in a cell, and sets the row and column count.

C#

```

fpSpread1.ActiveSheet.SheetCorner.DefaultStyle.Renderer = new
FarPoint.Win.Spread.CellType.CornerRenderer();
fpSpread1.ActiveSheet.AllowTableCorner = true;
fpSpread1.ActiveSheet.SheetCorner.RowCount = 6;
fpSpread1.ActiveSheet.SheetCorner.ColumnCount = 6;
fpSpread1.ActiveSheet.SheetCorner.AlternatingRows[0].BackColor = Color.Violet;
fpSpread1.ActiveSheet.SheetCorner.Cells[0, 0].Text = "Test";
fpSpread1.ActiveSheet.SheetCorner.Columns[0].Border = new
FarPoint.Win.LineBorder(Color.Green);
fpSpread1.ActiveSheet.SheetCorner.Rows[0].Border = new
FarPoint.Win.LineBorder(Color.Green);
fpSpread1.ActiveSheet.SheetCorner.HorizontalGridLine = new
FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.None);
fpSpread1.ActiveSheet.SheetCorner.VerticalGridLine = new
FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.None);
fpSpread1.ActiveSheet.SheetCorner.DefaultStyle.VisualStyle =
FarPoint.Win.VisualStyle.Off;

```

VB

```

fpSpread1.ActiveSheet.SheetCorner.DefaultStyle.Renderer = New
FarPoint.Win.Spread.CellType.CornerRenderer
fpSpread1.ActiveSheet.AllowTableCorner = True
fpSpread1.ActiveSheet.SheetCorner.RowCount = 6
fpSpread1.ActiveSheet.SheetCorner.ColumnCount = 6
fpSpread1.ActiveSheet.SheetCorner.AlternatingRows(0).BackColor = Color.Violet
fpSpread1.ActiveSheet.SheetCorner.Cells(0, 0).Text = "Test"
fpSpread1.ActiveSheet.SheetCorner.Columns(0).Border = New
FarPoint.Win.LineBorder(Color.Green)
fpSpread1.ActiveSheet.SheetCorner.Rows(0).Border = New
FarPoint.Win.LineBorder(Color.Green)
fpSpread1.ActiveSheet.SheetCorner.HorizontalGridLine = New
FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.None)
fpSpread1.ActiveSheet.SheetCorner.VerticalGridLine = New
FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.None)
fpSpread1.ActiveSheet.SheetCorner.DefaultStyle.VisualStyle =
FarPoint.Win.VisualStyle.Off

```

Using SheetView class

Customizing the Corner Color

You can set the sheet corner style by using the **SheetCornerStyle** ('**SheetCornerStyle Property**' in the **on-line documentation**) property of the **SheetView** ('**SheetView Class**' in the **on-line documentation**) object to specify individual properties of the sheet corner (as in the following example). You can also specify the grid lines around the sheet corner using the **SheetCornerHorizontalGridLine** ('**SheetCornerHorizontalGridLine Property**' in the **on-line documentation**) and **SheetCornerVerticalGridLine** ('**SheetCornerVerticalGridLine Property**' in the **on-line documentation**) properties.

Example

This example code sets the background color to light blue and sets the border as shown in the figure.

C#

```
fpSpread1.ActiveSheet.ColumnHeader.RowCount = 3;
fpSpread1.ActiveSheet.RowHeader.ColumnCount = 3;
fpSpread1.ActiveSheet.SheetCorner.DefaultStyle.Renderer = new
FarPoint.Win.Spread.CellType.CornerRenderer();
fpSpread1.ActiveSheet.SheetCornerStyle.BackColor = Color.LightBlue;
fpSpread1.ActiveSheet.SheetCornerStyle.Border = new
FarPoint.Win.LineBorder(Color.Green, 2);
```

VB

```
fpSpread1.ActiveSheet.ColumnHeader.RowCount = 3
fpSpread1.ActiveSheet.RowHeader.ColumnCount = 3
fpSpread1.ActiveSheet.SheetCorner.DefaultStyle.Renderer = New
FarPoint.Win.Spread.CellType.CornerRenderer
fpSpread1.ActiveSheet.SheetCornerStyle.BackColor = Color.LightBlue
fpSpread1.ActiveSheet.SheetCornerStyle.Border = New
FarPoint.Win.LineBorder(Color.Green, 2)
```

Customizing the Corner Image

You can place a graphic in the sheet corner by setting a background image to a cell type and assigning that cell type to the sheet corner, which is just a cell. Then, specify a particular graphic file for the image (Picture object).

Example

This example code sets the background image of a cell type and assigns that cell type to the sheet corner.

C#

```
FarPoint.Win.Spread.CellType.GeneralCellType gencell = new
FarPoint.Win.Spread.CellType.GeneralCellType();
FarPoint.Win.Picture cornerimage = new
FarPoint.Win.Picture(Image.FromFile("D:\\images\\logocorner.jpg"));
gencell.BackgroundImage = cornerimage;
fpSpread1.ActiveSheet.SheetCornerStyle.CellType = gencell;
```

VB

```
Dim gencell As New FarPoint.Win.Spread.CellType.GeneralCellType
Dim cornerimage As New
FarPoint.Win.Picture(Image.FromFile("D:\\images\\logocorner.jpg"))
gencell.BackgroundImage = cornerimage
FpSpread1.ActiveSheet.SheetCornerStyle.CellType = gencell
```

Text Display in the Sheet Corner

You can add text to the sheet corner cell of the spreadsheet by setting the **Cells ('Cells Property' in the on-line documentation)** property of **SheetCorner ('SheetCorner Class' in the on-line documentation)** class or overriding the **PaintCell** method. The figure shows the results of the example code in this topic.

Text	A	B
1		
2		
3		

For information about setting text using the **SheetCorner** properties, see **General Style of the Sheet Corner**.

Using Code

You can assign text to the sheet corner cell by overriding the painting of the cell and painting it with the specified text.

Example

C#

```
class CornerCell : FarPoint.Win.Spread.CellType.GeneralCellType
{
    public override void PaintCell(Graphics g, Rectangle r,
FarPoint.Win.Spread.Appearance appearance, object value, bool isSelected, bool
isLocked, float zoomFactor)
    {
        base.PaintCell(g, r, appearance, "Text", isSelected, isLocked, zoomFactor);
    }
}
```

```
CornerCell sctextcell = new CornerCell();
fpSpread1.Sheets[0].SheetCornerStyle.CellType = sctextcell;
```

VB

```
Public Class SheetCorner
Inherits FarPoint.Win.Spread.CellType.GeneralCellType
    Public Overrides Sub PaintCell(ByVal g As System.Drawing.Graphics, ByVal r As
System.Drawing.Rectangle, ByVal appearance As FarPoint.Win.Spread.Appearance, ByVal
value As Object, ByVal isSelected As Boolean, ByVal isLocked As Boolean, ByVal
zoomFactor As Single)
        MyBase.PaintCell(g, r, appearance, "Text", isSelected, isLocked, zoomFactor)
    End Sub
End Class
```

```
Dim sctextcell As New SheetCorner()
fpSpread1.Sheets(0).SheetCornerStyle.CellType = sctextcell
```

Table Display in the Sheet Corner

The sheet corner can have a table or grid display with columns and rows of cells, though cells are not editable. You can set the sheet corner to a table (range) display or a single cell display with the **AllowTableCorner** ('**AllowTableCorner Property**' in the on-line documentation) property of the sheet (**SheetView** ('**SheetView Class**' in the on-line documentation) class).

The sheet corner also supports cell spans in the table display.

When you set the **AllowTableCorner** ('**AllowTableCorner Property**' in the on-line documentation) property of the sheet to true, the sheet corner displays as a table with the number of columns equal to the number of row headers of the sheet and the number of rows equal to the number of column headers of the sheet. The following figure illustrates

a table sheet corner display:

			A	B	C
1					
2					
3					

When you set the **AllowTableCorner** ('**AllowTableCorner Property**' in the on-line documentation) property of the sheet to false, the first cell of the sheet corner spans the entire area of the sheet corner, based on the number of columns and rows set for the corner. The following figure illustrates a single-cell sheet corner display:

			A	B	C
1					
2					
3					
4					
5					

Example

The following code sets the sheet corner to appear as a table.

C#

```
fpSpread1.ActiveSheet.AllowTableCorner = true;
fpSpread1.ActiveSheet.SheetCorner.ColumnCount = 4;
fpSpread1.ActiveSheet.SheetCorner.RowCount = 5;
```

VB

```
fpSpread1.ActiveSheet.AllowTableCorner = True
fpSpread1.ActiveSheet.SheetCorner.ColumnCount = 4
fpSpread1.ActiveSheet.SheetCorner.RowCount = 5
```

Example

The following code sets the sheet corner to appear as a single cell.

C#

```
fpSpread1.ActiveSheet.AllowTableCorner = false;
fpSpread1.ActiveSheet.SheetCorner.ColumnCount = 4;
fpSpread1.ActiveSheet.SheetCorner.RowCount = 5;
```

VB

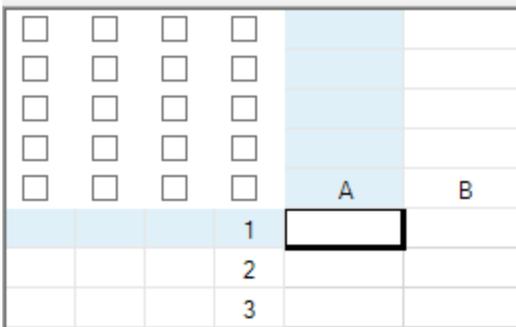
```
fpSpread1.ActiveSheet.AllowTableCorner = False
fpSpread1.ActiveSheet.SheetCorner.ColumnCount = 4
fpSpread1.ActiveSheet.SheetCorner.RowCount = 5
```

Customizable Cell in the Sheet Corner

The single cell or table of cells in the sheet corner can be customized. You can set the cells to any of the cell types that are supported by Spread. You can also customize properties of the **Cell ('Cell Class' in the on-line documentation)** class.

You can change the cell type of the cells in the sheet corner by setting a new cell type for the **SheetCornerStyle ('SheetCornerStyle Property' in the on-line documentation)** property of the sheet (**SheetView ('SheetView Class' in the on-line documentation)** class). When you set the SheetCornerStyle.CellType, all the cells in the sheet corner are changed to that type.

The following examples set the sheet corner to be a check box or a button.

Example

The following code sets the sheet corner to be check box cells by setting the sheet corner style (as shown in the preceding illustration).

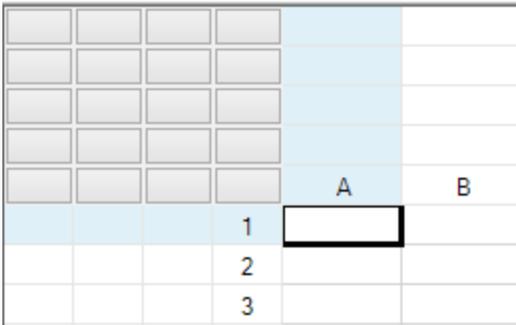
C#

```
fpSpread1.ActiveSheet.AllowTableCorner = true;
fpSpread1.ActiveSheet.SheetCorner.ColumnCount = 4;
fpSpread1.ActiveSheet.SheetCorner.RowCount = 5;
fpSpread1.ActiveSheet.SheetCornerStyle.CellType = new
FarPoint.Win.Spread.CellType.CheckBoxCellType();
```

VB

```
fpSpread1.ActiveSheet.AllowTableCorner = True
fpSpread1.ActiveSheet.SheetCorner.ColumnCount = 4
fpSpread1.ActiveSheet.SheetCorner.RowCount = 5
fpSpread1.ActiveSheet.SheetCornerStyle.CellType = New
FarPoint.Win.Spread.CellType.CheckBoxCellType()
```

Example



The following code sets the sheet corner to be button cells by assigning the button cell type to the cells in the sheet corner (as shown in the preceding illustration).

C#

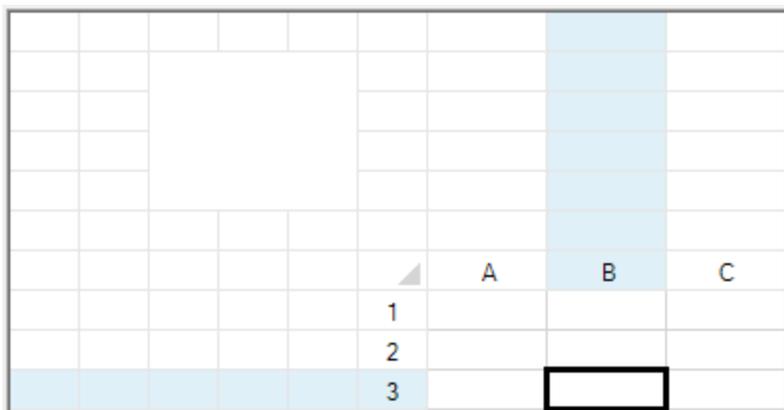
```
fpSpread1.ActiveSheet.AllowTableCorner = true;
fpSpread1.ActiveSheet.SheetCorner.ColumnCount = 4;
fpSpread1.ActiveSheet.SheetCorner.RowCount = 5;
fpSpread1.ActiveSheet.SheetCorner.Cells[2,3].CellType = new
FarPoint.Win.Spread.CellType.ButtonCellType();
```

VB

```
fpSpread1.ActiveSheet.AllowTableCorner = True
fpSpread1.ActiveSheet.SheetCorner.ColumnCount = 4
fpSpread1.ActiveSheet.SheetCorner.RowCount = 5
fpSpread1.ActiveSheet.SheetCorner.Cells(2,3).CellType = New
FarPoint.Win.Spread.CellType.ButtonCellType()
```

Cell Spans in the Sheet Corner

Cell spans are supported in the sheet corner. Any number of rows and columns of cells can be selected and merged to create cell spans. You can set spans of cells in rows or columns or both.



For more information on creating cell spans, refer to **Creating a Span of Cells**.

Example

The following code creates a span of row cells and a span of column cells in the sheet corner.

C#

```
fpSpread1.ActiveSheet.AllowTableCorner = true;
fpSpread1.ActiveSheet.SheetCorner.ColumnCount = 6;
fpSpread1.ActiveSheet.SheetCorner.RowCount = 7;
fpSpread1.ActiveSheet.SheetCorner.Cells[1, 2].ColumnSpan = 3;
fpSpread1.ActiveSheet.SheetCorner.Cells[1, 2].RowSpan = 4;
```

VB

```
fpSpread1.ActiveSheet.AllowTableCorner = True
fpSpread1.ActiveSheet.SheetCorner.ColumnCount = 6
fpSpread1.ActiveSheet.SheetCorner.RowCount = 7
fpSpread1.ActiveSheet.SheetCorner.Cells(1, 2).ColumnSpan = 3
fpSpread1.ActiveSheet.SheetCorner.Cells(1, 2).RowSpan = 4
```

Header Count Synchronization in the Sheet Corner

The number of rows and columns in the sheet corner depend on number of the column header rows and row header columns of the sheet. In fact, these two are synchronized by Spread, so if you change sheet corner size, the sheet header rows and columns are adjusted accordingly and if you change the number of sheet header rows or columns of the sheet, the sheet corner adjusts accordingly.

You can use the Rows and Columns properties of the sheet corner to modify the sheet's row header column count and column header row count. The row count of column headers of the sheet always equals the row count of the sheet corner; the column count of row headers of the sheet always equals the column count of the sheet corner.

Example

The following example illustrates how the number of rows in the sheet corner change when you change the number of column header rows.

C#

```
fpSpread1.ActiveSheet.AllowTableCorner = true;
fpSpread1.ActiveSheet.SheetCorner.ColumnCount = 6;
fpSpread1.ActiveSheet.SheetCorner.RowCount = 7;
//Change rows in sheet corner by change Row count of columns header
fpSpread1.ActiveSheet.ColumnHeader.RowCount = 5;
```

VB

```
fpSpread1.ActiveSheet.AllowTableCorner = True
fpSpread1.ActiveSheet.SheetCorner.ColumnCount = 6
fpSpread1.ActiveSheet.SheetCorner.RowCount = 7
'Change rows in sheet corner by change Row count of columns header
fpSpread1.ActiveSheet.ColumnHeader.RowCount = 5
```

Example

The following example illustrates how the number of column header rows change when you change the number of sheet corner columns.

C#

```
fpSpread1.ActiveSheet.ColumnHeader.RowCount = 5;
fpSpread1.ActiveSheet.SheetCorner.ColumnCount = 3;
```

VB

```
fpSpread1.ActiveSheet.ColumnHeader.RowCount = 5
fpSpread1.ActiveSheet.SheetCorner.ColumnCount = 3
```

Drawing (Rendering) Style

You can customize the corner renderer, which draws the sheet corner.

There are two pre-defined corner renderers in Spread.

The default renderer draws the sheet corner with or without Windows XP style depending on the setting of the system.

The enhanced corner renderer always draws the sheet corner with an appearance similar to Microsoft Excel 2007.

Example

This example lists the methods that are used to create a custom corner renderer.

C#

```
public class MyCornerRenderer : IRenderer {
    /// <summary>
    /// Gets the preferred (maximum needed) size of the cell for the renderer control.
    /// </summary>
    /// <param name="g">Graphics device interface for painting the cell</param>
    /// <param name="size">Preferred or maximum needed size</param>
    /// <param name="appearance">Appearance settings of the renderer control</param>
    /// <param name="value">Object containing the name of the renderer control</param>
    /// <param name="zoomFactor">Numeric value for zoom factor for scaling the display of
    the renderer control</param>
    /// <returns></returns>
    public Size GetPreferredSize(Graphics g, Size size, Appearance appearance, object
value, float zoomFactor)
    {
        ///Your Code add here
    }
    /// <summary>
    /// Paints the corner cell when not in edit mode to the specified graphics interface
    /// with the specified appearance settings.
    /// </summary>
    /// <param name="g">Graphics device interface for painting the corner cell</param>
    /// <param name="r">Location and size of a rectangular region for painting the corner
    cell</param>
    /// <param name="appearance">Appearance settings of the corner cell</param>
    /// <param name="value">Object containing the name of the renderer control of the
    corner cell</param>
    /// <param name="isSelected">Whether the corner cell is selected</param>
    /// <param name="isLocked">Whether the corner cell is locked</param>
    /// <param name="zoomFactor">Numeric value for scaling the display of the corner
    cell</param>
    public virtual void PaintCell(Graphics g, Rectangle r, Appearance appearance, object
value, bool isSelected, bool isLocked, float zoomFactor)
    {
        ///Your Code add here
    }
}
```

```
/// <summary>
/// Paints the corner cell.
/// </summary>
/// <param name="g">Graphics device interface for painting the corner cell</param>
/// <param name="r">Location and size of a rectangular region for painting the corner
cell</param>
/// <param name="backColor">Background color of the corner cell</param>
/// <param name="foreColor">Foreground color of the corner cell</param>
/// <param name="f">Font</param>
/// <param name="horizontalAlignment">Horizontal alignment of corner cell
content</param>
/// <param name="verticalAlignment">Vertical alignment of the corner cell
content</param>
/// <param name="s">String to paint</param>
/// <param name="textOrientation">Orientation of the text</param>
/// <param name="wordWrap">Whether wrap words to multiple lines</param>
/// <param name="hotkeyPrefix">Whether to show hotkey effect</param>
/// <param name="stringTrim">String trimming mode</param>
/// <param name="visualStyles">Visual styles</param>
/// <param name="mouseOver">Whether the mouse is over the corner cell</param>
/// <param name="rightToLeft">Whether to display right to left</param>
/// <param name="zoomFactor">Numeric value for scaling the display of the corner
cell</param>
public virtual void PaintCorner(Graphics g, Rectangle r, Color backColor, Color
foreColor, Font f, HorizontalAlignment horizontalAlignment, VerticalAlignment
verticalAlignment, string s, TextOrientation textOrientation, bool wordWrap,
HotkeyPrefix hotkeyPrefix, StringTrimming stringTrim, VisualStyles visualStyles, bool
mouseOver, bool rightToLeft, float zoomFactor)
{
    ///Your Code add here
}
/// <summary>
/// Determines whether this cell can overflow into an adjacent cell.
/// </summary>
/// <returns></returns>
public bool CanOverflow()
{
    ///Your Code add here
}
/// <summary>
/// Determines whether adjacent cells can overflow into this cell.
/// </summary>
/// <returns></returns>
public bool CanBeOverflown()
{
    ///Your Code add here
}
}
// Assign new corner render to drawing sheet corner:
fpSpread1.ActiveSheet.SheetCornerStyle.Renderer = new MyCornerRenderer();
```

Cells

When you work with cells in the data area of the spreadsheet, you can work with the objects using the shortcut objects in code (**Cell ('Cell Class' in the on-line documentation)** and **Cells ('Cells Class' in the on-line documentation)** classes), or you can work directly with the model. Most developers who are not creating extensive customizations find it easier to work with the shortcut objects.

These tasks relate to working with cells in the data area of the spreadsheet:

- **Working with the Active Cell**
- **Adding Hyperlink in a Cell**
- **Creating a Range of Cells**
- **Managing Data on a Sheet**
 - **Placing and Retrieving Data**
 - **Handling Data Using Sheet Methods**
 - **Handling Data Using Cell Properties**
 - **Moving Data on a Sheet**
 - **Copying Data on a Sheet**
 - **Repeatedly Filling a Range of Cells with Copied Cells**
 - **Swapping Data on a Sheet**
 - **Removing Data from a Sheet**
 - **Remove Duplicates from Range**
 - **Text to Columns**
 - **Creating Data Type for Custom Objects**
- **Displaying Cell Data**
 - **Resizing the Data to Fit the Cell**
 - **Allowing Cell Data to Overflow**
 - **Aligning Cell Contents**
 - **Resizing a Cell to Fit the Data**
- **Creating a Span of Cells**
- **Allowing Cells to Merge Automatically**
- **Adding a Note to a Cell**
- **Adding a Tag to a Cell**
- **Displaying Text Tips in a Cell**
- **Working with Cell Format Strings**
- **Working with Pattern and Gradient Fill Effects**
- **Inserting Cells**
- **Setting Rich Text in a Cell**
- **Adding a Comment to a Cell**
- **Adding Image in a Cell**
- **Formatting a Cell Value**



Note: The word "appearance" is used to mean the general look of the cell, not just the settings in the Appearance class, which contains only a few settings and is used for the appearance of several parts of the interface. Most of the appearance settings for a cell are in the StyleInfo class.

Settings applied to a particular cell override the settings that are set at the column or row level. Refer to **Object Parentage**.

Other cell-level appearance settings are set by the cell type. For more information on settings related to cell types, refer

to **Cell Types**. You can edit properties of the Cells classes in the **Properties** window (in Spread Designer or in Visual Studio .NET). For more information on the **Cells, Columns, and Rows Editor** that is available from the **Properties** window, refer to the explanation of this editor in the **Spread Designer Guide (on-line documentation)**.

Working with the Active Cell

The active cell is the cell that currently receives any user interaction. The active cell is the single cell that has keyboard focus. There is always an active cell. Usually, the active cell displays some indication that it is in focus.

You can specify the active cell programmatically using the **SetActiveCell ('SetActiveCell Method' in the on-line documentation)** method of the **SheetView ('SheetView Class' in the on-line documentation)** class. You can also use the **ActiveCell ('ActiveCell Property' in the on-line documentation)** property to find the active cell coordinates.

The active cell is stored in the **ActiveRowIndex ('ActiveRowIndex Property' in the on-line documentation)** and **ActiveColumnIndex ('ActiveColumnIndex Property' in the on-line documentation)** properties in the SheetView class. The **LeaveCell ('LeaveCell Event' in the on-line documentation)** event is raised any time the active cell changes.

You can change the focus indicator; for more information, refer to **Customizing the Focus Indicator for a Cell**.

The cell selection is one or more cells that have been highlighted by the user or application. At any given time, there may or may not be a cell selection present. The cell selection (if present) is stored in the selection model inside the SheetView class. The **Changed ('Changed Event' in the on-line documentation)** event (in the selection model) is raised any time the cell selection changes.

Spread Windows Forms has two implementations of selection coloring. When the **SelectionMode ('SelectionMode Property' in the on-line documentation)** property is set to SelectionColors, the cell is painted using selection colors (that is, both SelectionBackColor and SelectionForeColor). When the **SelectionMode** property is set to SelectionRenderer (which is the default), the cell is painted using normal coloring and then over painted with the SelectionRenderer. The default SelectionRenderer uses a semi-transparent version of the system's selection color.

In RowMode, which is an operation mode where only rows can be selected, there is an active cell. The active row is painted similar to a selected row.

Spread Windows Forms uses a painting scheme similar to Excel. If the cell is the active cell then the cell is painted using normal coloring and a focus box. If the cell is selected then the cell is painted using selection coloring, or else the cell is painted using normal coloring.

There is a different painting scheme in OpenOffice. If the cell is both the active cell and a selected cell then the cell is painted using selection coloring and a focus box. Spread Windows Forms does not support OpenOffice's painting scheme.

You can change what can be selected by the user. For more information, refer to **Specifying What the User Can Select**. You can also customize how the selection appears. For more information, refer to **Customizing the Selection Appearance**.

Using a Shortcut

You can use **ActiveCell ('ActiveCell Property' in the on-line documentation)** as a shortcut object for the active cell when specifying properties of that cell. Use the **SetActiveCell ('SetActiveCell Method' in the on-line documentation)** to set the active cell.

Example

Set the active cell and do not clear previously selected cells.

C#

```
fpSpread1.ActiveSheet.SetActiveCell(2, 2, false);
```

VB

```
fpSpread1.ActiveSheet.SetActiveCell(2, 2, False)
```

Adding Hyperlink in a Cell

Hyperlinks can be added in cells to access relevant information present at any other location. Spread for WinForms allows you to add hyperlinks which can be used to:

- Access a webpage URL - For example, <https://developer.mescius.com/>
- Open mail box with recipient's email address - For example, spread.sales@mescius.com
- Access a range location in current workbook - For example, Sheet2!A1:B2
- Perform any custom operation - For example, showing a popup instead of navigating to URL.

Add Hyperlinks

You can add hyperlinks to a sheet or range by using the **IWorksheet.Hyperlinks.Add** or **IRange.Hyperlinks.Add** method respectively.

The below example code shows how to insert hyperlinks to access a range in a workbook, a webpage, or an email address.

C#

```
// Hyperlink to range locations
fpSpread1.AsWorkbook().ActiveSheet.Hyperlinks.Add("B3", "", "Sheet1!A1", "Click here to go
to A1", "Goto Cell A1");

// Hyperlink to web URLs
fpSpread1.AsWorkbook().ActiveSheet.Hyperlinks.Add("B5", "http://developer.mescius.com/",
"", "Click here to go to mescius website", "Mescius Website");

// Hyperlink to email
fpSpread1.AsWorkbook().ActiveSheet.Hyperlinks.Add("B7",
"mailto:spread.support@mescius.com?subject=spread.support", "", "Click here to mail for
spread support", "Mail to: Spread Support");
```

VB

```
' Hyperlink to range locations
fpSpread1.AsWorkbook().ActiveSheet.Hyperlinks.Add("B3", "", "Sheet1!A1", "Click here to go
to A1", "Goto Cell A1")

' Hyperlink to web URLs
fpSpread1.AsWorkbook().ActiveSheet.Hyperlinks.Add("B5", "http://developer.mescius.com/", "",
"Click here to go to mescius website", "Mescius Website")
' Hyperlink to email

fpSpread1.AsWorkbook().ActiveSheet.Hyperlinks.Add("B7", "mailto:spread.support@mescius.com?
subject=spread.support", "", "Click here to mail for spread support", "Mail to: Spread
Support")
```

 **Note:** If another hyperlink is set to a cell containing a hyperlink, the new hyperlink will replace the old one rather than merging it.

The below example code shows that you can perform a custom action after disabling the default behavior of hyperlink.

C#

```
// Hyperlink to custom action
fpSpread1.HyperLinkClicked += FpSpread1_HyperLinkClicked;
private void FpSpread1_HyperLinkClicked(object sender, HyperLinkClickedEventArgs e)
{
    // Set e.Link to null to skip the default behavior and customize accordingly
    e.Link = null;
}

```

VB

```
'Hyperlink to custom action
fpSpread1.HyperLinkClicked += AddressOf FpSpread1_HyperLinkClicked
Private Sub FpSpread1_HyperLinkClicked(ByVal sender As Object, ByVal e As
HyperLinkClickedEventArgs)
    e.Link = Nothing
End Sub

```

The below example code shows how to display the built-in 'Insert Hyperlink' dialog.

C#

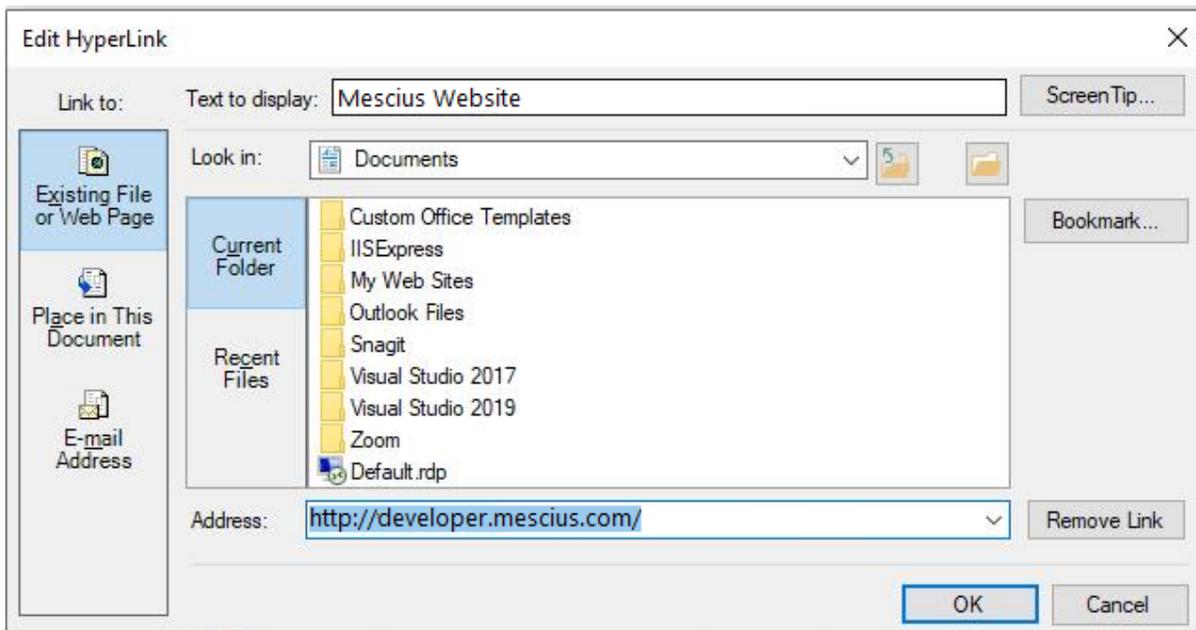
```
// Show HyperlinkForm
HyperlinkForm form = new HyperlinkForm(fpSpread1.AsWorkbook().ActiveSheet);
form.ShowDialog();

```

VB

```
' Show HyperlinkForm
Dim form As HyperlinkForm = New HyperlinkForm(fpSpread1.AsWorkbook().ActiveSheet)
form.ShowDialog()

```



Create Hyperlinks Automatically

You can also generate a hyperlink automatically by entering a link like string value and setting the **AutoCreateHyperlink** property to true.

- If a link like value starts with www, the hyperlink will consider it as a url
- If a link like value matches the email link but it doesn't start with mailto:, the hyperlink will consider it as a url
- If a link like value matches the workbook location type which starts with 'spread://', the hyperlink will set the value as a url

C#

```
fpSpread1.Features.AutoCreateHyperlink = true;
//input link like value in any cell and it will be automatically turn into hyperlink
```

VB

```
fpSpread1.Features.AutoCreateHyperlink = True
'Input link like value in any cell and it will be automatically turn into hyperlink
```

Modify Hyperlinks

You can edit a hyperlink in a sheet or range by changing the settings in **IWorksheet.Hyperlinks[index]** or **IRange.Hyperlinks[index]** object respectively.

The below example code shows how to modify a hyperlink in Cell B3.

C#

```
// Edit link
fpSpread1.AsWorkbook().ActiveSheet.Hyperlinks.Add("B3", "", "Sheet1!A1", "Click here to go
to A1", "Goto Cell A1");
GrapeCity.Spreadsheet.IHyperlink hpl =
fpSpread1.AsWorkbook().ActiveSheet.Cells["B3"].Hyperlinks[0];
hpl.SubAddress = "Sheet1!A2";
hpl.TextToDisplay = "Goto Cell A2";
hpl.ScreenTip = "Click here to go to A2";
```

VB

```
' Edit link
fpSpread1.AsWorkbook().ActiveSheet.Hyperlinks.Add("B3", "", "Sheet1!A1", "Click here to go
to A1", "Goto Cell A1")
Dim hpl As GrapeCity.Spreadsheet.IHyperlink =
fpSpread1.AsWorkbook().ActiveSheet.Cells("B3").Hyperlinks(0)
hpl.SubAddress = "Sheet1!A2"
hpl.TextToDisplay = "Goto Cell A2"
hpl.ScreenTip = "Click here to go to A2"
```

Delete Hyperlinks

You can delete a hyperlink while retaining the cell text by using **IWorksheet.Hyperlinks[index].Delete** or **IRange.Hyperlinks[index].Delete** method. All the links in a sheet or range can also be deleted at once by using **IWorksheet.Hyperlinks.Delete** or **IRange.Hyperlinks.Delete** method respectively.

The below example shows how to delete a hyperlink in cell B3.

C#

```
// Delete link
fpSpread1.AsWorkbook().ActiveSheet.Hyperlinks.Add("B3", "", "Sheet1!A1", "Click here to go
to A1", "Goto Cell A1");
fpSpread1.AsWorkbook().ActiveSheet.Cells["B3"].Hyperlinks[0].Delete();
```

VB

```
' Delete link
fpSpread1.AsWorkbook().ActiveSheet.Hyperlinks.Add("B3", "", "Sheet1!A1", "Click here to go
to A1", "Goto Cell A1")
fpSpread1.AsWorkbook().ActiveSheet.Cells("B3").Hyperlinks(0).Delete()
```

Set Hyperlink Style

The built-in style of hyperlink is added along with it, by default. When you click on a link, it is painted with the text color of "Followed Hyperlink" style. However, you can customize the hyperlink style by changing the settings of "Hyperlink" and "Followed Hyperlink" built-in styles.

The below example code shows how to change the built-in hyperlink styles.

C#

```
// Change setting of "Hyperlink" and "Followed Hyperlink" build-in styles
fpSpread1.AsWorkbook().Styles[GrapeCity.Spreadsheet.BuiltInStyle.Hyperlink].Font.Color =
GrapeCity.Spreadsheet.Color.FromKnownColor(GrapeCity.Core.KnownColor.Red);
fpSpread1.AsWorkbook().Styles[GrapeCity.Spreadsheet.BuiltInStyle.FollowedHyperlink].Font.Color
= GrapeCity.Spreadsheet.Color.FromKnownColor(GrapeCity.Core.KnownColor.Cyan);
```

VB

```
' Change setting of "Hyperlink" and "Followed Hyperlink" build-in styles
fpSpread1.AsWorkbook().Styles(GrapeCity.Spreadsheet.BuiltInStyle.Hyperlink).Font.Color =
GrapeCity.Spreadsheet.Color.FromKnownColor(GrapeCity.Core.KnownColor.Red)

fpSpread1.AsWorkbook().Styles(GrapeCity.Spreadsheet.BuiltInStyle.FollowedHyperlink).Font.Color
= GrapeCity.Spreadsheet.Color.FromKnownColor(GrapeCity.Core.KnownColor.Cyan)
```

UI Behavior

The Hyperlink displays following behavior when UI interaction is performed:

- The painting area of a cell containing hyperlink is used for navigating to the target location (by displaying Hand cursor).
- The cross cursor is shown for other areas, if all of the below are set:
 - Horizontal alignment is either Left, Right, Center
 - Vertical alignment is either Top, Bottom, Center
 - TextRotation is 0
 - WrapText is False
- The tooltip is shown when mouse hovers over hyperlink cell text and hides when mouse leaves the hyperlink cell text.
- To select the cell without navigating to the target location, click into the cell and hold for a moment until the cursor shows as a cross cursor.
- Hyperlink supports formulas. However, the visited state like actual hyperlink is not supported.
- Hyperlink supports overflow but the overflowed content cannot be clicked as link.
- When you type a formatted URL directly in a cell and press ENTER, the cell edit action is finished. The link is automatically created by a separate create link action. Now if you use undo command or CTRL+Z, only the link is removed from the cell, but the text of URL is retained. If the undo command or CTRL+Z is executed again, then the text URL is removed from the cell.

Limitations

- Hyperlinks can only be copied when RichClipboard is set to true.
- Hyperlink with DragFill is not supported.

Creating a Range of Cells

You can create a range of cells to allow you to define properties and behaviors for those cells. A range may be any set of cells.

To fill ranges using drag-and-drop or drag-and-fill actions, refer to **Using Drag Operations to Fill Cells**.

Using Code

1. Define a range of cells using the **Cell ('Cell Class' in the on-line documentation)** object.
2. Set properties for the range such as the **Note ('Note Property' in the on-line documentation)** property.

Example

This example code sets the **Note ('Note Property' in the on-line documentation)** property for a range of **Cell ('Cell Class' in the on-line documentation)** objects.

C#

```
FarPoint.Win.Spread.Cell range1;  
range1 = fpSpread1.ActiveSheet.Cells[1, 1, 3, 3];  
range1.Value = "Value Here";  
range1.Note = "This is the note that describes the value.";
```

VB

```
Dim range1 As FarPoint.Win.Spread.Cell  
range1 = fpSpread1.ActiveSheet.Cells(1, 1, 3, 3)  
range1.Value = "Value Here"  
range1.Note = "This is the note that describes the value."
```

Using the Spread Designer

In the **Cell, Column, or Row** editor and in the Spread Designer, select the cells that you want to be in the range. Properties you set are then applied to those cells.

Working with Cell Format Strings

Spread for WinForms enables you to format the data displayed in a cell. You can format the values in cells as fractions, currency, percentages, decimals, scientific data, date, time, and so on. The Spread API uses [Excel's number formats](#) to display various cell number formats.

The following formats are supported by Spread for WinForms:

- **General Format**
- **Fraction Format**
- **Number Format**
- **Percentage Format**
- **Scientific Format**
- **Currency Format**
- **Time Format**

- **Date Format**
- **DateTime Format**
- **Accounting Format**
- **Color Format**
- **Text Format**
- **Special Format**

General Format

This format does not have any specific number format. Numbers that are formatted with the General format are displayed just the way they are typed in the cell.

However, this format automatically rounds the numbers with decimals if it exceeds a specific character limit in the cell or if the cell is not wide enough to show the entire number.

One of the following scenarios can be observed when inputting long numbers with General format:

- **With Decimals:** If a number is longer than 11 characters including the decimal place, then the decimals are rounded to show a maximum of 11 characters.
For example, 123456.7891234 is rounded off to 123456.7891
- **Without Decimals:** If a number is longer than 11 characters and there are no decimals, the number is changed to Scientific format.
For example, 123451234512 is rounded off to 1.23451E+11

Users can also observe the following behavior when reducing column widths of cells with General format applied:

- Decimals are rounded to show only the number that will fit the column.
- The number will be rounded to an integer if there is no room for decimals.
- The number will change to Scientific format if the rounded integer does not fit.
- The cell will display number signs if the number can't be displayed in Scientific format.

 **Note:** This behavior can be disabled by using formats other than "General" like Number and Accounting.

Fraction Format

This format displays numbers as fractions like mixed number fractions or other types of fractions in two different layouts - # ?/? and # ??/?. For this format, Spread for WinForms allows you to select either number of decimal places to display the result in or the nearest place to round off the result. In case you enter data in improper fractions, it will be auto-calculated to the integer part and proper fraction. To enter a negative fraction value, use the (-) sign before entering the mixed number or input the mixed number into parenthesis "()".

Code Format	Input Value	Display Value
# ?/?	8.333	8 1/3
# ??/??	0.846	11/13

The following code example shows how to set fraction format to a cell containing a value.

C#

```
var worksheet = fpSpread1_Sheet1.AsWorksheet();

worksheet.SetValue(0, 0, 8.333);
worksheet.Cells["A1"].NumberFormat = "# ?/?";
```

Visual Basic

```
Dim worksheet As GrapeCity.Spreadsheet.IWorksheet = FpSpread1_Sheet1.AsWorksheet()

worksheet.SetValue(0, 0, 8.333)
worksheet.Cells("A1").NumberFormat = "# ?/?"
```

Number Format

This format displays data in Number format in two different layouts - #,##0 and #,##0.00. You can display data with thousand separator and decimal places by checking the "Use 1000 Separator (,)" option and mentioning the number of decimal places to display. To enter a negative number value, use the (-) sign before or input the number into parenthesis "()".

Code Format	Input Value	Display Value
#,##0	1231	1,231
#,##0.00	4561	4,561.00

The following code example shows how to set the number format to a cell containing a value.

C#

```
var worksheet = fpSpread1_Sheet1.AsWorksheet();

worksheet.SetValue(0, 0, 1234);
worksheet.Cells["A1"].NumberFormat = "#,##0";
```

Visual Basic

```
Dim worksheet As GrapeCity.Spreadsheet.IWorksheet = FpSpread1_Sheet1.AsWorksheet()

worksheet.SetValue(0, 0, 1234)
worksheet.Cells("A1").NumberFormat = "#,##0"
```

Percentage Format

This format displays data in Percentage format in two different layouts - 0% and 0.00%. This format multiplies the cell value by 100 and displays the result with a % symbol. You can set the number of decimal places to be displayed in the result. To enter a negative number value, use the (-) sign before the number or input the number into parenthesis "()".

Code Format	Input Value	Display Value
0%	0.05	5%
0.00%	0.8	80.00%

The following code example shows how to set percentage format to a cell containing a value.

C#

```
var worksheet = fpSpread1_Sheet1.AsWorksheet();

worksheet.SetValue(0, 0, 0.05);
worksheet.Cells["A1"].NumberFormat = "0%";
```

Visual Basic

```
Dim worksheet As GrapeCity.Spreadsheet.IWorksheet = FpSpread1_Sheet1.AsWorksheet()

worksheet.SetValue(0, 0, 0.05)
```

```
worksheet.Cells("A1").NumberFormat = "0%"
```

Scientific Format

This format displays data in Scientific format in only one layout - 0.00E+00. The scientific format displays numbers in an exponential notation by replacing parts of the number with E+n; where “E” stands for the exponent that multiplies the preceding number by 10 to the nth power. To display a number in scientific (exponential) format, you need to enter numbers in the form "mEn where coefficient m refers to any real number, while the exponent n is an integer that corresponds to the number of places that the decimal point was moved. You can replace the character E by E+; e by e+; and E- by e-. To enter a negative number value, use the (-) sign before the number or input the number into parenthesis "()".

Code Format	Input Value	Display Value
0.00E+00	12345678901	1.23E+10

The following code example shows how to set scientific format to a cell containing a value.

C#

```
var worksheet = fpSpread1_Sheet1.AsWorksheet();

worksheet.SetValue(0, 0, 12345678901);
worksheet.Cells["A1"].NumberFormat = "0.00E+00";
```

Visual Basic

```
Dim worksheet As GrapeCity.Spreadsheet.IWorksheet = FpSpread1_Sheet1.AsWorksheet()

worksheet.SetValue(0, 0, 12345678901)
worksheet.Cells("A1").NumberFormat = "0.00E+00"
```

Currency Format

This format displays data in Currency format in two different layouts - \$#,##0_);[Red](\$#,##0) and \$#,##0.00_);[Red](\$#,##0.00). The position of the currency symbol is based on the default language of the MS Office programs. You need to enter the correct symbol position of the currency to format cells with the currency format. To enter a negative fraction value, use the (-) sign before entering the mixed number or input the mixed number into parenthesis "()".

Code Format	Input Value	Display Value
\$#,##0_);[Red](\$#,##0)	-1235	(\$1,235)
\$#,##0.00_);[Red](\$#,##0.00)	4597	\$4,597.00

The following code example shows how to set currency format to a cell containing a value.

C#

```
var worksheet = fpSpread1_Sheet1.AsWorksheet();

worksheet.SetValue(0, 0, -1235);
worksheet.Cells["A1"].NumberFormat = "$#,##0_);[Red]($#,##0)";
```

Visual Basic

```
Dim worksheet As GrapeCity.Spreadsheet.IWorksheet = FpSpread1_Sheet1.AsWorksheet()

worksheet.SetValue(0, 0, -1235)
```

```
worksheet.Cells("A1").NumberFormat = "$#,##0_);[Red]($#,##0)"
```

Time Format

This format displays data in Time format in six different layouts - h:mm tt, h:mm:ss tt, H:mm, H:mm:ss, [h]:mm:ss and mm:ss.0. Only integer values are accepted for the hours, minutes, and seconds. To enter a time value in either Japanese, Chinese or Korean language, combine the value with the time text like 時 and 分.

Code Format	Input Value	Display Value
h:mm tt	21:05	9:05 tt
h:mm:ss tt	7:49:15	7:49:15 tt
h:mm	17:25	17:25
h:mm:ss	3:19	3:19:00
[h]:mm:ss	236:34	20:34:00
mm:ss.0	5:05	05:00.0

The following code example shows how to set the time format to a cell containing a value.

C#

```
var worksheet = fpSpread1.ActiveSheet.AsWorksheet();
worksheet.SetValue(0, 0, DateTime.Now);
worksheet.Cells["A1"].NumberFormat = "h:mm:ss";
```

Visual Basic

```
Dim worksheet As GrapeCity.Spreadsheet.IWorksheet = FpSpread1_Sheet1.AsWorksheet()

worksheet.SetValue(0, 0, DateTime.Now)
worksheet.Cells("A1").NumberFormat = "h:mm:ss"
```

Date Format

This format displays data in Date format in four different layouts - m/d/yyyy, d-mmm-yy, d-mmm, and mmm-yy. To use this format, you need to enter values in at least one combination - date and month, date and year, or complete value of date, month, and year. When you enter values only for date and month, the value of the year is automatically set to the current year. As a separating character, Spread for WinForms supports both (-) sign and (/) sign. You can use these separating characters in any combination.

The data values for the date, month, and year can be entered based on the following table:

Months	m	1-12
Months	mm	01-12
Months	mmm	Jan-Dec
Months	mmmm	January-December
Days	d	1-31
Days	dd	01-31
Years	yy	00-99
Years	yyyy	1900-9999

If you enter any year from 0-29, the cell value is automatically formatted to 2000-2029. Similarly, if you enter any year

from 30-99, the value is formatted to 1930-1999. You can enter text value for a month in both upper case and low case. To enter date value in either Japanese, Chinese or Korean language, combine the value with the date text like 時 and 分.

Spread for WinForms also supports the use of [DBNumX] modifier to display data in East Asia numeric format. For example, in Japanese locale, if you use "[DBNum1][\$-411]d/mm/yyyy" format with the value 43413, the formatted text will be "九/十一/二〇一八".

Code Format	Input Value	Display Value
yyyy/m/d	2019/02/20	2019/2/20
yyyy"年"m"月"	2019/02/20	2019年2月
m"月"d"日"	2019/02/20	2月20日

The following code example shows how to set the date format to a cell containing a value.

C#

```
var worksheet = fpSpread1_Sheet1.AsWorksheet();

worksheet.Cells["A1"].Text = "1/1/2021";
worksheet.Cells["A1"].NumberFormat = "dd-mm-yyyy";
```

Visual Basic

```
Dim worksheet As GrapeCity.Spreadsheet.IWorksheet = FpSpread1_Sheet1.AsWorksheet()

worksheet.Cells("A1").Text = "1/1/2021"
worksheet.Cells("A1").NumberFormat = "dd-mm-yyyy"
```

DateTime Format

This format displays data in DateTime format, with only one layout - m/d/yyyy h:mm. When you enter the date value combined with the time value, the data is automatically formatted in DateTime format (m/d/yyyy h:mm). The value of a date can be placed before or after the time value. To enter a value for time with the format "hour:" or "hour:minute", the date value needs to be combined with day, month, and year. If the data only consists of a date value, it will be formatted as a Date format.

Code Format	Input Value	Display Value
yyyy/m/d h:mm	2019/02/20 12:30:00	2019/2/20 12:30

The following code example shows how to set the datetime format to a cell containing a value.

C#

```
var worksheet = fpSpread1_Sheet1.AsWorksheet();

worksheet.Cells["A1"].Text = "2019/02/20 12:30:00";
worksheet.Cells["A1"].NumberFormat = "yyyy/m/d h:mm";
```

Visual Basic

```
Dim worksheet As GrapeCity.Spreadsheet.IWorksheet = FpSpread1_Sheet1.AsWorksheet()

worksheet.Cells("A1").Text = "2019/02/20 12:30:00"
worksheet.Cells("A1").NumberFormat = "yyyy/m/d h:mm"
```

Accounting Format

This format displays data in Accounting format in two different layouts - `*$ #,##0` and `*$ #,##0.00`. In this format, the currency symbols and decimal points are aligned in a column. This is implemented using an asterisk (*) symbol to denote repeat characters. By default, code formats use " " (asterisk with a space after) to enter spaces in between the currency symbol and the value, but you can replace " " (space) with any other character.

Code Format	Input Value	Display Value
<code>*\$ #,##0</code>	-1513	-\$ 1,513
<code>*\$ #,##0.00</code>	2583	\$ 2,583.00

The following code example shows how to set the accounting format to a cell containing a value.

C#

```
var worksheet = fpSpread1_Sheet1.AsWorksheet();

worksheet.SetValue(0, 0, -1513);
worksheet.Cells["A1"].NumberFormat = "$* #,##0";
```

Visual Basic

```
Dim worksheet As GrapeCity.Spreadsheet.IWorksheet = FpSpread1_Sheet1.AsWorksheet()

worksheet.SetValue(0, 0, -1513)
worksheet.Cells("A1").NumberFormat = "$* #,##0"
```

Color Format

The Color Format displays data based on the color criteria that affects the `foreColor`.

	A	B	C	D	E	F
1	100.0	200.0	300.0	400.0	500.0	600.0
2						

It supports color string names as well as color indices:

- 8 Named colors: [Black] [Blue] [Cyan] [Green] [Magenta] [Red] [White] [Yellow]
- 56 Indexed colors: [Color1][Color2]...[Color56]

While specifying color formats in code, the name of the color comes first in the code and is enclosed in square brackets. Also, to show that the number format will be applied only if a specified condition is met, the criteria is enclosed in square brackets.

The condition will consist of a comparison operator and a value. For instance, the following number format will display numbers that are less than 50 in Yellow font and numbers that are greater than or equal to 50 in Magenta font.

```
[Yellow] [<50]; [Magenta] [>=50]
```

The following code example shows how to set color formatting by index to change the color of the cells according to the value range.

C#

```
var worksheet = fpSpread1_Sheet1.AsWorksheet();

worksheet.SetValue(0, 0, 100);
worksheet.SetValue(0, 1, 200);
worksheet.SetValue(0, 2, 300);
worksheet.SetValue(0, 3, 400);
```

```
worksheet.SetValue(0, 4, 500);  
worksheet.SetValue(0, 5, 600);
```

```
worksheet.Range("A1:F1").NumberFormat = "[color44]<300]0.0;[color3]>400]0.0;  
[color45]0.0";
```

Visual Basic

```
Dim worksheet As GrapeCity.Spreadsheet.IWorksheet = FpSpread1_Sheet1.AsWorksheet()
```

```
worksheet.SetValue(0, 0, 100)  
worksheet.SetValue(0, 1, 200)  
worksheet.SetValue(0, 2, 300)  
worksheet.SetValue(0, 3, 400)  
worksheet.SetValue(0, 4, 500)  
worksheet.SetValue(0, 5, 600)
```

```
worksheet.Range("A1:F1").NumberFormat = "[color44]<300]0.0;[color3]>400]0.0;  
[color45]0.0"
```

Text Format

This format displays data in Text format. In this format, the value of the cells is treated as text even when a number is entered. The result is displayed in the cell exactly as the data is entered.

Special Format

This format displays data in Special format in four different layouts - Zip Code, Zip Code + 4, Phone Number, and Social Security Number.

Managing Data on a Sheet

You can work with data in the cells in the data area of the spreadsheet in a number of ways:

- **Placing and Retrieving Data**
 - **Handling Data Using Sheet Methods**
 - **Handling Data Using Cell Properties**
- **Moving Data on a Sheet**
- **Copying Data on a Sheet**
- **Repeatedly Filling a Range of Cells with Copied Cells**
- **Swapping Data on a Sheet**
- **Removing Data from a Sheet**
- **Remove Duplicates from Range**
- **Text to Columns**
- **Creating Data Type for Custom Objects**

Placing and Retrieving Data

You can place (set) data in cells using a variety of methods and retrieve (get) the data using a complimentary set of methods. For more information refer to:

- **Handling Data Using Sheet Methods**
- **Handling Data Using Cell Properties**

For information on the effects of cell types on how data is displayed and managed, refer to **Understanding How Cell Types Work**.

Handling Data Using Sheet Methods

You can place data in cells as formatted or unformatted strings or as data objects. The best way to place data in cells depends on whether you want to add string data or data objects, and if you want to add data to an individual cell or to a range of cells.

If you are working with data provided by a user in a text box, for example, you probably want to add the data as string data that is parsed by the Spread component. If you are adding several values and want to add them directly to the data model, you can add them as objects.

The following table summarizes the ways you can add data using methods at the sheet level.

Data Description	How Many Cells	Method
As a string with formatting (for example "\$1,234.56")	Individual cell	GetText ('GetText Method' in the on-line documentation) SetText ('SetText Method' in the on-line documentation)
	Range of cells	GetClip ('GetClip Method' in the on-line documentation) SetClip ('SetClip Method' in the on-line documentation)
As a string without formatting (for example "1234.45")	Individual cell	GetValue ('GetValue Method' in the on-line documentation) SetValue ('SetValue Method' in the on-line documentation)
	Range of cells	GetClipValue ('GetClipValue Method' in the on-line documentation) SetClipValue ('SetClipValue Method' in the on-line documentation)
As a data object with formatting	Range of cells	GetArray ('GetArray Method' in the on-line documentation) SetArray ('SetArray Method' in the on-line documentation)

When you work with formatted data, the data is parsed by the cell type formatted for that cell and placed in the data model. When you work with unformatted data, the data goes directly into the data model. If you add data to the sheet that is placed directly into the data model, you might want to parse the data because the component does not do so. To understand the effect that the cell type has on this data, refer to the summary in **Understanding How Cell Types Display and Format Data**.

For detailed information about how to provide the data for each cell type, see the member topics in the **FarPoint.Win.Spread.CellType ('FarPoint.Win.Spread.CellType Namespace' in the on-line documentation)** namespace.

To add a large amount of information to the component, consider creating and opening existing files, such as text files or

Excel-formatted files, as explained in **Opening Existing Files**.

You can also return data by saving the data or the data and formatting to a text file, Excel-formatted file, or Spread XML file. For instructions for saving data to these file types, see **Saving Data to a File**.

Using the Properties Window

To add data to a cell, follow these instructions. You cannot add data to a range of cells unless you want to add the same data to all the cells in the range you select.

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. Click the sheet for which you want to add data.
5. Select the **Cells** property and then click the button to display the **Cell, Column, and Row Editor**.
6. Select the cell or range of cells to which you want to add data.
7. In the property list, set the **Text** property.
8. Click **OK** to close the **Cell, Column, and Row Editor**.
9. Click **OK** to close the **SheetView Collection Editor**.

Using a Shortcut

- To get or set data to a cell using code,
 - Place formatted string data using the **Sheets SetText ('SetText Method' in the on-line documentation)** method or retrieve data using the **GetText ('GetText Method' in the on-line documentation)** method.
 - Place data as objects directly into the data model using the **Sheets SetValue ('SetValue Method' in the on-line documentation)** method or retrieve the data directly using the **GetValue ('GetValue Method' in the on-line documentation)** method.
- To get or set data to a range of cells,
 - Place formatted string data using the **Sheets SetClip ('SetClip Method' in the on-line documentation)** method or retrieve the data using the **GetClip ('GetClip Method' in the on-line documentation)** method.
 - Place unformatted string data using the **Sheets SetClipValue ('SetClipValue Method' in the on-line documentation)** method or retrieve data using the **GetClipValue ('GetClipValue Method' in the on-line documentation)** method.
 - Add data as objects directly into the data model using the **Sheets SetArray ('SetArray Method' in the on-line documentation)** method or retrieve the data using the **GetArray ('GetArray Method' in the on-line documentation)** method.

Example

This example code adds formatted data to a range of cells.

C#

```
// Add data to cells A1 through C3.  
fpSpread1.Sheets[0].SetClip(0, 0, 3,  
3, "Sunday\tMonday\tTuesday\r\nWednesday\tThursday\tFriday\r\nSaturday\tSunday\tMonday");
```

VB

```
' Add data to cells A1 through C3.  
fpSpread1.Sheets(0).SetClip(0, 0, 3, 3, "Sunday" + vbTab + "Monday" + vbTab + "Tuesday"  
+ vbCrLf + "Wednesday" + vbTab + "Thursday" + vbTab + "Friday" + vbCrLf + "Saturday" +
```

```
vbTab + "Sunday" + vbTab + "Monday")
```

Using Code

- To add data to a cell using code,
 - Add formatted string data by calling the **SheetView** object **SetText** ('SetText Method' in the on-line documentation) method or by calling the **Cell** object **Text** ('Text Property' in the on-line documentation) property.
 - Add data as objects directly into the data model by calling the **SheetView** object **SetValue** ('SetValue Method' in the on-line documentation) method or by calling the **Cell** object **Value** ('Value Property' in the on-line documentation) property.
- To add data to a range of cells,
 - Add formatted string data by calling the **SheetView** object **SetClip** ('SetClip Method' in the on-line documentation) method.
 - Add unformatted string data by calling the **SheetView** object **SetClipValue** ('SetClipValue Method' in the on-line documentation) method.
 - Add data as objects directly into the data model by calling the **SheetView** object **SetArray** ('SetArray Method' in the on-line documentation) method.

Example

This example code adds formatted data to a range of cells.

C#

```
// Create a new SheetView object.
FarPoint.Win.Spread.SheetView newsheet=new FarPoint.Win.Spread.SheetView();
// Add data to cells A1 through C3.
newsheet.SetClip(0, 0, 3, 3, "Sunday\tMonday\tTuesday\r\nWednesday\tThursday\tFriday
\r\nSaturday\tSunday\tMonday");
// Assign the SheetView object to be the first sheet.
fpSpread1.Sheets[0] = newsheet;
```

VB

```
' Create a new SheetView object.
Dim newsheet As New FarPoint.Win.Spread.SheetView()
' Add data to cells A1 through C3.
newsheet.SetClip(0, 0, 3, 3, "Sunday" + vbTab + "Monday" + vbTab + "Tuesday" + vbCrLf +
"Wednesday" + vbTab + "Thursday" + vbTab + "Friday" + vbCrLf + "Saturday" + vbTab +
"Sunday" + vbTab + "Monday")
' Assign the SheetView object to be the first sheet.
fpSpread1.Sheets(0) = newsheet
```

Handling Data Using Cell Properties

The following table summarizes the ways you can get or set data in cells using the properties of the cell.

Data Description

As a string with formatting (for example "\$1,234.56")

As a string without formatting (for example "1234.45")

Cell Property

Text ('Text Property' in the on-line documentation)

Value ('Value Property' in the on-line documentation)

There is no limitation on the data types of values that can be stored in cells. Cell values are assigned and retrieved using the generic Object data type. Primitive data types (for example, bool, int, double, etc.) are assigned and retrieved using boxed primitives.

The C# and Visual Basic .NET languages automatically box primitives (that is, convert primitive to object) for you as illustrated in this code.

Using Code

Use the **Value ('Value Property' in the on-line documentation)** property or **SetValue ('SetValue Method' in the on-line documentation)** method to add data to a cell.

Example

This example adds data to cells.

C#

```
fpSpread1.Sheets[0].Cells[0, 3].Value = 123;  
fpSpread1.Sheets[0].SetValue(0, 6, "abc");
```

VB

```
fpSpread1.Sheets(0).Cells(0, 3).Value = 123  
fpSpread1.Sheets(0).SetValue(0, 6, "abc")
```

Example

You need to manually unbox primitives (that is, convert object to primitive) by using a cast:

C#

```
int i = (int)spread.Sheets[0].Cells[0, 3].Value;  
string s = (string)spread.Sheets[0].GetValue(0, 6);
```

VB

```
Dim i As Integer = CInt(spread.Sheets(0).Cells(0, 3).Value)  
Dim s As String = CStr(spread.Sheets(0).GetValue(0, 6))
```

 **Note:** Empty cells return a null value (Nothing in VB) that causes the cast to fail. If there is a possibility that a cell is empty or contains a value of unknown data type then your code should check the data type prior to performing the cast or should provide an exception handler to catch the exception thrown by the failed cast.

Moving Data on a Sheet

You can move data from one cell or range of cells to another using the **Move ('Move Method' in the on-line documentation)** methods for the sheet.

When you move data from one cell (or range of cells) to another, the data from the origination cell (or range of cells) replaces the data in the destination cell (or cells). If the operation moves a range of cells to an overlapping location, the values of all the cells of the range are replaced with the values of the cells in the moved range.

You can specify whether formulas are automatically updated when cells or ranges of cells are moved. For more information on automatic recalculation, refer to **Recalculating and Updating Formulas Automatically**.

To move data 3 rows up the sheet and 5 rows down, you would need to insert blank rows where you want to move the rows to.

To move 3 rows up and 5 rows down, copy the five rows temporarily then move the 3 rows up to their positions and then assign the five copied rows to the correct position.

Example

This example moves data in the sheet and inserts blank rows.

C#

```
fpSpread1.Sheets[0].SetText(6, 0, "test");
var with1 = (FarPoint.Win.Spread.Model.DefaultSheetDataModel) fpSpread1.Sheets[0].Models.Data;
FarPoint.Win.Spread.Model.DefaultSheetDataModel dm = new
FarPoint.Win.Spread.Model.DefaultSheetDataModel(5, with1.ColumnCount);
dm.SetArray(0, 0,
((FarPoint.Win.Spread.Model.DefaultSheetDataModel) fpSpread1.Sheets[0].Models.Data).GetArray(0,
0, 5, 5));
with1.RemoveRows(0, 5);
with1.AddRows(0, 3);
with1.Move(with1.RowCount - 4, 0, 0, 0, 3, with1.ColumnCount);
with1.RemoveRows(with1.RowCount - 4, 3);
with1.AddRows(with1.RowCount, 5);
with1.SetArray(with1.RowCount - 6, 0, dm.GetArray(0, 0, 5, with1.ColumnCount));
```

VB

```
fpSpread1.Sheets(0).SetText(6, 0, "test")
With CType(fpSpread1.Sheets(0).Models.Data, FarPoint.Win.Spread.Model.DefaultSheetDataModel)
Dim dm As New FarPoint.Win.Spread.Model.DefaultSheetDataModel(5, .ColumnCount)
dm.SetArray(0, 0, CType(fpSpread1.Sheets(0).Models.Data,
FarPoint.Win.Spread.Model.DefaultSheetDataModel).GetArray(0, 0, 5, 5))
.RemoveRows(0, 5)
.AddRows(0, 3)
.Move(.RowCount - 4, 0, 0, 0, 3, .ColumnCount)
.RemoveRows(.RowCount - 4, 3)
.AddRows(.RowCount, 5)
.SetArray(.RowCount - 6, 0, dm.GetArray(0, 0, 5, .ColumnCount))
End With
```

Copying Data on a Sheet

You can copy data to and from cells using the **Copy ('Copy Method' in the on-line documentation)** methods for the sheet.

When you copy data to a cell (or range of cells), the data replaces the data in the destination cell (or cells). If the operation copies a range of cells and pastes them to an overlapping location, the values of all the cells you are pasting are replaced with the values of the cells in the copied range.

You can specify whether formulas are automatically updated when cells or ranges of cells are copied. For more information on automatic recalculation, refer to **Recalculating and Updating Formulas Automatically**.

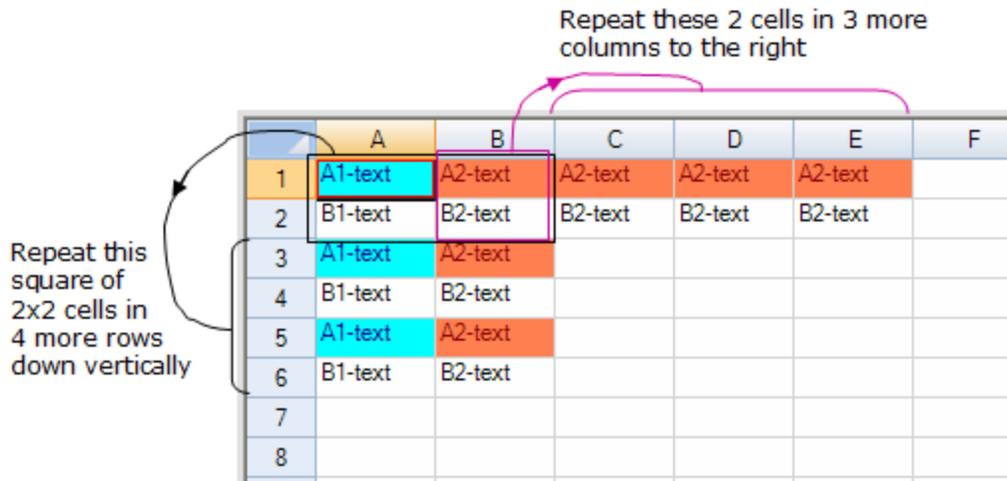
For more information on customizing Clipboard operations, such as copy, refer to **Customizing Clipboard Operation Options**.

Repeatedly Filling a Range of Cells with Copied Cells

You can copy a range of cells and fill another range with those cells, copying the data and the cell type with the **FillRange ('FillRange Method' in the on-line documentation)** method. For example, if you have a 2x2 range, you can repeat (fill) down the next five groups of 2x2 vertically.

The parameters for the **FillRange ('FillRange Method' in the on-line documentation)** method are:

- starting cell row and column index
- number of rows and columns in the range to copy
- number of either rows (if left or right) or columns (if up or down) to copy that range (not the number of times to repeat the entire range; the number of either rows or columns)



Using Code

1. Add data to the cells.
2. Set the **FillRange ('FillRange Method' in the on-line documentation)** method.

Example

For example, using this code, you would accomplish the results shown in the preceding figure.

C#

```
// Define the text to repeat.
fpSpread1.ActiveSheet.Cells[0, 0].Text = "A1-text";
fpSpread1.ActiveSheet.Cells[0, 1].Text = "A2-text";
fpSpread1.ActiveSheet.Cells[1, 0].Text = "B1-text";
fpSpread1.ActiveSheet.Cells[1, 1].Text = "B2-text";

fpSpread1.ActiveSheet.Cells[0, 0].BackColor = Color.Cyan;
fpSpread1.ActiveSheet.Cells[0, 0].ForeColor = Color.DarkBlue;
fpSpread1.ActiveSheet.Cells[0, 1].BackColor = Color.Coral;
fpSpread1.ActiveSheet.Cells[0, 1].ForeColor = Color.DarkRed;

// Fill 3 more columns to the right with the two columns' contents
fpSpread1.ActiveSheet.FillRange(0, 1, 2, 1, 3,
FarPoint.Win.Spread.FillDirection.Right);
// Fill 4 more rows down with the contents of the square
// of 2 rows and 2 columns
fpSpread1.ActiveSheet.FillRange(0, 0, 2, 2, 4, FarPoint.Win.Spread.FillDirection.Down);
```

VB

```
' Define the text to repeat.
fpSpread1.ActiveSheet.Cells(0, 0).Text = "A1-text"
```

```
fpSpread1.ActiveSheet.Cells(0, 1).Text = "A2-text"
fpSpread1.ActiveSheet.Cells(1, 0).Text = "B1-text"
fpSpread1.ActiveSheet.Cells(1, 1).Text = "B2-text"

fpSpread1.ActiveSheet.Cells(0, 0).BackColor = Color.Cyan
fpSpread1.ActiveSheet.Cells(0, 0).ForeColor = Color.DarkBlue
fpSpread1.ActiveSheet.Cells(0, 1).BackColor = Color.Coral
fpSpread1.ActiveSheet.Cells(0, 1).ForeColor = Color.DarkRed

' Fill 3 more columns to the right with the two columns' contents
fpSpread1.ActiveSheet.FillRange(0, 1, 2, 1, 3, FarPoint.Win.Spread.FillDirection.Right)
' Fill 4 more rows down with the contents of the square
' of 2 rows and 2 columns
fpSpread1.ActiveSheet.FillRange(0, 0, 2, 2, 4, FarPoint.Win.Spread.FillDirection.Down)
```

Swapping Data on a Sheet

You can swap the contents of two cells or two ranges of cells.

When you swap data from a cell or a range of cells to another cell or range of cells, the settings for the cell are swapped along with the data. If you provided settings for the column or the row containing the cell, or the spreadsheet, but not the cell itself, those settings are not swapped. For example, if you have set the source cell background color to red, the background color is swapped and the target cell has a red background. However, if you have set the background color of the column containing the source cell to red, that setting is not swapped.

When you swap data from one cell to another, the data in one cell becomes the data in the other cell, and vice versa. For example, if cell A1 contains the value 4 and cell B3 contains the value 6 and you swap the values of the cells, the value of cell A1 becomes 6 and the value of cell B3 becomes 4.

If you attempt to swap a range that is larger than the available range at the destination, the swap operation is not performed. For example, if you attempt to swap a range of four cells and specify the destination as a cell at the edge of the spreadsheet, the swap does not take place.

If the swap operation swaps overlapping ranges of cells, individual cells are swapped starting at the overlapping corner.

If the ranges overlap, such as moving rows 1 and 2 before row 0, you can add extra rows, move the rows, and then remove the extra rows.

For more information on methods to move or swap data, refer to their page in the API reference:

- **SheetView** Class: **SwapRange ('SwapRange Method' in the on-line documentation)** Method
- **SheetView** Class: **MoveRange ('MoveRange Method' in the on-line documentation)** Method
- Model Namespace: **DefaultSheetSpanModel** Class: **SwapColumns** Method
- Model Namespace: **DefaultSheetSpanModel** Class: **SwapRows Method** Method

Using Code

Here is an example of swapping a range of cells.

Example

This example swaps a range.

C#

```
fpSpread1.ActiveSheet.RowCount = 10;
fpSpread1.ActiveSheet.ColumnCount = 10;
private void button1_Click(object sender, System.EventArgs e)
```

```
{  
    fpSpread1.ActiveSheet.SwapRange(0, 0, 3, 0, 3, 3, true);  
}
```

VB

```
fpSpread1.ActiveSheet.RowCount = 10  
fpSpread1.ActiveSheet.ColumnCount = 10  
Private Sub void button1_Click(sender As Object, e As System.EventArgs)  
    fpSpread1.ActiveSheet.SwapRange(0, 0, 3, 0, 3, 3, True)  
End Sub 'button1_Click
```

Example

This example adds rows, moves rows, and then removes the extra rows.

C#

```
fpSpread1.ActiveSheet.RowCount = 10;  
fpSpread1.ActiveSheet.ColumnCount = 10;  
private void button1_Click(object sender, System.EventArgs e)  
{  
    fpSpread1.ActiveSheet.Rows[0,1].Add() ;  
    fpSpread1.ActiveSheet.MoveRange(3,0,0,0,2,4,true);  
    fpSpread1.ActiveSheet.Rows[3,4].Remove();  
}
```

VB

```
fpSpread1.ActiveSheet.RowCount = 10  
fpSpread1.ActiveSheet.ColumnCount = 10  
Private Sub void button1_Click(sender As Object, e As System.EventArgs)  
    fpSpread1.ActiveSheet.Rows(0,1).Add()  
    fpSpread1.ActiveSheet.MoveRange(3,0,0,0,2,4,True)  
    fpSpread1.ActiveSheet.Rows(3,4).Remove()  
End Sub 'button1_Click
```

When you swap ranges of data, you can specify whether formulas are adjusted. For more information, see **Recalculating and Updating Formulas Automatically**.

Removing Data from a Sheet

You can remove both data and cell formatting from a selected cell or range of cells, or remove only the data, leaving the cell formatting intact. For more information about cell formatting, refer to **Understanding How Cell Types Display and Format Data**. You can remove the data using any of the clear methods or by cutting the data using the Clipboard operation.

You can remove the data using any of these clear methods in the default data model:

- **Clear ('Clear Method' in the on-line documentation)**, which clears both data and formulas
- **ClearFormulas ('ClearFormulas Method' in the on-line documentation)**, which clears only formulas
- **ClearData ('ClearData Method' in the on-line documentation)**, which clears data and formula of cells in the range.
- **ClearCustomNames ('ClearCustomNames Method' in the on-line documentation)**, which clears custom names, and **ClearCustomFunctions ('ClearCustomFunctions Method' in the on-line**

- documentation), which clears custom functions
- **ClearRange ('ClearRange Method' in the on-line documentation)**, which clears data, formulas, notes, and formatting from a range of cells

If you use **ClearRange** and set the *dataOnly* parameter to true, the method clears the formulas, the cell notes, and the text in the cells in that range; in other words, it clears all the information that is in the data model for those cells.

You can remove the contents of a range of cells using this method in the range interface:

- **IRangeSupport.Clear ('Clear Method' in the on-line documentation)**

Remove Duplicates from Range

In Spread, the duplicate data can be highlighted using conditional formatting. However, if there is a large amount of data, removing duplicate data is always preferred to ease data analysis.

With Spread for WinForms, you can permanently delete the duplicate data from the selected range using the "**Remove Duplicates**" option. When you remove the duplicate data or values from the selected range, the only effect is on the values in the cell range. Other values outside the range do not change or move. When the duplicate data is removed, the first occurrence of the value in the list is kept, but other identical values are deleted.

Using Code

You can set the **RemoveDuplicates** method from **IRange** interface to remove the duplicate data from the range of cells by specifying the target range.

The following image shows the data before and after it has been modified using code:

	A	B	C	D	E
1	Steve	1	1	1	1
2	Steve	1	1	1	1
3	Pete	2	2	2	2
4	Mike	3	3	3	3
5	Steve	5	5	5	5
6	Emily	8	8	8	8
7	Steve	13	13	13	13

Before removing duplicates

	A	B	C	D	E
1	Steve	1	1	1	1
2	Pete	2	2	2	2
3	Mike	3	3	3	3
4	Steve	5	5	5	5
5	Emily	8	8	8	8
6	Steve	13	13	13	13
7					

After removing duplicates

C#

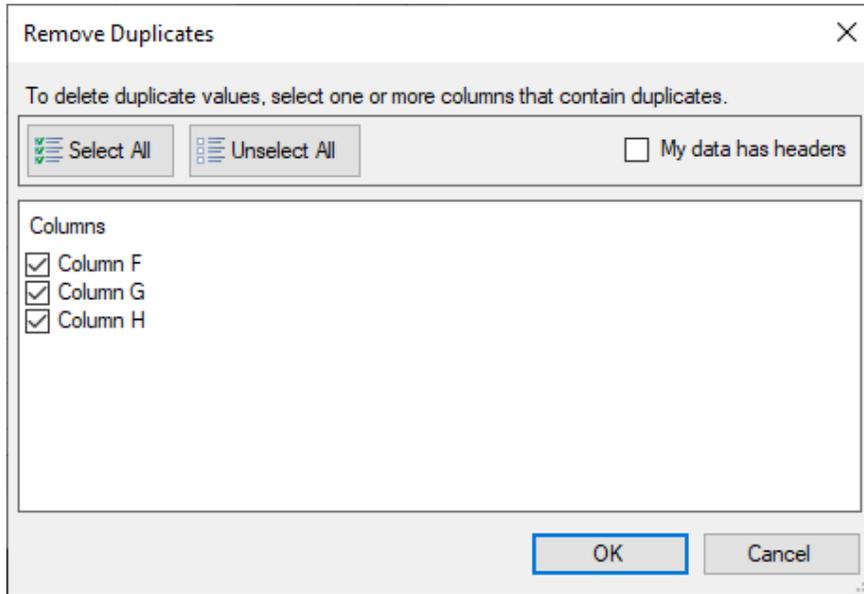
```
int[] columns = { 1, 2 };
fpSpread1.AsWorkbook().ActiveSheet.Range("A1:E7").RemoveDuplicates(columns,
YesNoGuess.No);
```

VB

```
Dim columns As Integer() = {1, 2}
fpSpread1.AsWorkbook().ActiveSheet.Range("A1:E7").RemoveDuplicates(columns,
YesNoGuess.No)
```

Using Runtime UI

You can enable the built-in remove duplicates dialog box using the **RemoveDuplicates** method of the **BuiltinDialogs** class at run-time.



The following code example shows how to use the runtime dialog box in a Spread worksheet:

C#

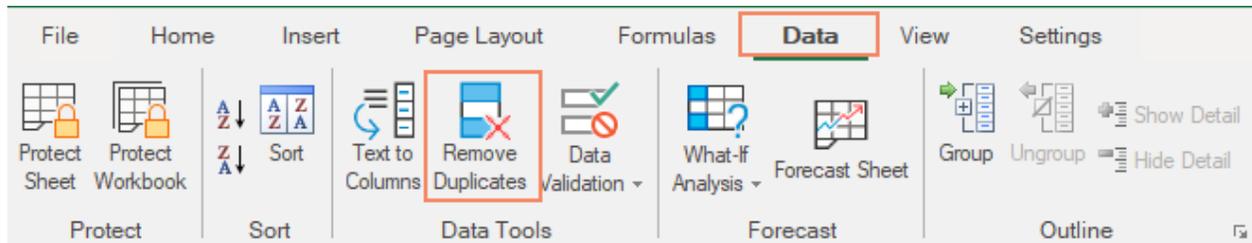
```
activeSheet.Cells["A1:E7"].Select();
FarPoint.Win.Spread.Dialogs.BuiltInDialogs.RemoveDuplicates(fpSpread1).ShowDialog(fpSpread1);
```

VB

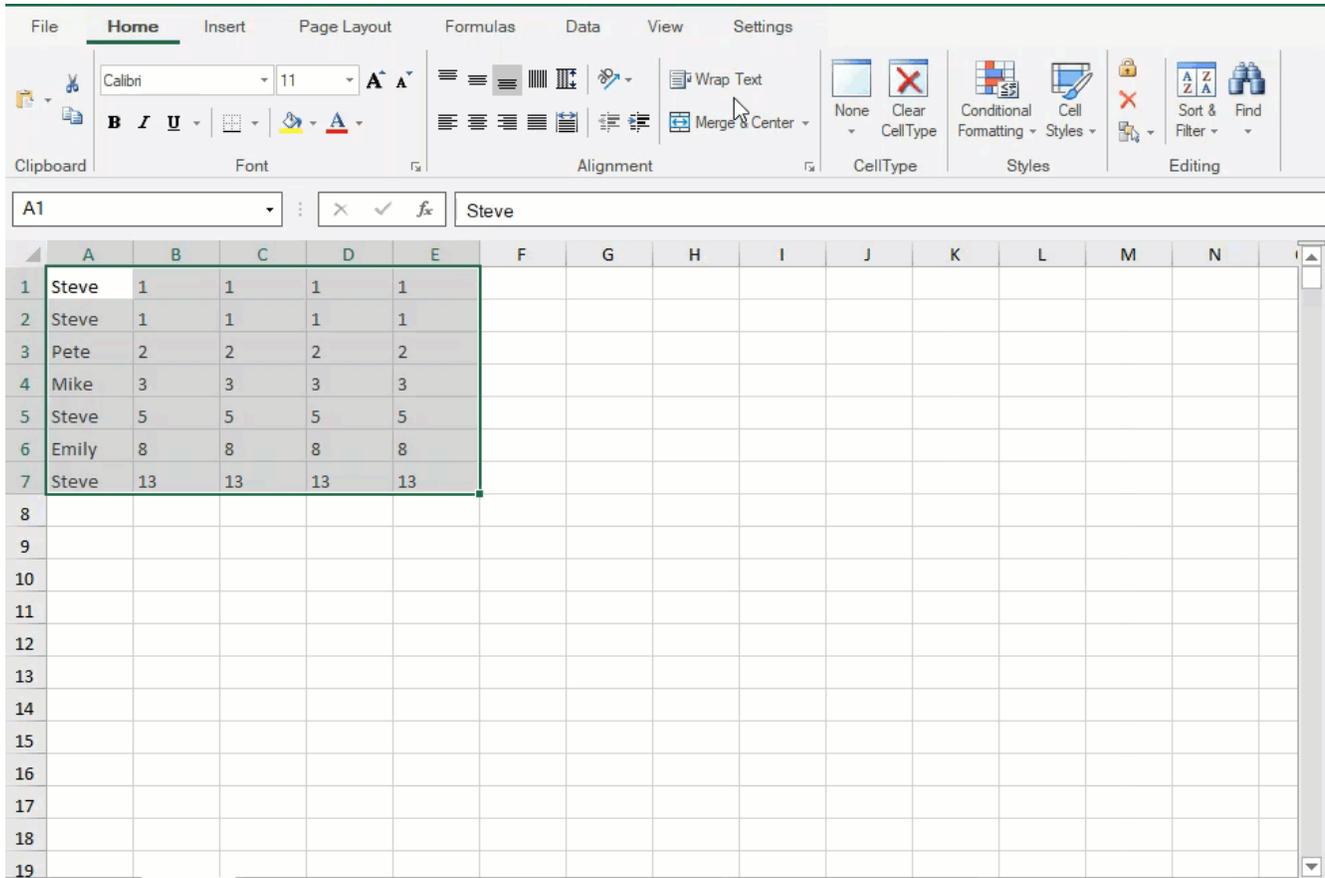
```
activeSheet.Cells("A1:E7").Select()
FarPoint.Win.Spread.Dialogs.BuiltInDialogs.RemoveDuplicates(fpSpread1).ShowDialog(fpSpread1)
```

Using Designer

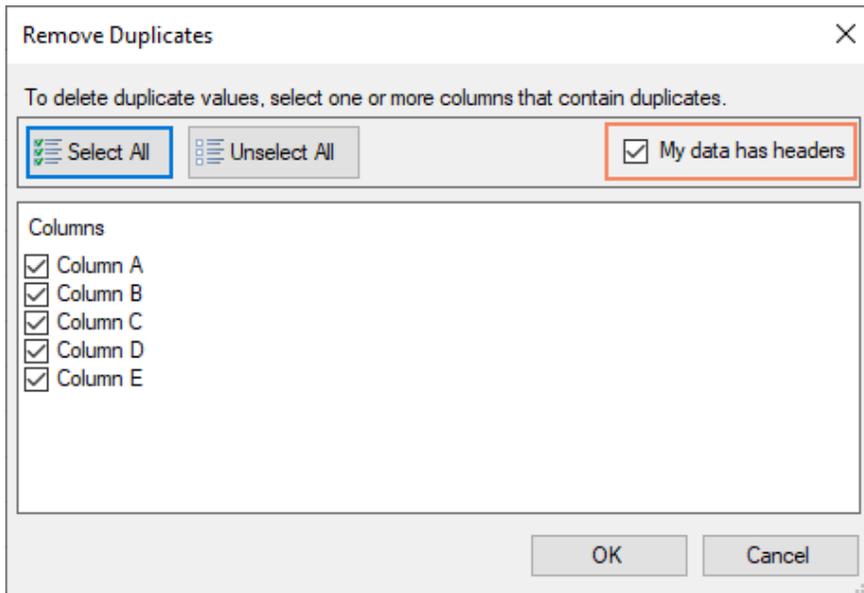
Spread for WinForms designer provides the **"Remove Duplicates"** ribbon button under the **"Data" > "Data Tools"** tab group.



The following GIF illustrates the removal of duplicates data from the selected range of data.



You can also choose to ignore the first row by selecting the "My Data has Headers" checkbox option.



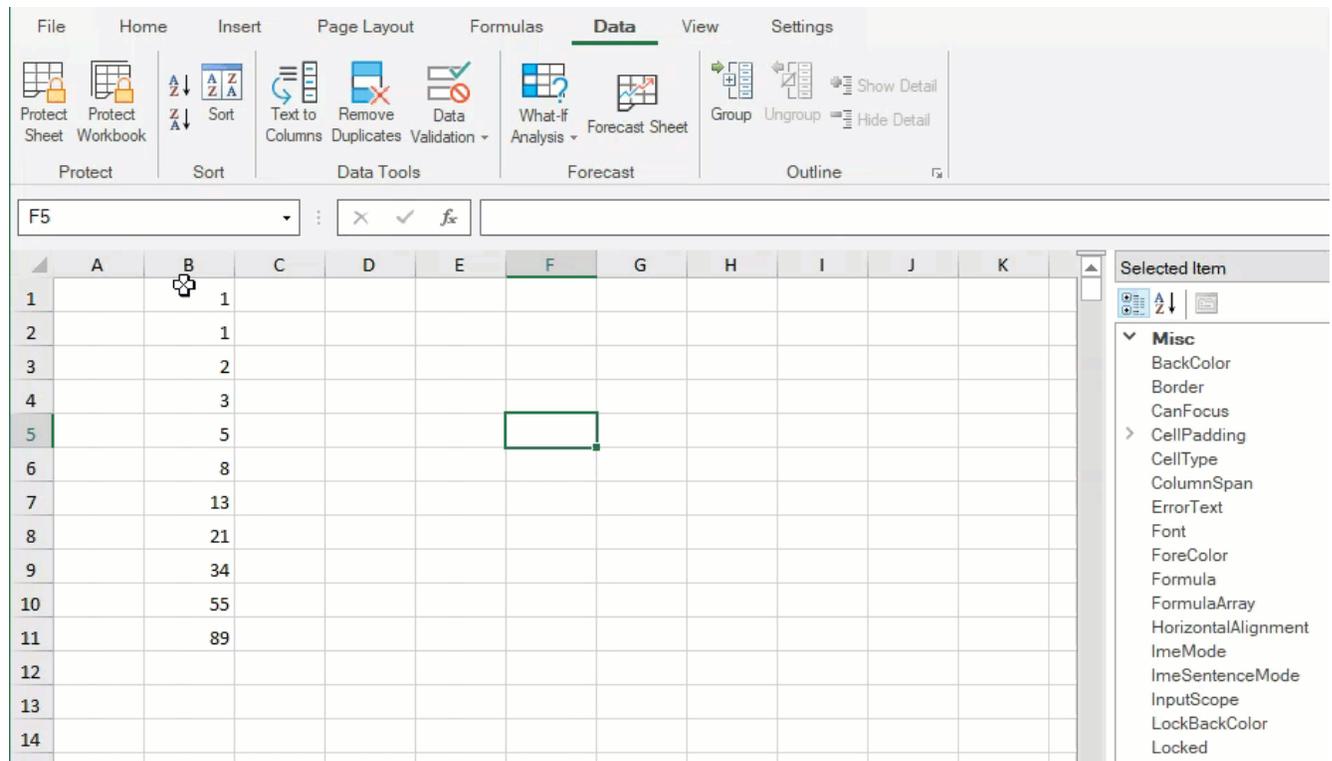
Feature Interaction with Different Types of Values

1. With Formulas

The formula in the cell after the duplicate cell is copied over. The following GIF illustrates the use of "Remove

Duplicates" in the range of cells where a formula is applied for the Fibonacci series.

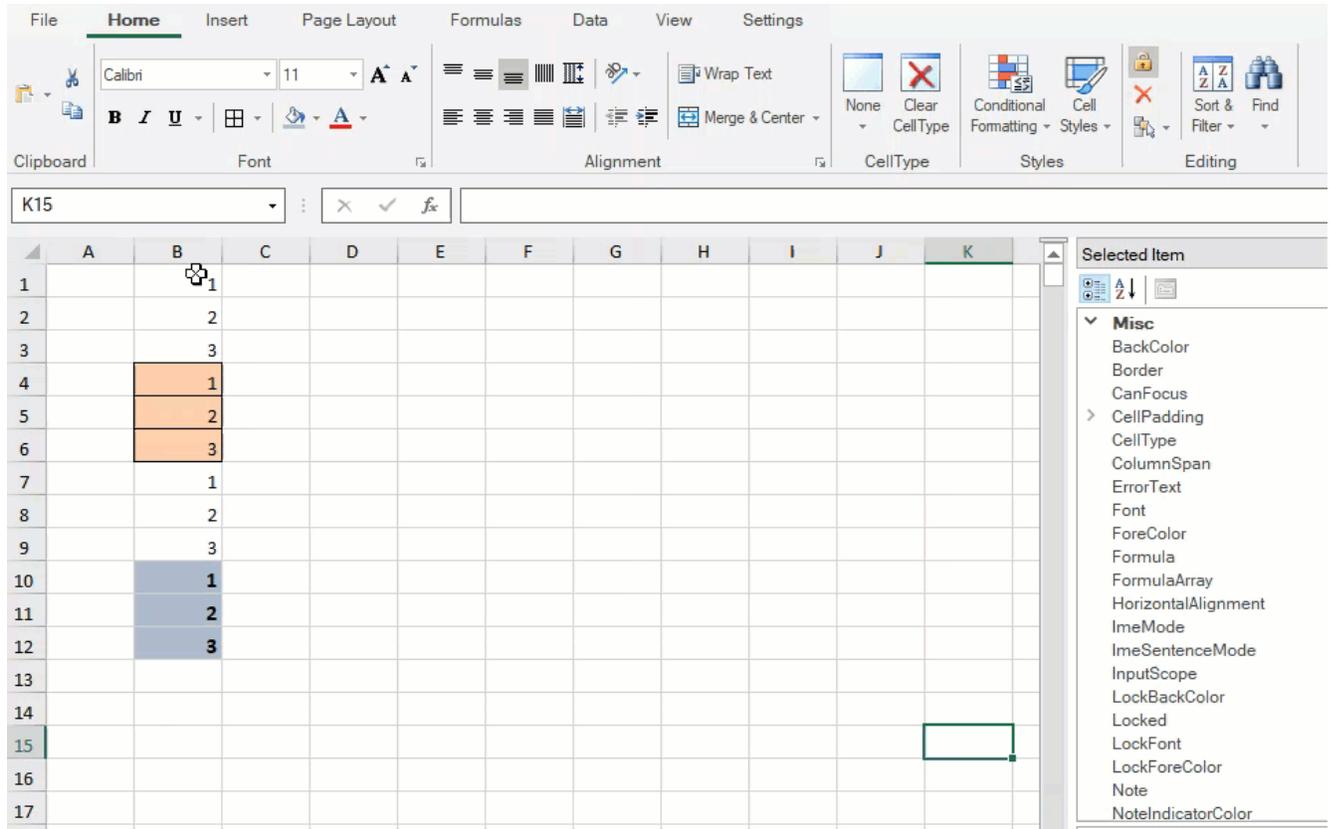
As observed, when the "Remove Duplicates" feature is implemented, it removed 'B2' and copies 'B3' to 'B2' and each next cell moves forward by one, changing "B3" to "=B1+B2".



2. With Formatting

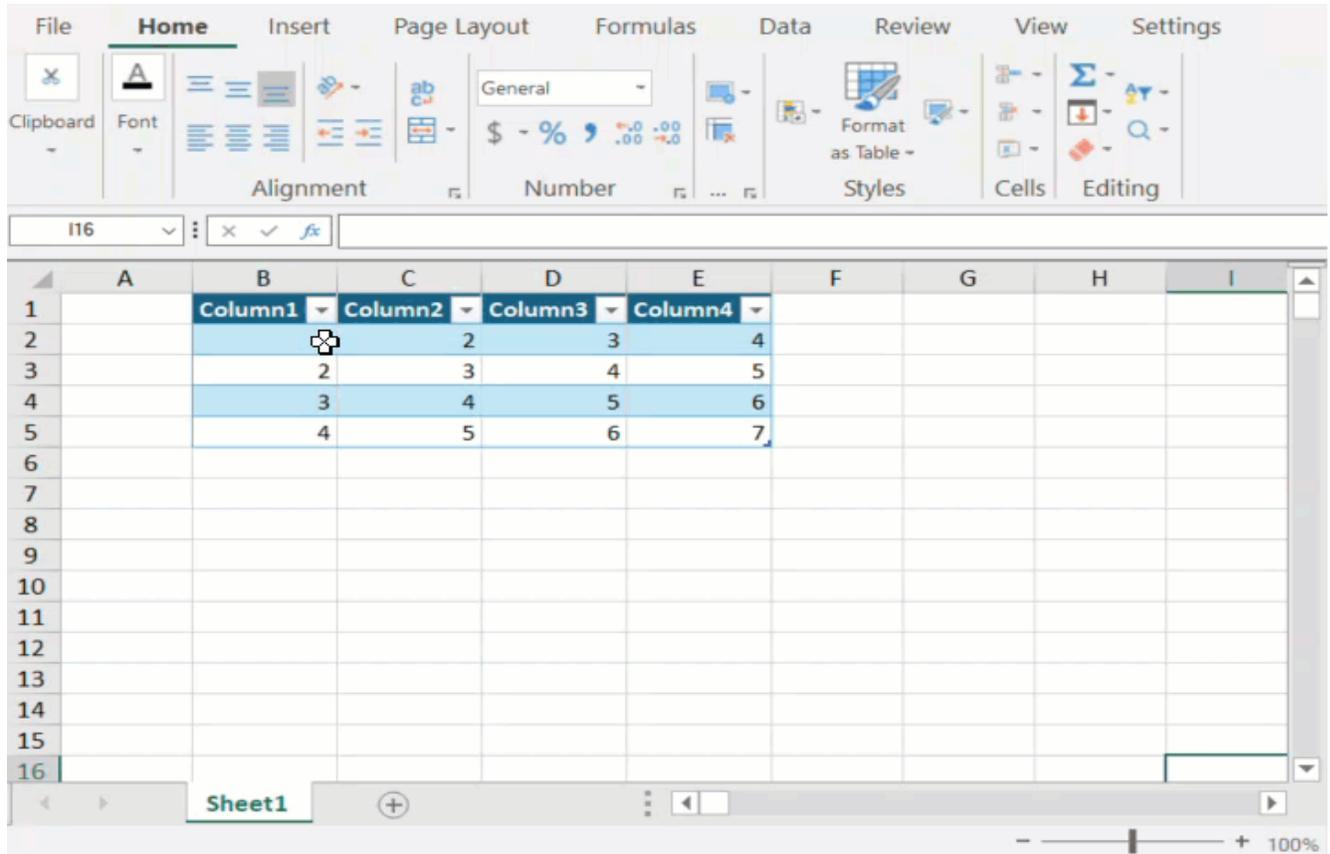
If the value of a cell is the same as that of another cell in the select range, formatting and style are not considered as different cells, and they are removed along with the value.

As observed, the cells have different formatting even though the value of the cells is 1,2, and 3. After implementing Remove Duplicates, all the same value cells are removed.



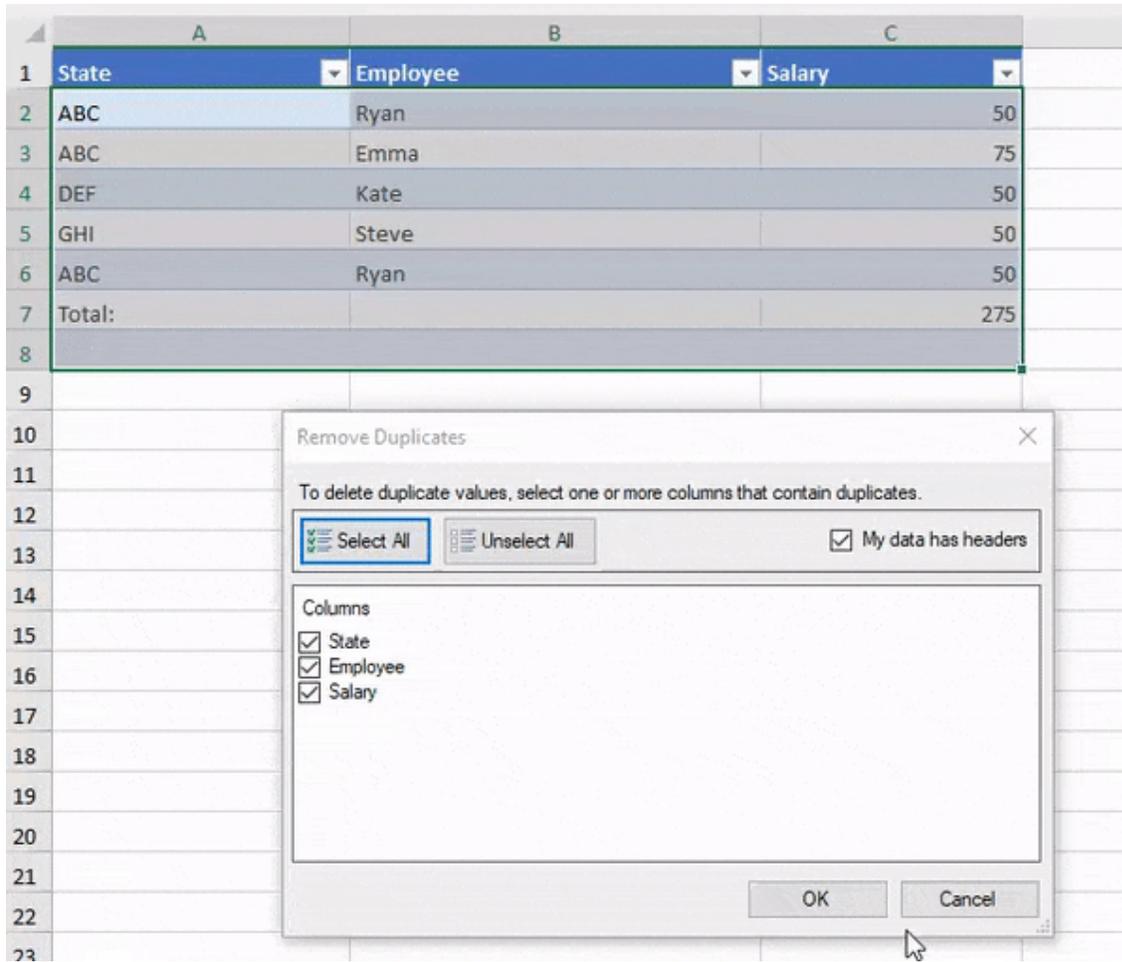
3. With Automatic Expansion Selection

While using remove duplicates, if you select one or more columns of the table and click on remove duplicates, then the selection is expanded to select the whole table.



4. **Table Row With Total**

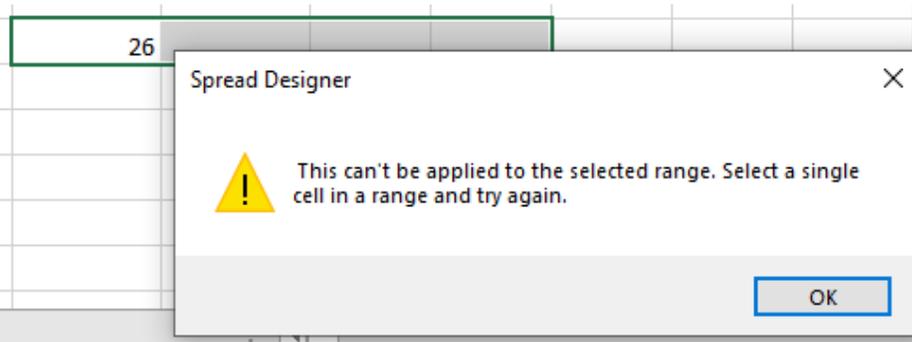
While using remove duplicates, if you select a range of cells from a table that contains a total row then the total row is not ignored and the duplicate values are removed as well.



Invalid Selection

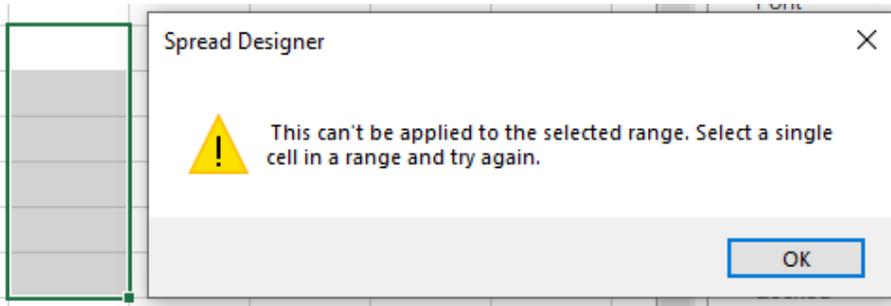
1. Select Single Row

If a single row is selected.



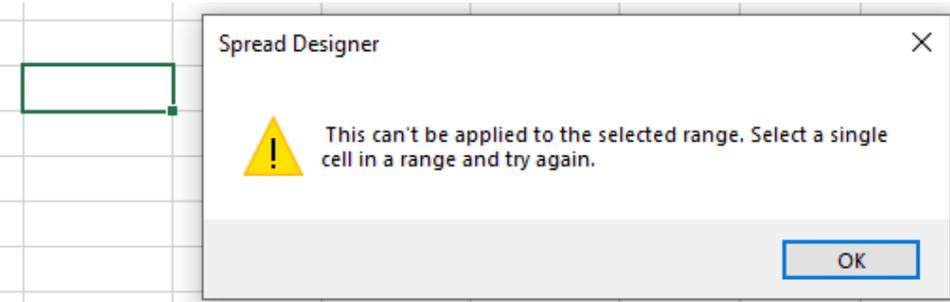
2. Select With No Data

If the selection does not contain any data.



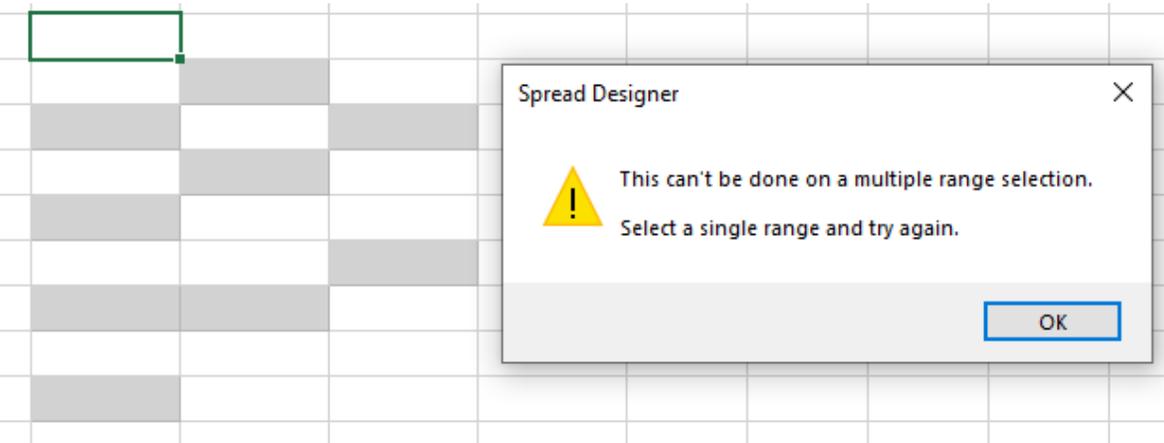
3. Select One Cell

If a single cell is selected.



4. Select Multiple Areas

If multiple ranges are selected.



Text to Columns

Text to Columns is used to convert simple text or sentences having multiple words into separate columns. It enables us to get the tabular structure of data using a suitable delimiter or column width option.

For example, when you want to separate a list of full names into first and last names, you can use Text to Columns feature.

Using Code

You can set the **TextToColumns** method from the **IRange** interface to parse the text from one cell or column into many columns.

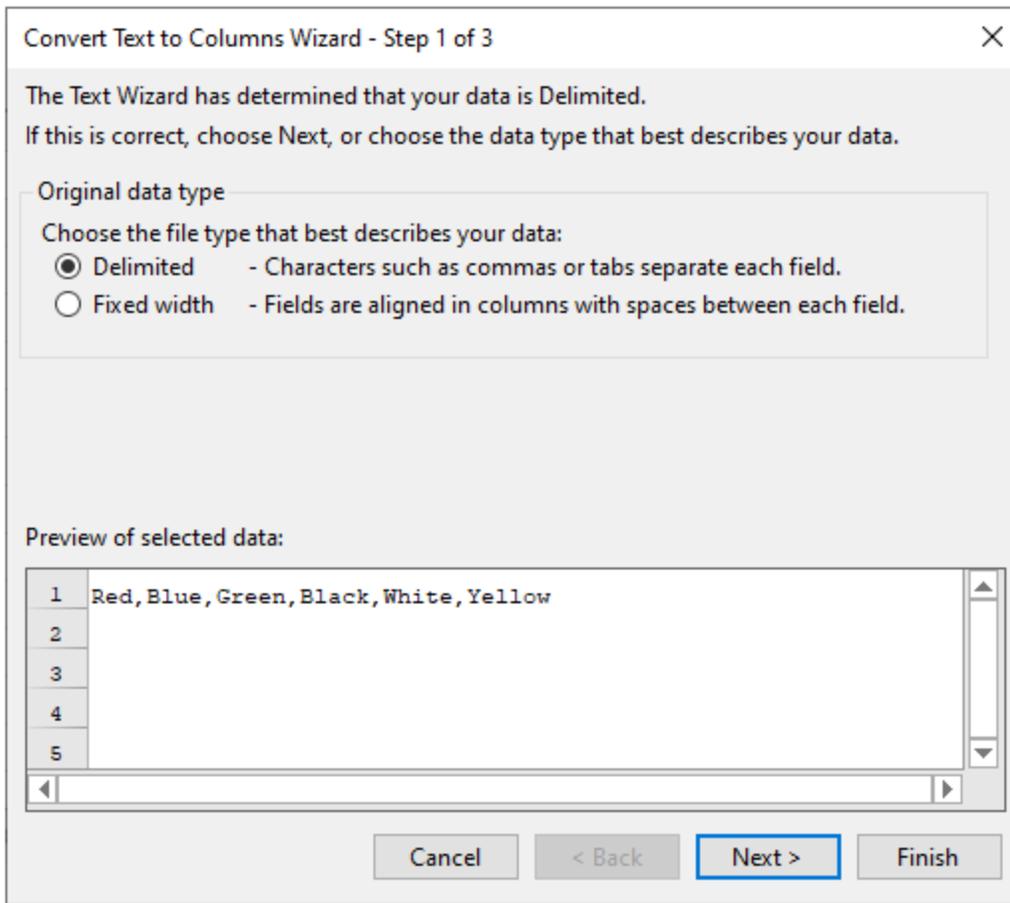
	A	B	C	D	E	F	G	H
1	Jacob Dev SpreadWin							
2	Steve Dev Raykit							
3	Serena Tester SpreadWin							
4	Keira Tester Raykit							
5	Jacob	Dev	SpreadWin					
6	Steve	Dev	Raykit					
7	Serena	Tester	SpreadWin					
8	Keira	Tester	Raykit					
9								
10								

C#

```
TestActiveSheet.Cells["A1"].Value = "Jacob Dev SpreadWin";
TestActiveSheet.Cells["A2"].Value = "Steve Dev Raykit";
TestActiveSheet.Cells["A3"].Value = "Serena Tester SpreadWin";
TestActiveSheet.Cells["A4"].Value = "Keira Tester Raykit";
TestActiveSheet.Cells["A1:A4"].TextToColumns("A5", TextParsingType.Delimited,
TextQualifier.None, false, false, false, false, true);
```

Using Runtime UI

You can enable the built-in TextToColumn dialog box using **BuiltInDialogs** class at run-time to the end-user with the specified range.



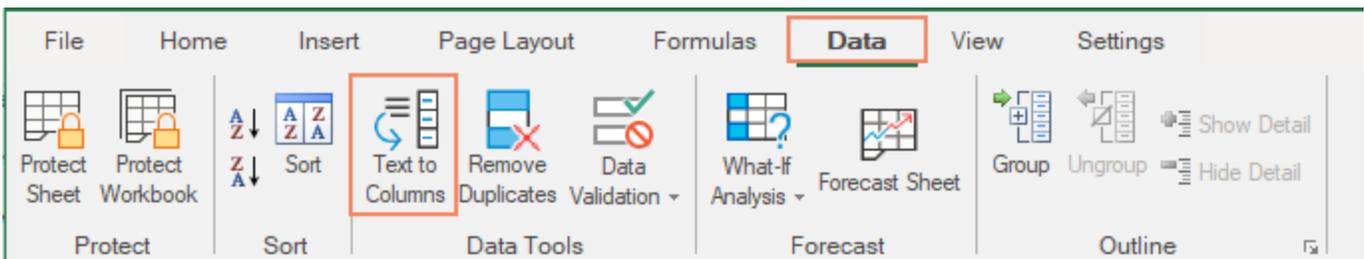
The following code example shows how to use the runtime dialog box in a Spread worksheet:

C#

```
TestActiveSheet.Cells["A1"].Value = "Red,Blue,Green,Black,White,Yellow";
FarPoint.Win.Spread.Dialogs.BuiltInDialogs.TextToColumns(fpSpread1).Show(fpSpread1);
```

Using Spread Designer

Spread for WinForms designer provides the "Text to Columns" ribbon button in the "Data Tools" group of the "Data" tab.



Steps to Convert Text to Columns

Using Delimiters

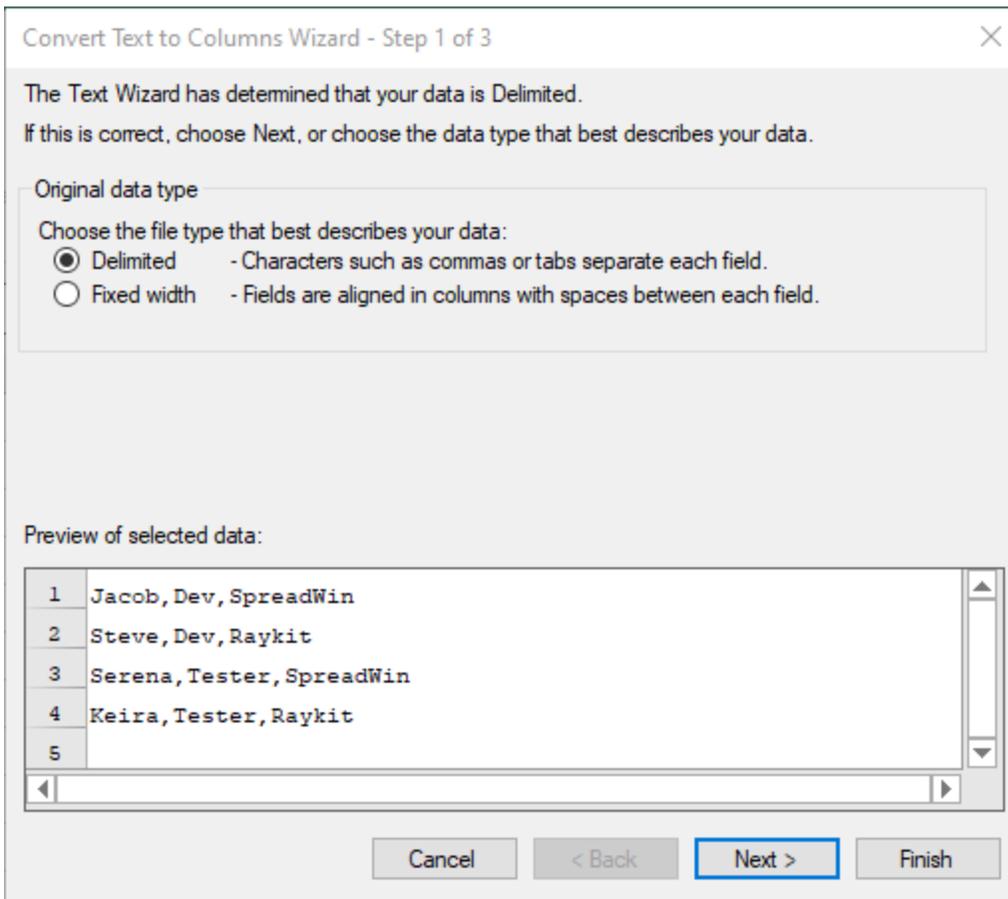
This method splits the text using the present delimiter in a cell or range. You can define your delimiter as well. Only one character is considered a delimiter.

Follow the steps below to convert the sample text using the comma delimiter.

1. Select the cell or range containing text.

	A	B
1	Jacob,Dev,SpreadWin	
2	Steve,Dev,Raykit	
3	Serena,Tester,SpreadWin	
4	Keira,Tester,Raykit	
5		

2. Click "**Text to Columns**" from the "Data Tools" group in the "Data" tab. The "Convert Text to Columns Wizard" window opens.
3. Select the **Delimited** option from the "Original data type" section.



Click **Next** to open the "**Convert Text to Columns Wizard- Step 2 for 3**" window.

4. Set the delimiter from the pre defined list of delimiters or you can enter your delimiter by specifying the desired character in the "Other" option.

Convert Text to Columns Wizard - Step 2 of 3

This screen lets you set the delimiters your data contains. You can see how your text is affected in the preview below.

Delimiters

Tab
 Semicolon
 Comma
 Space
 Other

Treat consecutive delimiters as one

Text qualifier: "

Data preview

Jacob	Dev	SpreadWin
Steve	Dev	Raykit
Serena	Tester	SpreadWin
Keira	Tester	Raykit

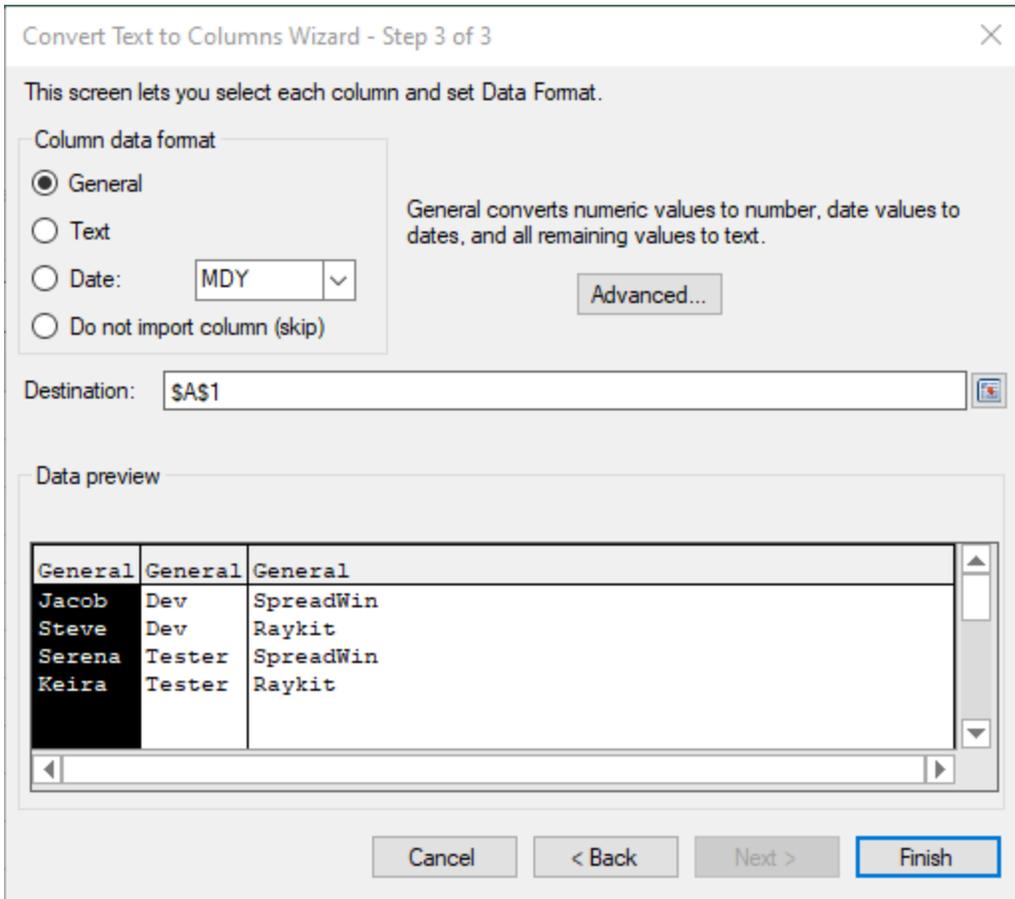
Cancel < Back **Next >** Finish

The delimiter in the "" or "" will be ignored.

Check "**Treat consecutive delimiters as one**" option to allow if multiple characters ";;;" are specified, they will be treated as one ";" only.

- Click **Next** to open the "**Convert Text to Columns wizard-Step 3 of 3**" window. Now, set the Data format for each column.

You can select the **Destination** for the text. The default destination is the original cell.



6. Click **Finish** to convert the text to the cell into columns.

	A	B	C
1	Jacob	Dev	SpreadWin
2	Steve	Dev	Raykit
3	Serena	Tester	SpreadWin
4	Keira	Tester	Raykit

Using Fixed Widths

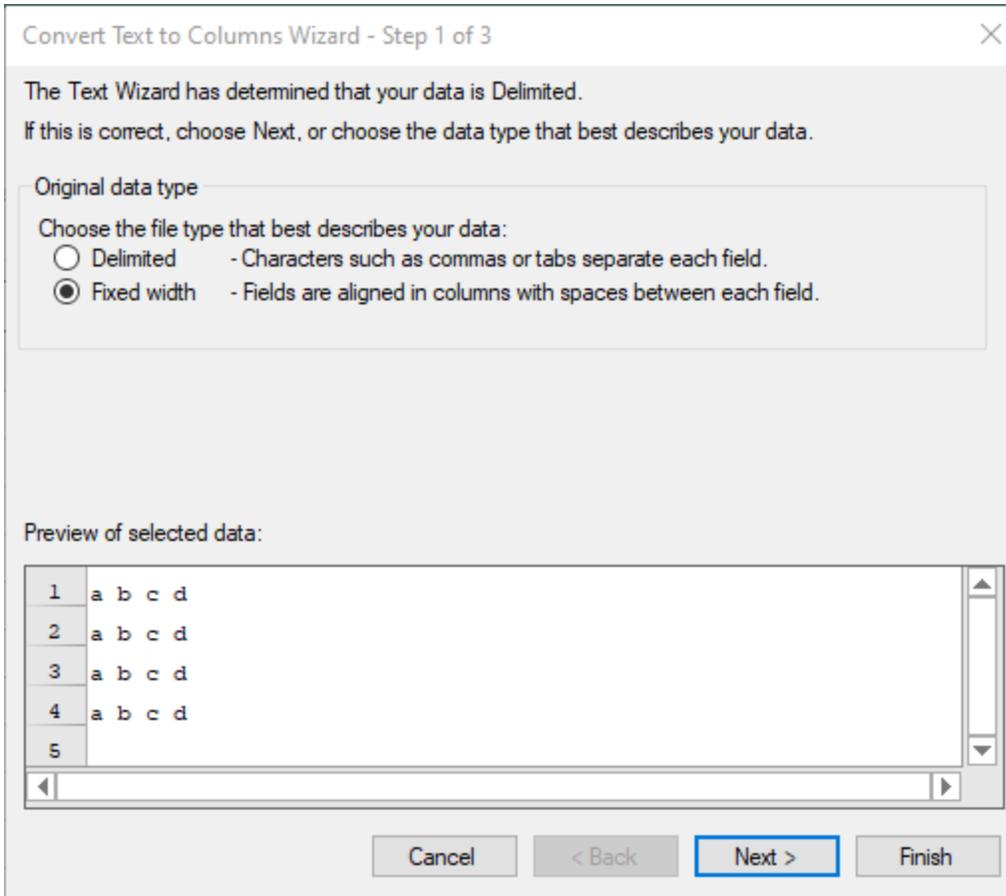
This method splits the text using the fixed width between text. You can set column breaks to separate the text into columns.

Follow the steps below to convert the sample text using the fixed width:

1. Select the cell or range where you want to convert the text into columns.

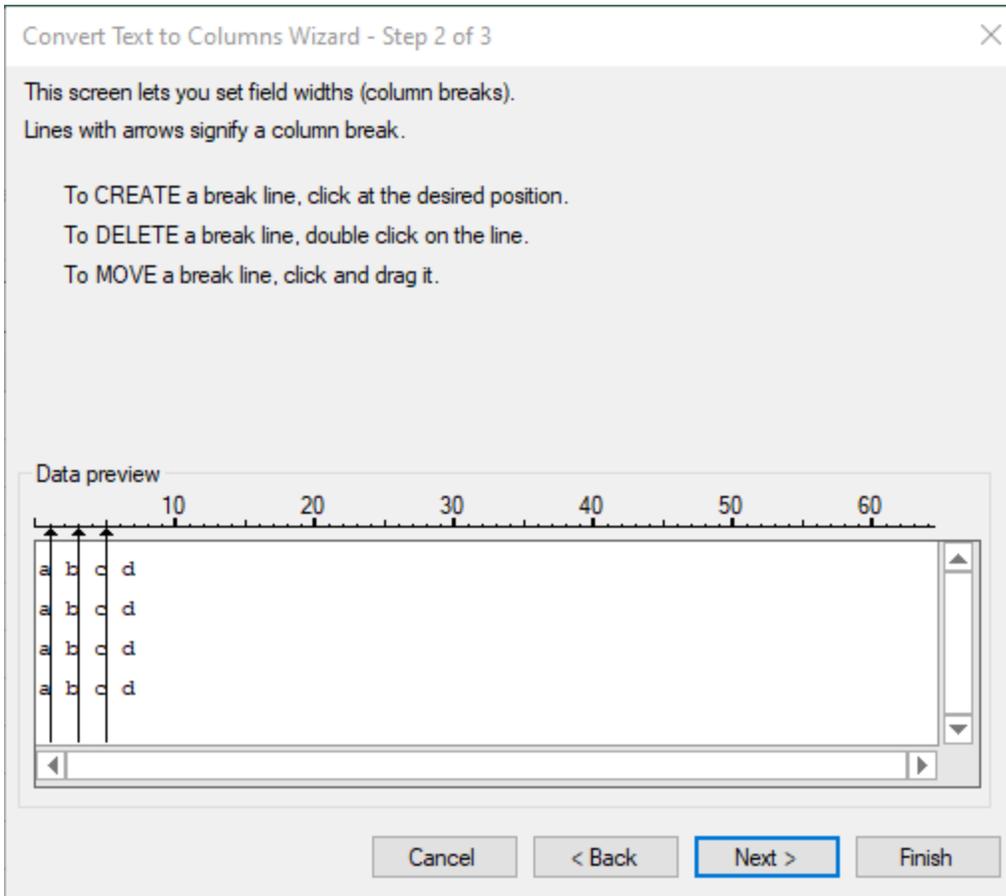
	A	B
1	a b c d	
2	a b c d	
3	a b c d	
4	a b c d	

2. Click "**Text to Columns**" from the "**Data Tools**" group in the "**Data**" tab. The "**Convert Text to Columns Wizard**" window opens.
3. Select the **Fixed width** option from the "**Original data type**" section.



Click **Next** to open the "**Convert Text to Columns Wizard- Step 2 for 3**" window.

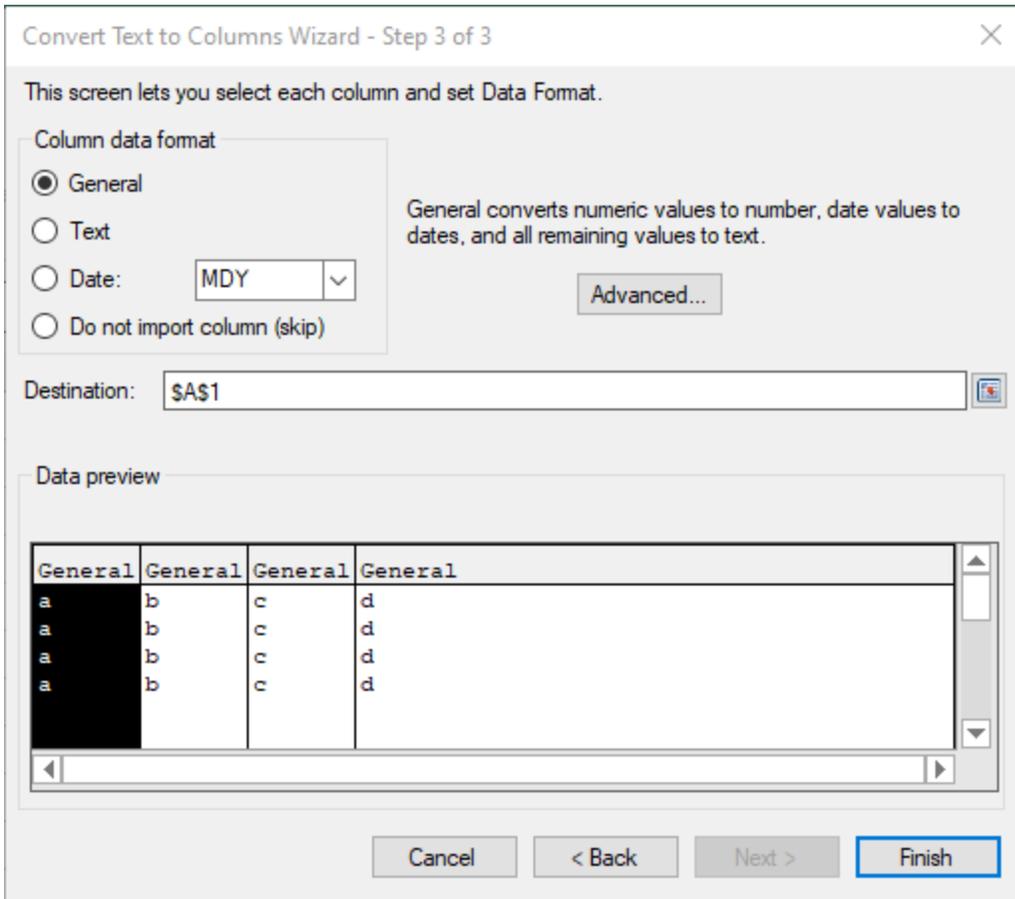
4. Create fixed-width column breaks using the instructions mentioned on the window.



Click **Next** to open the "**Convert Text to Columns wizard-Step 3 of 3**" window.

5. Set the **Data format** for each column.

You can select the **Destination** for the text. The default destination is the original cell.



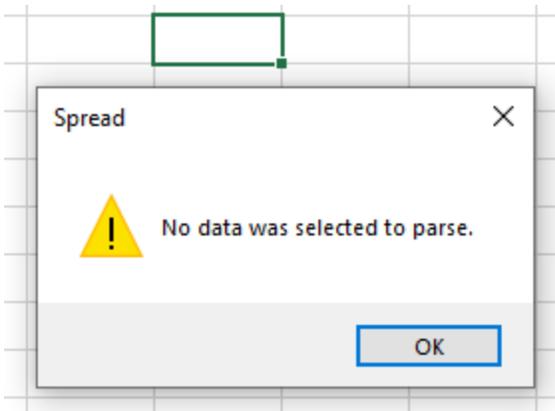
6. Click **Finish** to convert the text to the cell into columns.

	A	B	C	D
1	a	b	c	d
2	a	b	c	d
3	a	b	c	d
4	a	b	c	d
5				

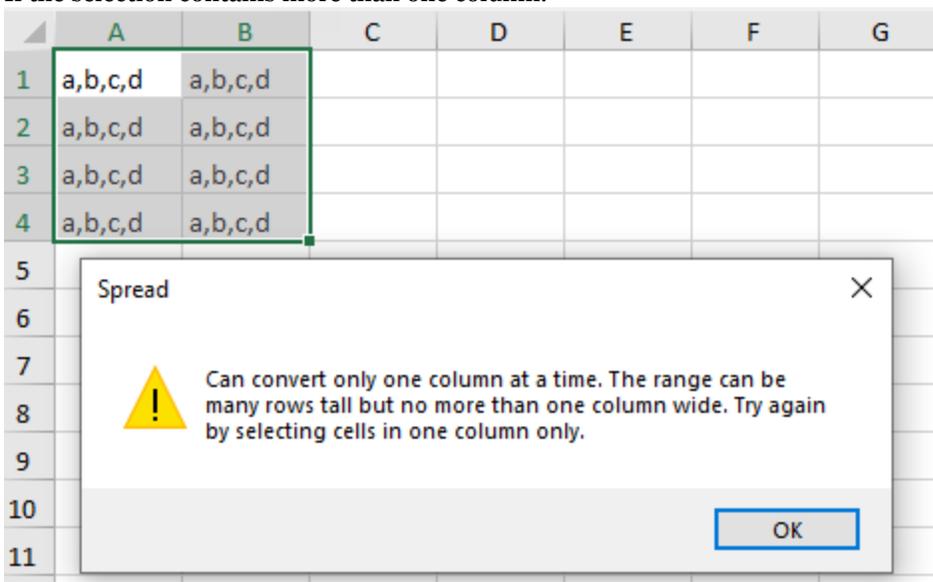
Warnings

A warning is displayed in the following scenarios:

1. If the selected cell or range has no data.



2. If the selection contains more than one column.



Creating Data Type for Custom Objects

Spread.NET provides support to add Data types for custom objects just like Excel. Users can access information about a wide range of subjects and areas within the Spread interface. This feature will allow users to access all the required information with an easy card pop up and extract key data from relevant objects effortlessly.

When a data type is added to a cell, a glyph is added to it indicating the data type. So, when the glyph is clicked, a data card pops up displaying the properties of the object. By default, data types are disabled in Spread. To enable them, **CalcFeatures** enumeration should be enabled.

You can add a data type to a cell using the **IRichValue** interface, which represents a rich cell value. For this purpose, you can implement a class which implements **IRichValue** interface and set the instance to the worksheet. Here, we have used the Employee class with the built-in class **RichValue<T>**.

In Spread, **DataTable** and **DataGridView** are built-in supported. Users can create the **IRichValue** object that allows to extract data from columns of **DataTable** or **DataGridView**.

C#

```
IRichValue richValue = BuiltInRichValue.FromDataTable(tbl, defaultColumnName, showHeaders);
```

VB

```
Dim richValue As IRichValue = BuiltInRichValue.FromDataTable(tbl, defaultColumnName, showHeaders)
```

In the use-case image shown below, the class `Employee` implements the **IMedia** interface. `Employee` class implements `IMedia` interface to customize the inline rich value icon of cell. This class has fields like `ID`, `First Name`, `Last Name`, `Designation`, `Department`, `Gender`, `Age` and `Year of Joining`. To add data type in the worksheet, you can set objects in cell values using the **Value** property of **IRange** interface, and set formulas in cells using the **Formula** property of **IRange** interface. Here, we have set values in cells from `B2 : B8` and `C2 : C3`.

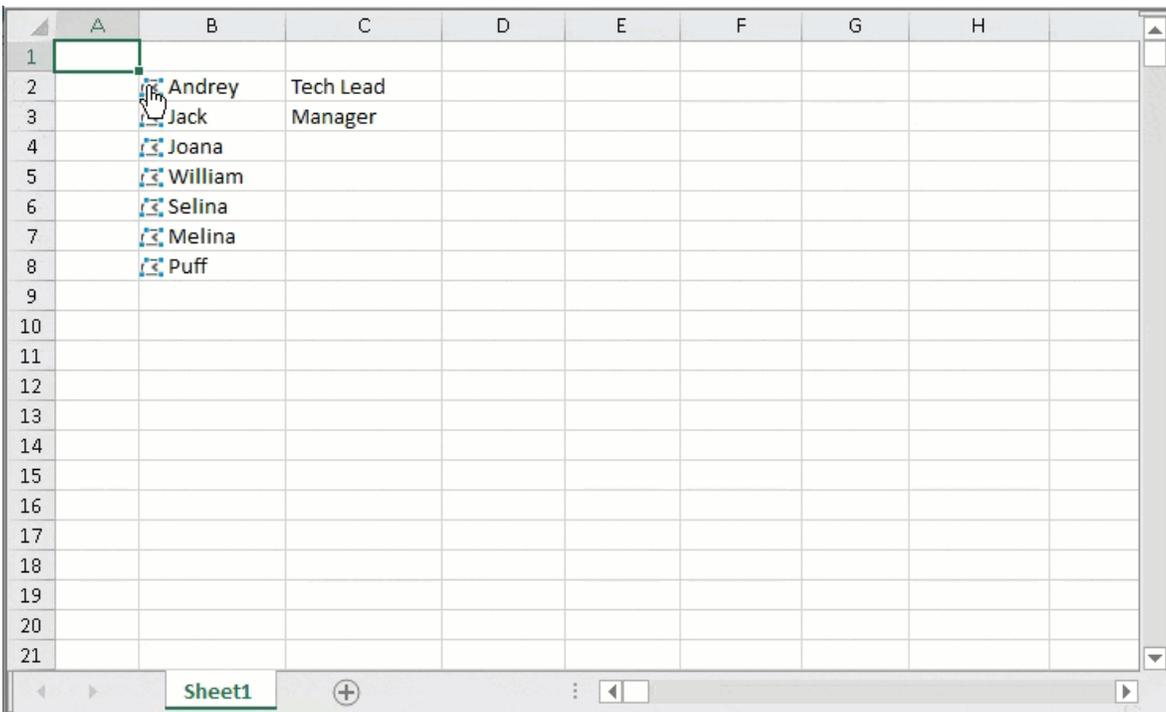
In case a normal cell containing a data type is activated, the insert button is displayed at the top-right of cell and users can add any of its related property (field) to the nearest empty cell at the right. The cell where the property is filled contains the related formula. On the other hand, if a table cell containing a data type is activated, the insert button is displayed at the top-right of table and users can add any of its related property (field). Once the user selects a property, a new column is added towards the right. The newly added column has the name of the added property and fills relevant values of the property in the column. It contains the related formula.

In order to use formula to access properties of a custom .NET data object we need to use the following syntax.

`B2.Property1`

or

`B2.[The Property 1]`



The code below implements the use-case depicting adding data types:

C#

```
private void Form1_Load(object sender, EventArgs e)
{
    // Enable CalcFeatures to All
    fpSpread1.AsWorkbook().WorkbookSet.CalculationEngine.CalcFeatures = CalcFeatures.All;
    fpSpread1.ActiveSheet.Columns[1, 10].Width = 120;
    GrapeCity.CalcEngine.RichValue<Employee> andreySmith = new
    GrapeCity.CalcEngine.RichValue<Employee>(new Employee ()
    {
        ID = 61371,
        FirstName = "Andrey",
        LastName = "Smith",
        Designation = "Tech Lead",
        Department = "IT",
        Gender = "M",
        Age = 34,
        YearOfJoining = 2013
    });
    GrapeCity.CalcEngine.RichValue<Employee> jackShang = new
    GrapeCity.CalcEngine.RichValue<Employee>(new Employee ()
    {
        ID = 37123,
```

```
        FirstName = "Jack",
        LastName = "Shang",
        Designation = "Manager",
        Department = "Sales",
        Gender = "M",
        Age = 34,
        YearOfJoining = 2017
    });
    GrapeCity.CalcEngine.RichValue<Employee> joanaJordan = new
    GrapeCity.CalcEngine.RichValue<Employee>(new Employee()
    {
        ID = 37564,
        FirstName = "Joana",
        LastName = "Jordan",
        Designation = "GraphicsDesigner",
        Department = "IT",
        Gender = "F",
        Age = 28,
        YearOfJoining = 2016
    });
    GrapeCity.CalcEngine.RichValue<Employee> williamSmith = new
    GrapeCity.CalcEngine.RichValue<Employee>(new Employee()
    {
        ID = 28034,
        FirstName = "William",
        LastName = "Smith",
        Designation = "AVP",
        Department = "HR",
        Gender = "M",
        Age = 42,
        YearOfJoining = 2012
    });
    GrapeCity.CalcEngine.RichValue<Employee> selinaWing = new
    GrapeCity.CalcEngine.RichValue<Employee>(new Employee()
    {
        ID = 65134,
        FirstName = "Selina",
        LastName = "Wing",
        Designation = "Technical Engineer",
        Department = "IT",
        Gender = "F",
        Age = 34,
        YearOfJoining = 2019
    });
    GrapeCity.CalcEngine.RichValue<Employee> melinaJackson = new
    GrapeCity.CalcEngine.RichValue<Employee>(new Employee()
    {
        ID = 45978,
        FirstName = "Melina",
        LastName = "Jackson",
        Designation = "Senior S/W Engineer",
        Department = "IT",
        Gender = "F",
        Age = 34,
        YearOfJoining = 2014
    });
    GrapeCity.CalcEngine.RichValue<Employee> puffDuplacy = new
    GrapeCity.CalcEngine.RichValue<Employee>(new Employee()
    {
        ID = 32700,
        FirstName = "Puff",
        LastName = "Duplacy",
        Designation = "Team Lead",
        Department = "IT",
        Gender = "M",
        Age = 36,
        YearOfJoining = 2012
    });
```

```

    });

    // set object in cells value
    fpSpread1.AsWorkbook().ActiveSheet.Cells["B2"].Value = andreySmith;
    fpSpread1.AsWorkbook().ActiveSheet.Cells["B3"].Value = jackShang;
    fpSpread1.AsWorkbook().ActiveSheet.Cells["B4"].Value = joanaJordan;
    fpSpread1.AsWorkbook().ActiveSheet.Cells["B5"].Value = williamSmith;
    fpSpread1.AsWorkbook().ActiveSheet.Cells["B6"].Value = selinaWing;
    fpSpread1.AsWorkbook().ActiveSheet.Cells["B7"].Value = melinaJackson;
    fpSpread1.AsWorkbook().ActiveSheet.Cells["B8"].Value = puffDuplacy;
    // set formulas in cells
    fpSpread1.AsWorkbook().ActiveSheet.Cells["C2"].Formula = "B2.[Designation]";
    fpSpread1.AsWorkbook().ActiveSheet.Cells["C3"].Formula = "B3.Designation";

}
[System.Reflection.DefaultMember("FirstName")]
public class Employee : IMedia
{
    public int ID { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Designation { get; set; }
    public string Department { get; set; }
    public string Gender { get; set; }
    public int Age { get; set; }
    public int YearOfJoining { get; set; }
    string IMedia.ContentType => "image/png";
    Stream IMedia.Data
    {
        get
        {
            return
                typeof(FpSpread).Assembly.GetManifestResourceStream("FarPoint.Win.Spread.SpreadResources.EditShape.png");
        }
    }
}

```

VB

```

Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs)
    fpSpread1.AsWorkbook().WorkbookSet.CalculationEngine.CalcFeatures = CalcFeatures.All
    fpSpread1.ActiveSheet.Columns(1, 10).Width = 120
    Dim andreySmith As GrapeCity.CalcEngine.RichValue(Of Employee) = New
    GrapeCity.CalcEngine.RichValue(Of Employee)(New Employee() With {
        .ID = 61371,
        .FirstName = "Andrey",
        .LastName = "Smith",
        .Designation = "Tech Lead",
        .Department = "IT",
        .Gender = "M",
        .Age = 34,
        .YearOfJoining = 2013
    })
    Dim jackShang As GrapeCity.CalcEngine.RichValue(Of Employee) = New
    GrapeCity.CalcEngine.RichValue(Of Employee)(New Employee() With {
        .ID = 37123,
        .FirstName = "Jack",
        .LastName = "Shang",
        .Designation = "Manager",
        .Department = "Sales",
        .Gender = "M",
        .Age = 34,
        .YearOfJoining = 2017
    })
    Dim joanaJordan As GrapeCity.CalcEngine.RichValue(Of Employee) = New
    GrapeCity.CalcEngine.RichValue(Of Employee)(New Employee() With {

```

```

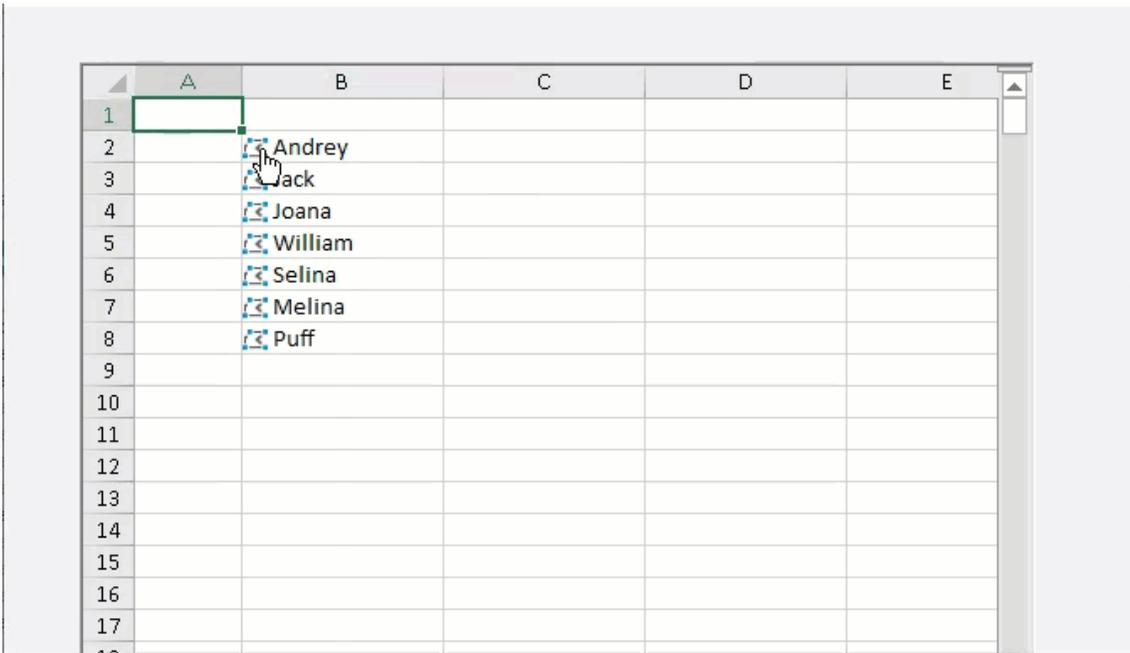
        .ID = 37564,
        .FirstName = "Joana",
        .LastName = "Jordan",
        .Designation = "GraphicsDesigner",
        .Department = "IT",
        .Gender = "F",
        .Age = 28,
        .YearOfJoining = 2016
    })
    Dim williamSmith As GrapeCity.CalcEngine.RichValue(Of Employee) = New
    GrapeCity.CalcEngine.RichValue(Of Employee)(New Employee() With {
        .ID = 28034,
        .FirstName = "William",
        .LastName = "Smith",
        .Designation = "AVP",
        .Department = "HR",
        .Gender = "M",
        .Age = 42,
        .YearOfJoining = 2012
    })
    Dim selinaWing As GrapeCity.CalcEngine.RichValue(Of Employee) = New
    GrapeCity.CalcEngine.RichValue(Of Employee)(New Employee() With {
        .ID = 65134,
        .FirstName = "Selina",
        .LastName = "Wing",
        .Designation = "Technical Engineer",
        .Department = "IT",
        .Gender = "F",
        .Age = 34,
        .YearOfJoining = 2019
    })
    Dim melinaJackson As GrapeCity.CalcEngine.RichValue(Of Employee) = New
    GrapeCity.CalcEngine.RichValue(Of Employee)(New Employee() With {
        .ID = 45978,
        .FirstName = "Melina",
        .LastName = "Jackson",
        .Designation = "Senior S/W Engineer",
        .Department = "IT",
        .Gender = "F",
        .Age = 34,
        .YearOfJoining = 2014
    })
    Dim puffDuplacy As GrapeCity.CalcEngine.RichValue(Of Employee) = New
    GrapeCity.CalcEngine.RichValue(Of Employee)(New Employee() With {
        .ID = 32700,
        .FirstName = "Puff",
        .LastName = "Duplacy",
        .Designation = "Team Lead",
        .Department = "IT",
        .Gender = "M",
        .Age = 36,
        .YearOfJoining = 2012
    })
    fpSpread1.AsWorkbook().ActiveSheet.Cells("B2").Value = andreySmith
    fpSpread1.AsWorkbook().ActiveSheet.Cells("B3").Value = jackShang
    fpSpread1.AsWorkbook().ActiveSheet.Cells("B4").Value = joanaJordan
    fpSpread1.AsWorkbook().ActiveSheet.Cells("B5").Value = williamSmith
    fpSpread1.AsWorkbook().ActiveSheet.Cells("B6").Value = selinaWing
    fpSpread1.AsWorkbook().ActiveSheet.Cells("B7").Value = melinaJackson
    fpSpread1.AsWorkbook().ActiveSheet.Cells("B8").Value = puffDuplacy
    fpSpread1.AsWorkbook().ActiveSheet.Cells("C2").Formula = "B2.[Designation]"
    fpSpread1.AsWorkbook().ActiveSheet.Cells("C3").Formula = "B3.Designation"
End Sub

```

Customizing DataCard PopUp

Users can customize the datacard popup using **FpSpread.ResolveCustomControl** event. Here, the datacard has been customized to show

text "Designation" and on clicking the datacard, it populates the empty cell with the Designation of the Employee.



The code below implements the use-case customizing datacard popup:

C#

```
private void FpSpread1_ResolveCustomControl(object sender, ResolveCustomControlEvents e)
{
    if (e.Type == CustomControlType.RichValueInsert)
    {
        Button button = new Button();
        button.Click += (object sender1, EventArgs buttonEventArgs) =>
        {
            e.Command.Execute("Designation", e.Spread);
        };
        button.Text = "Designation";
        e.Control = button;
    }
    else
    {
        e.Control = e.CreateDefaultControl();
    }
}
```

VB

```
Private Sub FpSpread1_ResolveCustomControl(ByVal sender As Object, ByVal e As
ResolveCustomControlEvents)
    If e.Type = CustomControlType.RichValueInsert Then
        Dim button As Button = New Button()
        button.Click += Function(ByVal sender1 As Object, ByVal buttonEventArgs As EventArgs)
            e.Command.Execute("Designation", e.Spread)
        End Function
        button.Text = "Designation"
        e.Control = button
    Else
        e.Control = e.CreateDefaultControl()
    End If
End Sub
```

Displaying Cell Data

You can display the cell data in a number of ways:

- **Resizing the Data to Fit the Cell**
- **Allowing Cell Data to Overflow**
- **Aligning Cell Contents**
- **Resizing a Cell to Fit the Data**

Resizing the Data to Fit the Cell

You can display all the text in the cell with the shrink to fit option. The font size is reduced if the text is too long for the visible area of the cell.

This property is available for the currency, datetime, mask, number, percent, regular expression, text, or general cell.

The following image shows the difference between a cell with the **ShrinkToFit ('ShrinkToFit Property' in the on-line documentation)** property set to True and a cell with the property set to False.

	A	B	C
1	11240082777		
2			
3		1240082777	
4			

Using Code

1. Create a cell that supports the shrink to fit option such as the regular expression cell.
2. Set the **ShrinkToFit** property.
3. Set the **CellType** property.
4. Type a valid value for the cell such as 11240082777 for the regular expression cell in this example.

Example

This example reduces the font size so the text is displayed in the cell.

C#

```
FarPoint.Win.Spread.CellType.RegularExpressionCellType testcell = new
FarPoint.Win.Spread.CellType.RegularExpressionCellType();
testcell.ShrinkToFit = true;
testcell.RegularExpression = "^\\d{11}$";
fpSpread1.Sheets[0].Cells[0, 0].CellType = testcell;
```

VB

```
Dim testcell As New FarPoint.Win.Spread.CellType.RegularExpressionCellType()
testcell.ShrinkToFit = True
testcell.RegularExpression = "^\\d{11}$"
```

```
fpSpread1.Sheets(0).Cells(0, 0).CellType = testcell
```

Using the Spread Designer

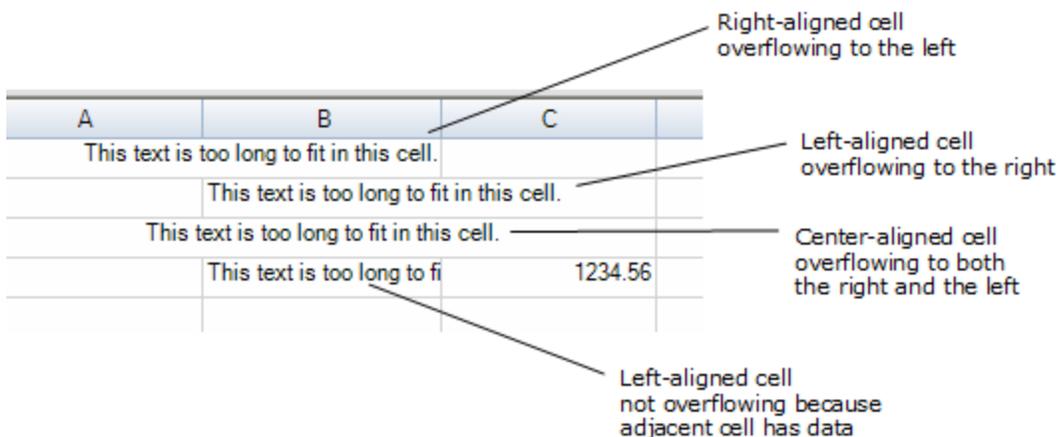
1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the cell type.
3. Expand the **CellType** property and set the **ShrinkToFit** property to True.
4. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Allowing Cell Data to Overflow

You can determine how the contents of a cell or a group of cells overflow into adjoining cells. If you allow cell contents to overflow:

- Left-aligned text in a cell overflows to the adjacent right cell.
- Right-aligned text in a cell overflows to the adjacent left cell.
- Centered text in a cell overflows to both the left and right adjacent cells.

An example of each of these is shown in the following figure.



To set the overflow behavior, use the following members for the overall component (**FpSpread** ('FpSpread Class' in the on-line documentation) class) or the child sheet (**SpreadView** ('SpreadView Class' in the on-line documentation) class):

- **AllowCellOverflow** ('AllowCellOverflow Property' in the on-line documentation)
- **GetMaximumCellOverflowWidth** ('GetMaximumCellOverflowWidth Method' in the on-line documentation)
- **SetMaximumCellOverflowWidth** ('SetMaximumCellOverflowWidth Method' in the on-line documentation)
- **AllowEditOverflow** ('AllowEditOverflow Property' in the on-line documentation)

You can specify the amount of content that will overflow into adjacent cells by specifying the number of pixels of overflow allowed using the **SetMaximumCellOverflowWidth** ('SetMaximumCellOverflowWidth Method' in the on-line documentation) method.

For more information on cell contents alignment, refer to **Aligning Cell Contents**.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.

2. Select (in the **Behavior** category) the **AllowCellOverflow** property or the **AllowEditOverflow** property.
3. Select **True** from the drop-down list to allow cells to overflow, or select **False** to prohibit cells from overflowing.

Using a Shortcut

1. Allow the contents of a set of cells to overflow by setting the **AllowCellOverflow** (**'AllowCellOverflow Property' in the on-line documentation**) property.
2. Specify the maximum amount of content allowed to overflow by calling the **SetMaximumCellOverflowWidth** (**'SetMaximumCellOverflowWidth Method' in the on-line documentation**) method to specify the number of pixels allowed to overflow.

Example

This example code sets the component to allow cells to overflow but only up to the maximum of 130 pixels.

C#

```
fpSpread1.AllowCellOverflow = true;
fpSpread1.SetMaximumCellOverflowWidth(130);
```

VB

```
fpSpread1.AllowCellOverflow = True
fpSpread1.SetMaximumCellOverflowWidth(130)
```

Using Code

1. Allow the contents of a set of cells to overflow by setting the **FpSpread** (**'FpSpread Class' in the on-line documentation**) **AllowCellOverflow** (**'AllowCellOverflow Property' in the on-line documentation**) property.
2. Specify the maximum amount of content allowed to overflow by calling the **SetMaximumCellOverflowWidth** (**'SetMaximumCellOverflowWidth Method' in the on-line documentation**) method to specify the number of pixels allowed to overflow.

Example

This example code sets the child sheet to allow cells to overflow up to a maximum width of 130.

C#

```
FarPoint.Win.Spread.SpreadView sv = fpSpread1.GetRootWorkbook();
sv.AllowCellOverflow = true;
sv.SetMaximumCellOverflowWidth(130);
```

VB

```
Dim sv As FarPoint.Win.Spread.SpreadView = fpSpread1.GetRootWorkbook
sv.AllowCellOverflow = True
sv.SetMaximumCellOverflowWidth(130)
```

Using the Spread Designer

1. Select the Spread component (or select Spread from the pull-down menu).
2. In the property list for the component, in the **Behavior** category, select the **AllowCellOverflow** property or the **AllowEditOverflow** and select the value **True**.

- From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Aligning Cell Contents

You can determine how the contents are aligned in a cell or in a group of cells. In code, simply set the **HorizontalAlignment** ('**HorizontalAlignment Property**' in the on-line documentation) property and the **VerticalAlignment** ('**VerticalAlignment Property**' in the on-line documentation) properties, and make use of the **CellHorizontalAlignment** ('**CellHorizontalAlignment Enumeration**' in the on-line documentation) and **CellVerticalAlignment** ('**CellVerticalAlignment Enumeration**' in the on-line documentation) enumerations. In the Spread Designer, set those properties accordingly. The following figure shows the result of the code in the first example.

Cell A1 is bottom and right aligned

	A	B	C	D
1	align this	normal	normal	normal
2	normal	centered	centered	normal
3	normal	centered	centered	normal
4	normal	normal	normal	normal

Cells B2 to C3 are center aligned

There are additional properties in the **Cell** ('**Cell Class**' in the on-line documentation) class such as **TextIndent** ('**TextIndent Property**' in the on-line documentation) and **CellPadding** ('**CellPadding Property**' in the on-line documentation) that can be used to create extra margins around cell text.

For an explanation of how the alignment affects the overflow of data, refer to **Allowing Cell Data to Overflow**

You can keep the cell alignment when editing with the **AllowEditorVerticalAlign** ('**AllowEditorVerticalAlign Property**' in the on-line documentation) property.

Using the Properties Window

- At design time, in the **Properties** window, select the Spread component.
- Select the **Sheets** property.
- Click the button to display the **SheetView Collection Editor**.
- In the **Members** list, select the sheet in which the cells appear.
- In the properties list, select the **Cells** property and then click the button to display the **Cell, Column, and Row Editor**.
- Select the cells for which you want to set the alignment.
- In the properties list, select the **HorizontalAlignment** property and choose the alignment.
- To set the vertical alignment, select the **VerticalAlignment** property in the properties list and choose the alignment.
- Click **OK** to close the **Cell, Column, and Row Editor**.
- Click **OK** to close the **SheetView Collection Editor**.

Using a Shortcut

Set the **HorizontalAlignment** ('**HorizontalAlignment Property**' in the on-line documentation) property and the **VerticalAlignment** ('**VerticalAlignment Property**' in the on-line documentation) properties for the Cells shortcut object.

Example

This example code sets the horizontal alignment of the first cell (A1) to be right-aligned, the vertical alignment of that cell to be bottom-aligned, and the horizontal alignment and vertical alignment of cells from B2 to C3 to be centered. The preceding figure illustrates the results.

C#

```
fpSpread1.Sheets[0].Cells[0,0].HorizontalAlignment =
FarPoint.Win.Spread.CellHorizontalAlignment.Right;
fpSpread1.Sheets[0].Cells[0,0].VerticalAlignment =
FarPoint.Win.Spread.CellVerticalAlignment.Bottom;
fpSpread1.Sheets[0].Cells[1,1,2,2].HorizontalAlignment =
FarPoint.Win.Spread.CellHorizontalAlignment.Center;
fpSpread1.Sheets[0].Cells[1,1,2,2].VerticalAlignment =
FarPoint.Win.Spread.CellVerticalAlignment.Center;
```

Using Code

Set the **HorizontalAlignment** ('**HorizontalAlignment Property**' in the on-line documentation) property and the **VerticalAlignment** ('**VerticalAlignment Property**' in the on-line documentation) properties for the **Cell** ('**Cell Class**' in the on-line documentation) object.

Example

This example code sets the horizontal alignment for a specific cell and the vertical alignment for a range of cells.

C#

```
FarPoint.Win.Spread.Cell cellA1;
cellA1 = fpSpread1.ActiveSheet.Cells[0, 0];
cellA1.HorizontalAlignment =
FarPoint.Win.Spread.CellHorizontalAlignment.Right;
cellA1.VerticalAlignment =
FarPoint.Win.Spread.CellVerticalAlignment.Bottom;
FarPoint.Win.Spread.Cell cellrange;
cellrange = fpSpread1.ActiveSheet.Cells[1,1,2,2];
cellrange.HorizontalAlignment =
FarPoint.Win.Spread.CellHorizontalAlignment.Center;
cellrange.VerticalAlignment =
FarPoint.Win.Spread.CellVerticalAlignment.Center;
```

Using the Spread Designer

1. In the work area, select the cell or cells for which you want to set the horizontal alignment of the cell contents.
2. In the properties list (in the **Misc** category), select the **HorizontalAlignment** property.
3. Click the drop-down button to display the choices and choose the alignment.
4. Repeat these steps and in the properties list, select **VerticalAlignment** to set the vertical alignment of the cell or cells.
5. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Center Across Selection

You can center-align the contents of the selected cells across columns without merging the cells. This type of horizontal alignment can be set using the **HorizontalAlignment.CenterAcrossSelection** enumeration option from the **GrapeCity.Spreadsheet** namespace.

In order to implement the **CenterAcrossSelection** alignment option, the **AllowCellOverflow** property of **FpSpread** class must be set to true and the **BorderCollapse** property must be set to 'Enhanced'.

	B	C	D	E	F	G	H	I	J	K
		test feature setting								
	test feature setting									
		test feature setting								
	test feature setting									

This example code sets the center across selection for cells:

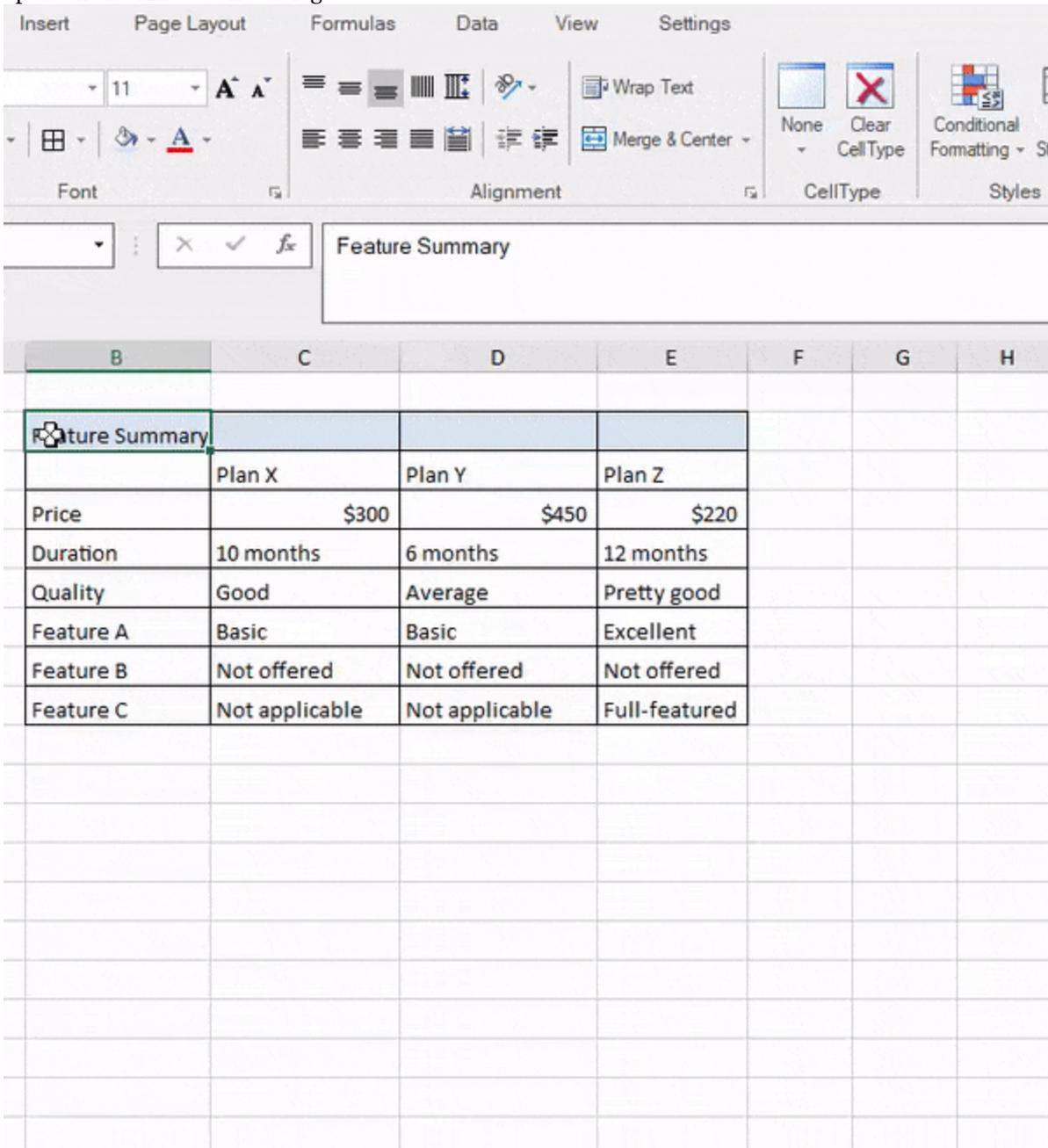
C#

```
fpSpread1.AllowCellOverflow = true;
fpSpread1.BorderCollapse = BorderCollapse.Enhanced;
TestActiveSheet.Cells["C2"].Text = "test feature setting";
TestActiveSheet.Cells["C2:K2"].HorizontalAlignment =
GrapeCity.Spreadsheet.HorizontalAlignment.CenterAcrossSelection;//set for
big range
TestActiveSheet.Cells["C3"].Text = "test feature setting";
TestActiveSheet.Cells["C3"].HorizontalAlignment =
GrapeCity.Spreadsheet.HorizontalAlignment.CenterAcrossSelection;//set for
1 cell
TestActiveSheet.Cells["C4"].Text = "test feature setting";
TestActiveSheet.Cells["C4:D4"].HorizontalAlignment =
GrapeCity.Spreadsheet.HorizontalAlignment.CenterAcrossSelection;//set for
small range start from data cell
TestActiveSheet.Cells["C5"].Text = "test feature setting";
TestActiveSheet.Cells["B5:C5"].HorizontalAlignment =
GrapeCity.Spreadsheet.HorizontalAlignment.CenterAcrossSelection;//set for
small range end by data cell
FarPoint.Win.Spread.Dialogs.BuiltInDialogs.FormatCells(TestActiveSheet.Ce
lls["A1:D5"]).ShowDialog();
```

Using the Spread Designer

1. In the work area, select the range of cells for which you want to set the center across selection.

2. Select the **Spread** option from the selected item list of the properties window.
3. Select the **BorderCollapse** property from the properties list of the **Appearance** category. Set the property to Enhanced.
4. Select the **AllowCellOverflow** property from the properties list of the **Behavior** category and set it to true.
5. Click the Dialog Box Launcher arrow at the bottom right corner of the **Alignment** group of **Home** menu. This opens the **Format Cells** dialog.



6. Select the **Center Across Selection** from the dropdown list of the Horizontal option of **Alignment** tab in the dialog.
7. Click **OK** to apply the changes.

Resizing a Cell to Fit the Data

You can resize the cell based on the length of the data in the cell. The size of the cell with the largest data is called the preferred size.

The SheetView **GetPreferredSize** (**GetPreferredSize Method** in the on-line documentation) method retrieves the preferred size of the specified cell.

This figure shows the result of the example code that resizes the column based on the text in the cells of that column.

	A	B
1	Expand the cell to fit the text.	
2		

Some cell types ignore the size parameter while other cell types use the size parameter. For example, a single line text cell type ignores the size parameter while a word-wrapping multiple-line text cell type uses the size parameter's width property to determine line breaks. Basically, the size parameter's width (or height) is used in determining wrapping breaks for horizontal (or vertical) content. The sheet's **GetPreferredSize** (**GetPreferredSize Method** in the on-line documentation) method passes the cell's current size to the cell renderer's **GetPreferredSize** methods and it assumes that you want multiple-line text cells to expand vertically using the cell's current width. For more information on how cell types display data, refer to **Understanding How Cell Types Display and Format Data**.

Besides getting the height for a row with the **GetPreferredSize** (**GetPreferredSize Method** in the on-line documentation) method, you can get a width for a column using the **GetPreferredSize** (**GetPreferredSize Method** in the on-line documentation) method.

For information on setting an entire row or column of cells based on the size of the data, refer to **Resizing the Row or Column to Fit the Data**.

Using Code

Set the width of the cell to the value of the preferred size to show the entire contents of the cell.

Example

After setting the text in the contents, resize the column width of the cell to match the maximum size of the cell.

C#

```
System.Drawing.Size sz;
fpSpread1.ActiveSheet.SetValue(0, 0, "Expand the cell to fit the text.");
sz = fpSpread1.ActiveSheet.GetPreferredSize(0,0);
fpSpread1.ActiveSheet.Columns[0].Width = sz.Width;
MessageBox.Show("The width of the cell is " + sz.Width.ToString());
```

VB

```
Dim sz As System.Drawing.Size
fpSpread1.ActiveSheet.SetValue(0, 0, "Expand the cell to fit the text.")
sz = fpSpread1.ActiveSheet.GetPreferredSize(0, 0)
fpSpread1.ActiveSheet.Columns(0).Width = sz.Width
MessageBox.Show("The width of the editor is " & sz.Width.ToString())
```

Creating a Span of Cells

You can combine cells to create a span of cells, as shown in the figure below. Creating a span of cells creates one large cell where there had previously been several. For example, if you create a span of cells from cell B2 to cell D3, cell B2

then appears to occupy the space from cell B2 through cell D3.

	A	B	C	D	E
1					
2		These six cells are spanned.			
3					
4					

The component is divided into four parts: sheet corner, column headers, rows headers, and data area. You can create spans within a part, but you can not create a span that goes across parts. For example, you can not span cells in the data area with cells in the row headers and you can not span cells in the column header with the sheet corner. This topic discusses spanning cells in the data area. For more information on creating a span of header cells, refer to **Creating a Span in a Header**.

When you create a span of cells, the data in the first cell in the span (the anchor cell) occupies all the space in the span. When you create a span, the data that was in each of the cells in the span is still in each cell, but not displayed. The data is simply hidden by the span range. If you remove the span from a group of cells, the content of the spanned cells, which previously was hidden, is displayed as appropriate. Create a span of cells by calling the **AddSpanCell ('AddSpanCell Method' in the on-line documentation)** method. The cell types of the cells combined in the span are not changed. The spanned cell takes the type of the left-most cell in the span.

You can return whether a specified cell is in a span of cells with the **GetSpanCell ('GetSpanCell Method' in the on-line documentation)** method.

You can remove a span from a range of cells by calling the **RemoveSpanCell ('RemoveSpanCell Method' in the on-line documentation)** method. You can remove a span range by calling this method, specifying the anchor cell of the span range to remove the range. When you remove a span range, the data that was previously in each of the cells in the span is re-displayed in the cell. The data was never removed from the cell, but simply hidden by the span range.

 **Note:** Spans that are added to a sorted sheet are not shown, and spans will be hidden when the sheet or any part of it is sorted with any sort method other than **SortRange ('SortRange Method' in the on-line documentation)**. Cell ranges that contain spans cannot be sorted with **SortRange ('SortRange Method' in the on-line documentation)**.

Whatever properties you set on the anchor cell are applied to the cell span. This includes setting a cell note, too. If you set a cell note to one of the cells in the span that is not the anchor, the cell note is not displayed.

Call the **GetSpanCell** method to return whether a cell is in a span of cells, and if it is in a span of cells, it returns the **CellRange ('CellRange Class' in the on-line documentation)** object that contains the column and row number of the anchor cell and the number of columns and rows in the span range. This method is called for the currently selected sheet unless you first set the **Sheets** object to specify the sheet with which to work.

If a merged column overlaps a span, then the merged column replaces the span. It is recommended that you do not merge cells that are part of a span. For more information on automatically merging cells with identical content, refer to **Allowing Cells to Merge Automatically**.

For information on the underlying model for spans, refer to **Understanding the Span Model**.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Properties** window on the right side of the **SheetView Collection Editor**, select the **Cells** property for the sheet.
5. Click the button to display the **Cell, Column, and Row Editor**.
6. In the editor, select either the **Row Span** or **Column Span** property and set the number to the number of cells

to span starting from the selected cell. To remove a span, set the value back to 1. The preview on the left side of the editor shows the cells spanned.

7. If you want to apply this change, click **Apply**.
8. Click **OK** to close each editor.

Using a Shortcut

To span cells (or remove spanning) use any of the following members:

- **AddSpanCell**, **GetSpanCell**, and **RemoveSpanCell**
- **AddColumnHeaderSpan** and **AddRowHeaderSpan**

For more information on these properties and methods, refer to the **SheetView** (**'SheetView Class' in the on-line documentation**) class.

Call the **Sheets** object **AddSpanCell** method to span the cells.

Example

This example code defines some content then spans six adjoining cells.

C#

```
// Create some content in two cells.
fpSpread1.ActiveSheet.Cells[1,1].Text = "These six cells are spanned.";
fpSpread1.ActiveSheet.Cells[2,2].Text = "This is text in 2,2.";
// Span six cells including the ones with different content.
fpSpread1.ActiveSheet.AddSpanCell(1, 1, 2, 3);
```

VB

```
' Create some content in two cells.
fpSpread1.ActiveSheet.Cells(1,1).Text = "These six cells are spanned."
fpSpread1.ActiveSheet.Cells(2,2).Text = "This is text in 2,2."
' Span six cells including the ones with different content.
fpSpread1.ActiveSheet.AddSpanCell(1, 1, 2, 3)
```

Using the Spread Designer

1. On the spreadsheet, select the cells to span.
2. Right-click and select **Span** or in the property list (in the **Misc** category), select either the **Row Span** or **Column Span** property and set the number to a value greater than 1 to span cells. To remove a span, set the value back to 1.

The Designer shows the cells spanned.

12							
13							
14							
15							
16							

3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Allowing Cells to Merge Automatically

You can have Spread automatically merge cells between columns or between rows if the cells have the same value based on the policy that you set. The component can automatically combine cells that have the same contents. You might want to do this, for example, when bound to a database, as shown in the figure below where the cells in the Year column merge where the year value is the same, as with 1995 and 2003.

	Title	Year	ISBN-10	Author
8	Applied Differential Equatio	2004	0-0201406-7-3	McKensie
9	Information Systems Archite	1998	0-0207992-0-9	Artherson
10	Information Systems Langu	1995	0-0230081-2-1	Genaldor
11	Information Systems Tutori		0-0230362-0-6	Alphacen
12	Inside Microsoft Visual Stud		0-0237650-8-7	Smith, Ge
13	Patterns in Software Develo	2003	0-0230942-8-1	Hensly
14	Structured C for Engineers	2005	0-0230942-1-2	Fabriconn
15	Software Development in a		0-0253774-2-3	Altimessir

Unlike spanning cells, merging is an automatic feature. You tell the component which columns and rows allow cells to be combined automatically, and any cells within that set that have the same contents are combined for you.

For more information on spanning cells, refer to **Creating a Span of Cells**.

The policy you set determines how the component handles merging, as follows:

- If the merge policy is set to None, cells within a row or column are not merged.
- If the merge policy is set to Always, cells within a row or column are merged when the cells have the same values.
- If the merge policy is set to Restricted, cells within a row or column are merged when the cells have the same values and the corresponding cells in the previous row or column also have the same value. For example, suppose cells A1:A8 contain {a; a; b; b; b; c; c} and cells B1:B8 contain {1; 1; 1; 1; 2; 2; 2}. If column B's merge policy is Always, the cells in column B are merged into two blocks B1:B4 and B5:B8. If column B's merge policy is Restricted then the cells in column B are merged into four blocks B1:B2, B3:B4, B5:B6, and B7:B8.

You can have the cells in the specified row or column combine the cells automatically, or only combine them if the cells to their left (in columns) or above them (in rows) are merged. Typically, if you set the merge policy on several adjacent rows or columns, then you would use Always on the first row or column and Restricted on the remaining rows or columns.

Merged cells take on the properties of the top-left merged cell. For example, if the top-left merged cell has a blue background color, the cells that merge with it display the same background color.

Merged cells do not lose their data; it is simply hidden by the merge. If you remove the merge, the data appears in each cell that was in the merge. You can edit a cell that is merged with another cell. When you double-click the cell to turn edit mode on, the contents of the cell appear in the cell for you to edit them. When you leave edit mode, if the contents of the cell are no longer identical to the cell or cells with which it was previously merged, the cells are no longer displayed as merged.

Cells that are different cell types but have the same contents can merge. For example, a date cell might contain the contents "01/31/02" and the adjacent edit cell might contain the same contents; if the column containing the cells is set to merge, the cells will merge. If the contents change or the merge is removed, the cells maintain their cell types as well as their data.

To set cells to be merged if they have the same value, use the following members:

- **GetColumnMerge** ('GetColumnMerge Method' in the on-line documentation) and **SetColumnMerge** ('SetColumnMerge Method' in the on-line documentation)
- **GetRowMerge** ('GetRowMerge Method' in the on-line documentation) and **SetRowMerge** ('SetRowMerge Method' in the on-line documentation)

- **GetMergePolicy** ('GetMergePolicy Method' in the on-line documentation) and **SetMergePolicy** ('SetMergePolicy Method' in the on-line documentation)

For more information on these members, refer to the **SheetView** ('SheetView Class' in the on-line documentation) class (or the **Row** ('Row Class' in the on-line documentation) or **Column** ('Column Class' in the on-line documentation) class) or the **DefaultSheetAxisModel** ('DefaultSheetAxisModel Class' in the on-line documentation) of the **FarPoint.Win.Spread.Model** ('FarPoint.Win.Spread.Model Namespace' in the on-line documentation) namespace.

Using a Shortcut

Use the **SetColumnMerge** ('SetColumnMerge Method' in the on-line documentation) or **SetRowMerge** ('SetRowMerge Method' in the on-line documentation) method for the **Sheets** or **ActiveSheet** shortcut object.

Example

This example code sets the row and column merge policies for all rows and all columns.

C#

```
fpSpread1.Sheets[0].SetRowMerge(-1, FarPoint.Win.Spread.Model.MergePolicy.Always);  
fpSpread1.Sheets[0].SetColumnMerge(-1, FarPoint.Win.Spread.Model.MergePolicy.Always);
```

VB

```
fpSpread1.Sheets(0).SetRowMerge(-1, FarPoint.Win.Spread.Model.MergePolicy.Always)  
fpSpread1.Sheets(0).SetColumnMerge(-1, FarPoint.Win.Spread.Model.MergePolicy.Always)
```

Using Code

Set the **SetColumnMerge** ('SetColumnMerge Method' in the on-line documentation) or **SetRowMerge** ('SetRowMerge Method' in the on-line documentation) method for a **SheetView** ('SheetView Class' in the on-line documentation) object.

Example

This example code merges columns and rows.

C#

```
FarPoint.Win.Spread.SheetView Sheet0;  
Sheet0 = fpSpread1.Sheets[0];  
Sheet0.SetRowMerge(-1, FarPoint.Win.Spread.Model.MergePolicy.Always);  
Sheet0.SetColumnMerge(-1, FarPoint.Win.Spread.Model.MergePolicy.Always);
```

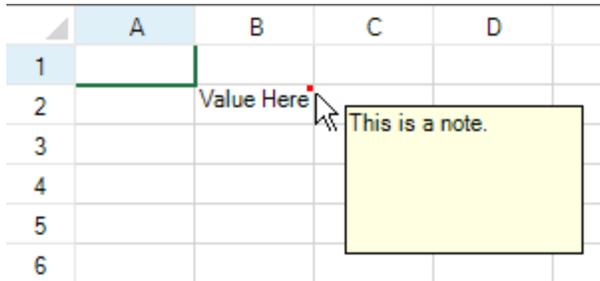
VB

```
Dim Sheet0 As FarPoint.Win.Spread.SheetView  
Sheet0 = fpSpread1.Sheets(0)  
Sheet0.SetRowMerge(-1, FarPoint.Win.Spread.Model.MergePolicy.Always)  
Sheet0.SetColumnMerge(-1, FarPoint.Win.Spread.Model.MergePolicy.Always)
```

Adding a Note to a Cell

You can add a note to a cell or range of cells. The note may contain text such as a comment, a question, or documentation describing the origin of the cell's value. Each cell with a note attached displays a cell note indicator (by

default a small red square) in the upper right corner of the cell. When the pointer is over a cell indicator of a cell that has a note, the note text displays in a box next to the cell. Alternatively, you can set the cell notes to always be displayed, not just when the pointer moves over the indicator. For cell notes that are set as popup notes, they are displayed in a similar manner as text tips. When the pointer is over the cell note indicator, the cell note text appears. This is illustrated in the following figure.

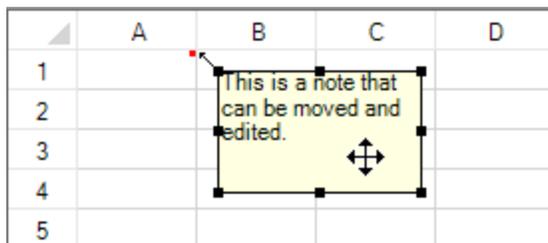


The red square in the upper right corner of the cell indicates that a note is available for that cell. An example is shown in the following figure. You can use the **CellNoteIndicatorVisible** (**'CellNoteIndicatorVisible Property' in the on-line documentation**) property to hide the cell note indicator when the pointer is over the cell note indicator. You can use the **NoteIndicatorPosition** (**'NoteIndicatorPosition Property' in the on-line documentation**) property for the cell to set the location of the note.



Customizing the Cell Note Behavior

You can allow notes to remain displayed, as if they were sticky notes. In this case they appear in a rectangle next to the cell with an expandable line that attaches the note to the cell, allowing the note to be moved by the user. An example of a sticky note that has been selected is shown in the following figure. The cell **NoteStyle** (**'NoteStyle Property' in the on-line documentation**) property must use the **NoteStyle** (**'NoteStyle Enumeration' in the on-line documentation**) enumeration to allow this. The sticky note in this case is a shape that can be moved.



To move the note, press the left mouse button when the pointer is on the note to select it, drag it to the destination, and release the mouse button to place it. The line from the cell note indicator to the sticky note stretches to accommodate any placement of the note.

You can allow the user to edit cell notes if the notes are always shown. To allow the user to edit it, set the **AllowNoteEdit** (**'AllowNoteEdit Property' in the on-line documentation**) property for the sheet, which sets all sticky notes on that sheet to be editable by the user.

The cell note can contain an extra bit of human readable information for the end user; you can also allow the user to attach their own information in cell notes. The information can be whatever is useful to the end user. For example, the end user might use the cell note to indicate the original source of the cell value (cell note = "Value as obtained from an article in the July issue of our corporate magazine").

You can further customize the use of notes:

- automatically size cell notes based on contents
- customize locations of cell notes
- make sticky notes so they stay where placed
- customize the note indicator (see section below)
- print cell notes

There are additional classes that can be used to customize the appearance of the cell note. Use the [StickyNoteStyleInfo](#) class for notes.

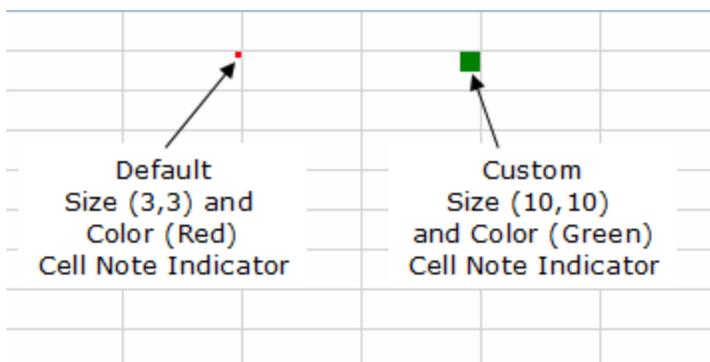
Understanding Limitations

There are some limitations to the use and display of cell notes:

- The note does not display when the **NoteStyle** ('**NoteStyle Property**' in the on-line documentation) property of the cell object is set to hidden.
- The note does not display in certain cell types when the **IsReservedLocation** method for that cell type is set to true. This may occur in a check box cell, or in a combo box cell that is not editable, or when a pointer is over a link in a hyperlink cell.
- The cell note indicator does not appear when the cell is in edit mode.
- The cell note of an anchor cell is displayed in a cell span, but the cell notes of any other cell in the span are not displayed.
- Use caution in choosing a red color as the background for a cell that could contain a red cell note. The cell note indicator could be invisible against a red background.

Customizing the Cell Note Indicator

You can change the size and the color of the cell note indicator. The default size of the cell note indicator is a 3x3 square but you can modify the width or the height of the note indicator to any positive integer value. The default color of the cell note indicator is red but you can assign any color value to it. The following figure shows the indicator using default values and a custom indicator using custom values. The custom values are set using the **NoteIndicatorColor** ('**NoteIndicatorColor Property**' in the on-line documentation) and **NoteIndicatorSize** ('**NoteIndicatorSize Property**' in the on-line documentation) properties.



For more information on notes, refer to the **Note** ('**Note Property**' in the on-line documentation) property in the **Cell** ('**Cell Class**' in the on-line documentation) class.

For information about printing cell notes, refer to **Printing a Sheet with Cell Notes**.

Using the Properties Window

1. At design time, in the **Properties** window, select the **Spread** component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.

4. In the **Members** list, select the sheet in which the cells appear.
5. In the properties list, select the **Cells** property and then click the button to display the **Cell, Column, and Row Editor**.
6. Select the cells for which you want to set the note.
7. In the properties list, select the **Note** property and type the note text.
8. Click **OK** to close the **Cell, Column, and Row Editor**.
9. Click **OK** to close the **SheetView Collection Editor**.

Using a Shortcut

1. Set the **Note** property for the cells or range of cells in the sheet of the Spread component.
2. Set the properties for customizing the note, including the note indicator and note position and display.

Example

This example code sets an editable cell note for a range of cells and sets the color of the cell note indicator to green (from the default red).

C#

```
fpSpread1.Sheets[0].AllowNoteEdit = true;
fpSpread1.Sheets[0].Cells[1, 1, 3, 3].Note = "test";
fpSpread1.Sheets[0].Cells[1, 1, 3, 3].NoteIndicatorColor = Color.Green;
fpSpread1.Sheets[0].Cells[1, 1, 3, 3].NoteStyle =
FarPoint.Win.Spread.NoteStyle.StickyNote;
```

VB

```
fpSpread1.Sheets(0).AllowNoteEdit = True
fpSpread1.Sheets(0).Cells(1, 1, 3, 3).Note = "test"
fpSpread1.Sheets(0).Cells(1, 1, 3, 3).NoteIndicatorColor = Color.Green
fpSpread1.Sheets(0).Cells(1, 1, 3, 3).NoteStyle =
FarPoint.Win.Spread.NoteStyle.StickyNote
```

Example

This example code defines a range of cells and then sets the note for that range.

C#

```
FarPoint.Win.Spread.Cell range1;
range1 = fpSpread1.ActiveSheet.Cells[1, 1, 3, 3];
range1.Value = "Value Here";
range1.Note = "This is the note that describes the value.";
```

VB

```
Dim range1 As FarPoint.Win.Spread.Cell
range1 = fpSpread1.ActiveSheet.Cells(1, 1, 3, 3)
range1.Value = "Value Here"
range1.Note = "This is the note that describes the value."
```

Using the Spread Designer

1. In the work area, select the cell or cells for which you want to set the notes to display.

2. In the properties list (in the **Misc** group), select the **Note** property and type in the text of the note. (Or select the **Cells** property and click on the button to call up the **Cell, Column, and Row** editor and select the cells in that editor.)
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Adding a Tag to a Cell

You can add an application tag to a cell or range of cells. If you prefer, you can associate data with any cell in the spreadsheet, or the cells in a column, a row, or the entire spreadsheet. The string data can be used to interact with a cell or to provide information to the application you create. The cell data, or cell tag, is similar to item data you can provide for the spreadsheet, columns, or rows.

The cell tag is for the application much like the cell note is for the end user. The cell note contains an extra bit of human-readable information that is useful to the end user. The cell tag allows the application to attach an extra bit of computer-readable information to a cell. The information can be whatever is useful to the application. For example, suppose the application is manually populating an unbound sheet with values from a DataTable. The application could use the cell tag to indicate the **DataRow** in the DataTable from which the cell value was obtained (cell tag = DataRow). As another example, the application could use the cell tag as a dirty flag (cell tag = True indicates that the cell needs to be processed).

Since the cell tag is declared as an object, it is very flexible; it can be a number, a boolean, a string, or an instance of a class.

For more information on tags, refer to the **Tag ('Tag Property' in the on-line documentation)** property in the **Cell ('Cell Class' in the on-line documentation)** class and the **GetCellFromTag ('GetCellFromTag Method' in the on-line documentation)** method in the **SheetView ('SheetView Class' in the on-line documentation)** class.

Using a Shortcut

Set the **Tag ('Tag Property' in the on-line documentation)** property for the cells in the sheet of the Spread component.

Example

This example code sets the **Tag ('Tag Property' in the on-line documentation)** property for a range of Cell objects.

C#

```
fpSpread1.Sheets[0].Cells[1, 1, 3, 3].Tag = "This is the tag that describes the value.";
fpSpread1.Sheets[0].Cells[1, 1, 3, 3].Value = "Value Here";
```

VB

```
fpSpread1.Sheets(0).Cells(1, 1, 3, 3).Tag = "This is the tag that describes the value."
fpSpread1.Sheets(0).Cells(1, 1, 3, 3).Value = "Value Here"
```

Using Code

Set the **Tag ('Tag Property' in the on-line documentation)** property for the **Cell ('Cell Class' in the on-line documentation)** object for a range of cells.

Example

This example code sets the **Tag** (**'Tag Property' in the on-line documentation**) property for a range of Cell objects.

C#

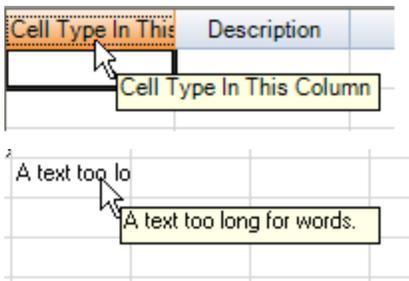
```
FarPoint.Win.Spread.Cell range1;
range1 = fpSpread1.ActiveSheet.Cells[1, 1, 3, 3];
range1.Value = "Value Here";
range1.Tag = "This is the tag that describes the value.";
```

VB

```
Dim range1 As FarPoint.Win.Spread.Cell
range1 = fpSpread1.ActiveSheet.Cells(1, 1, 3, 3)
range1.Value = "Value Here"
range1.Tag = "This is the tag that describes the value."
```

Displaying Text Tips in a Cell

For cells that are too small to display all the text in the cell, you can allow the end user to see the contents in a text tip. You can set the policy and location for text tips for a cell or range of cells. Text tips can be displayed for data cells or header cells. When the pointer is over such a cell, the text tip displays.



To display a text tip over a combo box cell, you would need to turn change the default behavior. By default, the combo box cell captures the mouse when moving over the cell, so it can go into edit mode on the first click and drop-down the list. You can either turn off this behavior, by setting the **Editable** (**'Editable Property' in the on-line documentation**) property to True or you can create a custom cell type and override the **IsReservedLocation** method to return nothing for the parts of the cell that you want the text tip to show over and return an instance of the cell type in the parts of the cell for which you want the click to open the list.

For more information refer to the **TextTipAppearance** (**'TextTipAppearance Property' in the on-line documentation**) property, **TextTipDelay** (**'TextTipDelay Property' in the on-line documentation**) property, and **TextTipPolicy** (**'TextTipPolicy Property' in the on-line documentation**) property.

Refer also to the **TextTipFetchEventArgs** (**'TextTipFetchEventArgs Class' in the on-line documentation**) class and the **TextTipFetch** (**'TextTipFetch Event' in the on-line documentation**) event and **OnTextTipFetch** (**'OnTextTipFetch Method' in the on-line documentation**) method.

For information on scroll bar tips, refer to the section on setting scroll bar tips in **Customizing the Scroll Bars**.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select (in the **Behavior** group) the **TextTipPolicy** property.
3. Click the drop-down arrow to display the choices and choose a value.
4. If you want to set the delay, select the **TextTipDelay** property and type in a value.

Using Code

1. Set the **TextTipPolicy** (**'TextTipPolicy Property' in the on-line documentation**) property for the Spread component.
2. If you want to set a delay, set the **TextTipDelay** (**'TextTipDelay Property' in the on-line documentation**) property.

Example

This example creates a new control, sets whether to display text tips, the location of the tips, and how long to wait before the text tip is shown.

C#

```
FarPoint.Win.Spread.FpSpread fpSpread1 = new FarPoint.Win.Spread.FpSpread();
FarPoint.Win.Spread.SheetView shv = new FarPoint.Win.Spread.SheetView();
fpSpread1.Location = new Point(10, 10);
fpSpread1.Height = 200;
fpSpread1.Width = 400;
Controls.Add(fpSpread1);
fpSpread1.Sheets.Add(shv);
fpSpread1.ActiveSheet.SetValue(0, 0, "TestTextTip");
fpSpread1.TextTipPolicy = FarPoint.Win.Spread.TextTipPolicy.Floating;
fpSpread1.TextTipDelay = 1000;
MessageBox.Show("Place the pointer over the text to see the text tip.", "",
MessageBoxButtons.OK);
```

VB

```
Dim fpSpread1 As New FarPoint.Win.Spread.FpSpread()
Dim shv As New FarPoint.Win.Spread.SheetView()
fpSpread1.Location = New Point(10, 10)
fpSpread1.Height = 200
fpSpread1.Width = 400
Controls.Add(fpSpread1)
fpSpread1.Sheets.Add(shv)
fpSpread1.ActiveSheet.SetValue(0, 0, "TestTextTip")
fpSpread1.TextTipPolicy = FarPoint.Win.Spread.TextTipPolicy.Floating
fpSpread1.TextTipDelay = 1000
MessageBox.Show("Place the pointer over the text to see the text tip.", "",
MessageBoxButtons.OK)
```

Using the Spread Designer

1. Select the Spread component (or select **Spread** from the pull-down menu).
2. In the property list, in the **Behavior** category, select the **TextTipPolicy** property.
3. Click the drop-down arrow to display the choices and choose a value.
4. If you want to set the delay, select the **TextTipDelay** property and type in a value.
5. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Working with Cell Format Strings

Spread for WinForms enables you to format the data displayed in a cell. You can format the values in cells as fractions, currency, percentages, decimals, scientific data, date, time, and so on. The Spread API uses [Excel's number formats](#) to

display various cell number formats.

The following formats are supported by Spread for WinForms:

- **General Format**
- **Fraction Format**
- **Number Format**
- **Percentage Format**
- **Scientific Format**
- **Currency Format**
- **Time Format**
- **Date Format**
- **DateTime Format**
- **Accounting Format**
- **Color Format**
- **Text Format**
- **Special Format**

General Format

This format does not have any specific number format. Numbers that are formatted with the General format are displayed just the way they are typed in the cell.

However, this format automatically rounds the numbers with decimals if it exceeds a specific character limit in the cell or if the cell is not wide enough to show the entire number.

One of the following scenarios can be observed when inputting long numbers with General format:

- **With Decimals:** If a number is longer than 11 characters including the decimal place, then the decimals are rounded to show a maximum of 11 characters.
For example, 123456.7891234 is rounded off to 123456.7891
- **Without Decimals:** If a number is longer than 11 characters and there are no decimals, the number is changed to Scientific format.
For example, 123451234512 is rounded off to 1.23451E+11

Users can also observe the following behavior when reducing column widths of cells with General format applied:

- Decimals are rounded to show only the number that will fit the column.
- The number will be rounded to an integer if there is no room for decimals.
- The number will change to Scientific format if the rounded integer does not fit.
- The cell will display number signs if the number can't be displayed in Scientific format.

 **Note:** This behavior can be disabled by using formats other than "General" like Number and Accounting.

Fraction Format

This format displays numbers as fractions like mixed number fractions or other types of fractions in two different layouts - # ?/? and # ??/?. For this format, Spread for WinForms allows you to select either number of decimal places to display the result in or the nearest place to round off the result. In case you enter data in improper fractions, it will be auto-calculated to the integer part and proper fraction. To enter a negative fraction value, use the (-) sign before entering the mixed number or input the mixed number into parenthesis "()".

Code Format	Input Value	Display Value
# ?/?	8.333	8 1/3
# ??/??	0.846	11/13

Code Format**Input Value****Display Value**

The following code example shows how to set fraction format to a cell containing a value.

C#

```
var worksheet = fpSpread1_Sheet1.AsWorksheet();

worksheet.SetValue(0, 0, 8.333);
worksheet.Cells["A1"].NumberFormat = "# ?/?";
```

Visual Basic

```
Dim worksheet As GrapeCity.Spreadsheet.IWorksheet = FpSpread1_Sheet1.AsWorksheet()

worksheet.SetValue(0, 0, 8.333)
worksheet.Cells("A1").NumberFormat = "# ?/?"
```

Number Format

This format displays data in Number format in two different layouts - #,##0 and #,##0.00. You can display data with thousand separator and decimal places by checking the "Use 1000 Separator (,)" option and mentioning the number of decimal places to display. To enter a negative number value, use the (-) sign before or input the number into parenthesis "()".

Code Format**Input Value****Display Value**

#,##0

1231

1,231

#,##0.00

4561

4,561.00

The following code example shows how to set the number format to a cell containing a value.

C#

```
var worksheet = fpSpread1_Sheet1.AsWorksheet();

worksheet.SetValue(0, 0, 1234);
worksheet.Cells["A1"].NumberFormat = "#,##0";
```

Visual Basic

```
Dim worksheet As GrapeCity.Spreadsheet.IWorksheet = FpSpread1_Sheet1.AsWorksheet()

worksheet.SetValue(0, 0, 1234)
worksheet.Cells("A1").NumberFormat = "#,##0"
```

Percentage Format

This format displays data in Percentage format in two different layouts - 0% and 0.00%. This format multiplies the cell value by 100 and displays the result with a % symbol. You can set the number of decimal places to be displayed in the result. To enter a negative number value, use the (-) sign before the number or input the number into parenthesis "()".

Code Format**Input Value****Display Value**

0%

0.05

5%

0.00%

0.8

80.00%

The following code example shows how to set percentage format to a cell containing a value.

C#

```
var worksheet = fpSpread1_Sheet1.AsWorksheet();

worksheet.SetValue(0, 0, 0.05);
worksheet.Cells["A1"].NumberFormat = "0%";
```

Visual Basic

```
Dim worksheet As GrapeCity.Spreadsheet.IWorksheet = FpSpread1_Sheet1.AsWorksheet()

worksheet.SetValue(0, 0, 0.05)
worksheet.Cells("A1").NumberFormat = "0%"
```

Scientific Format

This format displays data in Scientific format in only one layout - 0.00E+00. The scientific format displays numbers in an exponential notation by replacing parts of the number with E+n; where “E” stands for the exponent that multiplies the preceding number by 10 to the nth power. To display a number in scientific (exponential) format, you need to enter numbers in the form "mEn where coefficient m refers to any real number, while the exponent n is an integer that corresponds to the number of places that the decimal point was moved. You can replace the character E by E+; e by e+; and E- by e-. To enter a negative number value, use the (-) sign before the number or input the number into parenthesis "()".

Code Format

0.00E+00

Input Value

12345678901

Display Value

1.23E+10

The following code example shows how to set scientific format to a cell containing a value.

C#

```
var worksheet = fpSpread1_Sheet1.AsWorksheet();

worksheet.SetValue(0, 0, 12345678901);
worksheet.Cells["A1"].NumberFormat = "0.00E+00";
```

Visual Basic

```
Dim worksheet As GrapeCity.Spreadsheet.IWorksheet = FpSpread1_Sheet1.AsWorksheet()

worksheet.SetValue(0, 0, 12345678901)
worksheet.Cells("A1").NumberFormat = "0.00E+00"
```

Currency Format

This format displays data in Currency format in two different layouts - \$#,##0_);[Red](\$#,##0) and \$#,##0.00_);[Red](\$#,##0.00). The position of the currency symbol is based on the default language of the MS Office programs. You need to enter the correct symbol position of the currency to format cells with the currency format. To enter a negative fraction value, use the (-) sign before entering the mixed number or input the mixed number into parenthesis "()".

Code Format

\$#,##0_);[Red](\$#,##0)

\$#,##0.00_);[Red](\$#,##0.00)

Input Value

-1235

4597

Display Value

(\$1,235)

\$4,597.00

The following code example shows how to set currency format to a cell containing a value.

C#

```
var worksheet = fpSpread1_Sheet1.AsWorksheet();

worksheet.SetValue(0, 0, -1235);
worksheet.Cells["A1"].NumberFormat = "$#,##0_);[Red]($#,##0)";
```

Visual Basic

```
Dim worksheet As GrapeCity.Spreadsheet.IWorksheet = FpSpread1_Sheet1.AsWorksheet()

worksheet.SetValue(0, 0, -1235)
worksheet.Cells("A1").NumberFormat = "$#,##0_);[Red]($#,##0)";
```

Time Format

This format displays data in Time format in six different layouts - h:mm tt, h:mm:ss tt, H:mm, H:mm:ss, [h]:mm:ss and mm:ss.o. Only integer values are accepted for the hours, minutes, and seconds. To enter a time value in either Japanese, Chinese or Korean language, combine the value with the time text like 時 and 分.

Code Format	Input Value	Display Value
h:mm tt	21:05	9:05 tt
h:mm:ss tt	7:49:15	7:49:15 tt
h:mm	17:25	17:25
h:mm:ss	3:19	3:19:00
[h]:mm:ss	236:34	20:34:00
mm:ss.o	5:05	05:00.0

The following code example shows how to set the time format to a cell containing a value.

C#

```
var worksheet = fpSpread1.ActiveSheet.AsWorksheet();
worksheet.SetValue(0, 0, DateTime.Now);
worksheet.Cells["A1"].NumberFormat = "h:mm:ss";
```

Visual Basic

```
Dim worksheet As GrapeCity.Spreadsheet.IWorksheet = FpSpread1_Sheet1.AsWorksheet()

worksheet.SetValue(0, 0, DateTime.Now)
worksheet.Cells("A1").NumberFormat = "h:mm:ss"
```

Date Format

This format displays data in Date format in four different layouts - m/d/yyyy, d-mmm-yy, d-mmm, and mmm-yy. To use this format, you need to enter values in at least one combination - date and month, date and year, or complete value of date, month, and year. When you enter values only for date and month, the value of the year is automatically set to the current year. As a separating character, Spread for WinForms supports both (-) sign and (/) sign. You can use these separating characters in any combination.

The data values for the date, month, and year can be entered based on the following table:

Months	m	1-12
Months	mm	01-12
Months	mmm	Jan-Dec
Months	mmmm	January-December
Days	d	1-31
Days	dd	01-31
Years	yy	00-99
Years	yyyy	1900-9999

If you enter any year from 0-29, the cell value is automatically formatted to 2000-2029. Similarly, if you enter any year from 30-99, the value is formatted to 1930-1999. You can enter text value for a month in both upper case and low case. To enter date value in either Japanese, Chinese or Korean language, combine the value with the date text like 時 and 分.

Spread for WinForms also supports the use of [DBNumX] modifier to display data in East Asia numeric format. For example, in Japanese locale, if you use "[DBNum1][\$-411]d/mm/yyyy" format with the value 43413, the formatted text will be "九/十一/二〇一八".

Code Format	Input Value	Display Value
yyyy/m/d	2019/02/20	2019/2/20
yyyy"年"m"月"	2019/02/20	2019年2月
m"月"d"日"	2019/02/20	2月20日

The following code example shows how to set the date format to a cell containing a value.

C#

```
var worksheet = fpSpread1_Sheet1.AsWorksheet();

worksheet.Cells["A1"].Text = "1/1/2021";
worksheet.Cells["A1"].NumberFormat = "dd-mm-yyyy";
```

Visual Basic

```
Dim worksheet As GrapeCity.Spreadsheet.IWorksheet = FpSpread1_Sheet1.AsWorksheet()

worksheet.Cells("A1").Text = "1/1/2021"
worksheet.Cells("A1").NumberFormat = "dd-mm-yyyy"
```

DateTime Format

This format displays data in DateTime format, with only one layout - m/d/yyyy h:mm. When you enter the date value combined with the time value, the data is automatically formatted in DateTime format (m/d/yyyy h:mm). The value of a date can be placed before or after the time value. To enter a value for time with the format "hour:" or "hour:minute", the date value needs to be combined with day, month, and year. If the data only consists of a date value, it will be formatted as a Date format.

Code Format	Input Value	Display Value
yyyy/m/d h:mm	2019/02/20 12:30:00	2019/2/20 12:30

The following code example shows how to set the datetime format to a cell containing a value.

C#

```
var worksheet = fpSpread1_Sheet1.AsWorksheet();

worksheet.Cells["A1"].Text = "2019/02/20 12:30:00";
worksheet.Cells["A1"].NumberFormat = "yyyy/m/d h:mm";
```

Visual Basic

```
Dim worksheet As GrapeCity.Spreadsheet.IWorksheet = FpSpread1_Sheet1.AsWorksheet()

worksheet.Cells("A1").Text = "2019/02/20 12:30:00"
worksheet.Cells("A1").NumberFormat = "yyyy/m/d h:mm"
```

Accounting Format

This format displays data in Accounting format in two different layouts - `*$ #,##0` and `*$ #,##0.00`. In this format, the currency symbols and decimal points are aligned in a column. This is implemented using an asterisk (*) symbol to denote repeat characters. By default, code formats use "*" (asterisk with a space after) to enter spaces in between the currency symbol and the value, but you can replace " " (space) with any other character.

Code Format	Input Value	Display Value
<code>*\$ #,##0</code>	-1513	-\$ 1,513
<code>*\$ #,##0.00</code>	2583	\$ 2,583.00

The following code example shows how to set the accounting format to a cell containing a value.

C#

```
var worksheet = fpSpread1_Sheet1.AsWorksheet();

worksheet.SetValue(0, 0, -1513);
worksheet.Cells["A1"].NumberFormat = "$* #,##0";
```

Visual Basic

```
Dim worksheet As GrapeCity.Spreadsheet.IWorksheet = FpSpread1_Sheet1.AsWorksheet()

worksheet.SetValue(0, 0, -1513)
worksheet.Cells("A1").NumberFormat = "$* #,##0"
```

Color Format

The Color Format displays data based on the color criteria that affects the `foreColor`.

	A	B	C	D	E	F
1	100.0	200.0	300.0	400.0	500.0	600.0
2						

It supports color string names as well as color indices:

- 8 Named colors: [Black] [Blue] [Cyan] [Green] [Magenta] [Red] [White] [Yellow]
- 56 Indexed colors: [Color1][Color2]...[Color56]

While specifying color formats in code, the name of the color comes first in the code and is enclosed in square brackets. Also, to show that the number format will be applied only if a specified condition is met, the criteria is enclosed in square brackets.

The condition will consist of a comparison operator and a value. For instance, the following number format will display numbers that are less than 50 in Yellow font and numbers that are greater than or equal to 50 in Magenta font.

```
[Yellow] [<50]; [Magenta] [>=50]
```

The following code example shows how to set color formatting by index to change the color of the cells according to the value range.

C#

```
var worksheet = fpSpread1_Sheet1.AsWorksheet();

worksheet.SetValue(0, 0, 100);
worksheet.SetValue(0, 1, 200);
worksheet.SetValue(0, 2, 300);
worksheet.SetValue(0, 3, 400);
worksheet.SetValue(0, 4, 500);
worksheet.SetValue(0, 5, 600);

worksheet.Range("A1:F1").NumberFormat = "[color44] [<300]0.0; [color3] [>400]0.0; [color45]0.0";
```

Visual Basic

```
Dim worksheet As GrapeCity.Spreadsheet.IWorksheet = FpSpread1_Sheet1.AsWorksheet()

worksheet.SetValue(0, 0, 100)
worksheet.SetValue(0, 1, 200)
worksheet.SetValue(0, 2, 300)
worksheet.SetValue(0, 3, 400)
worksheet.SetValue(0, 4, 500)
worksheet.SetValue(0, 5, 600)

worksheet.Range("A1:F1").NumberFormat = "[color44] [<300]0.0; [color3] [>400]0.0; [color45]0.0"
```

Text Format

This format displays data in Text format. In this format, the value of the cells is treated as text even when a number is entered. The result is displayed in the cell exactly as the data is entered.

Special Format

This format displays data in Special format in four different layouts - Zip Code, Zip Code + 4, Phone Number, and Social Security Number.

Working with Pattern and Gradient Fill Effects

Spread for WinForms allows users to import and export the pattern and gradient fill effects applied to the cells of a worksheet. With extensive support for saving progressive color shades and distinct patterns to an excel file, you can use different fill effects like shadow effect, three-dimensional color effect and reflection effect while working with spreadsheets.

A worksheet with pattern fill and gradient fill applied to the cells is shown in the screenshot shared below.

	A	B	C	D
1				
2				
3				
4				
5				
6				
7				
8				

While importing an excel file with pattern fill or gradient fill effect, the visual appearance will remain intact even when the user is using old cell types (provided the default style settings are not modified). Also, the pattern fill effects and gradient fill effects can be exported without any changes when the worksheet is set to the new flatten style mode.

Further, if you are opening or saving an excel file with the color scale conditional formatting rule and pattern fill applied to the cells of the worksheet, the specified color will blend with the pattern fill of the cell instead of replacing it. In such a scenario, the background color of the pattern fill is replaced by the color of the color scale rule.

The following limitations should be considered while applying pattern fill and gradient fill effects.

- Users can set the pattern and gradient fill effects only for the new default cell type. For all other cell types, fill effects are displayed as solid fill only.

While importing the xlsx file, if the user is using the previous style system, the pattern fill and gradient fill effects won't be displayed. In order to see the pattern and gradient fill effect in the imported xlsx file, you can use the code shown below that enables the flatten style system.

C#

```
// Enable the flatten style system only
var fpSpread1 = new FpSpread(LegacyBehavior.All & ~LegacyBehaviors.Style);
// Now, import the excel file
fpSpread1.OpenExcel("MyExcelFile.xlsx");
```

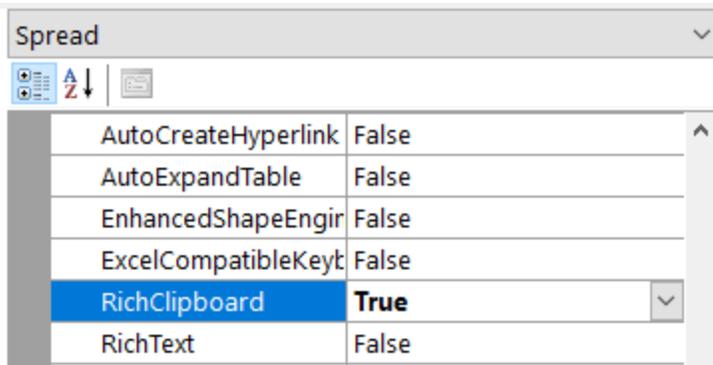
VB

```
'Enable the flatten style system only
Dim fpSpread1 = New FpSpread(LegacyBehavior.All And Not LegacyBehaviors.Style)
'Now, import the excel file
fpSpread1.OpenExcel("MyExcelFile.xlsx")
```

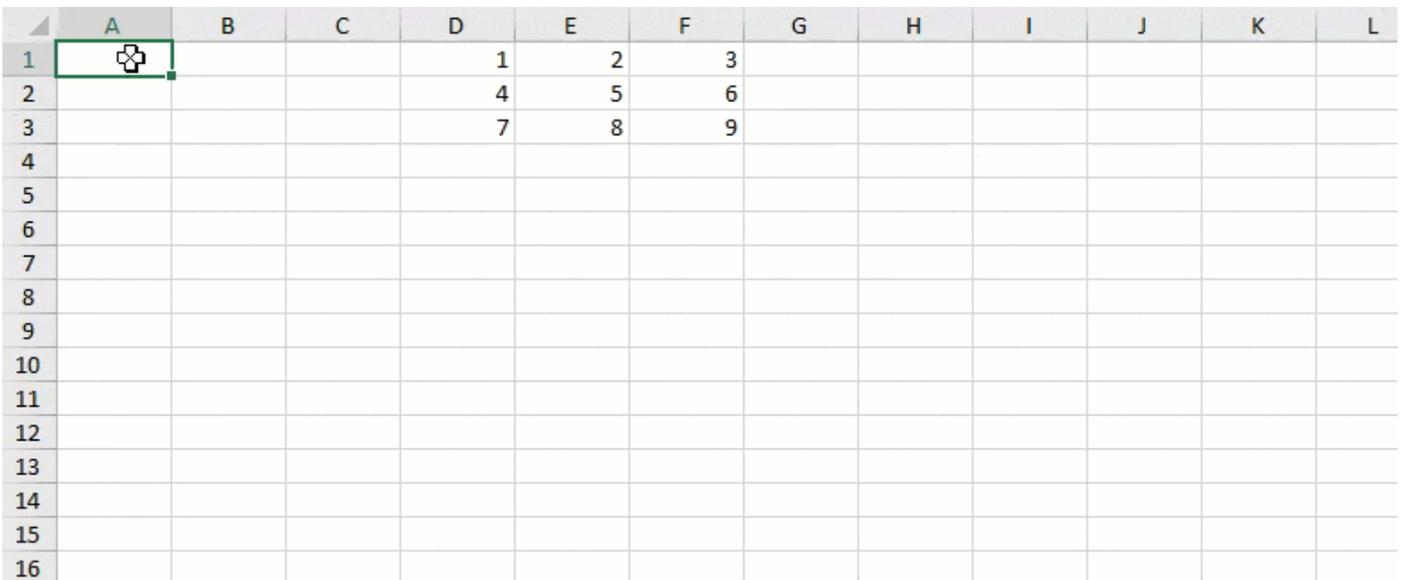
- Pattern fill and Gradient fill effects disappear when the cell is in edit mode. These effects are visible only in the view mode.

Inserting Cells

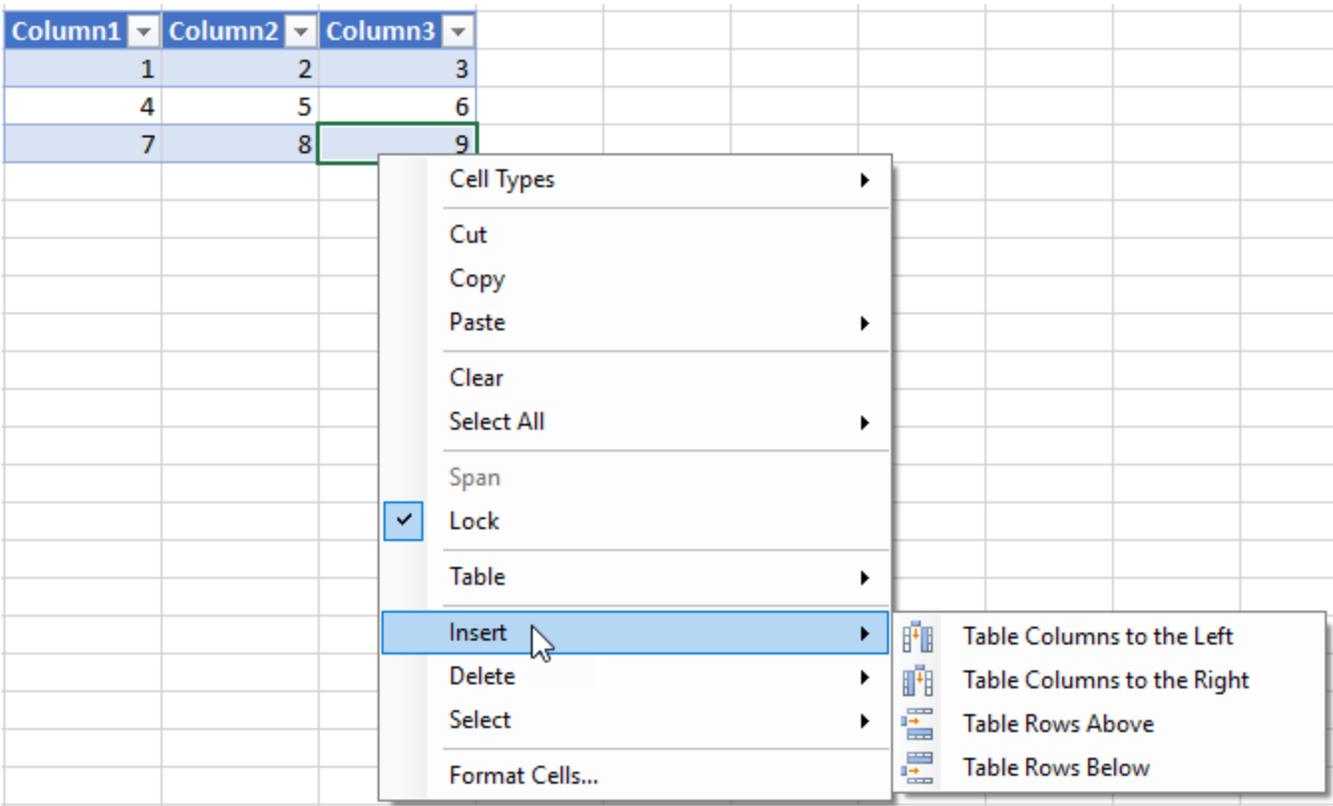
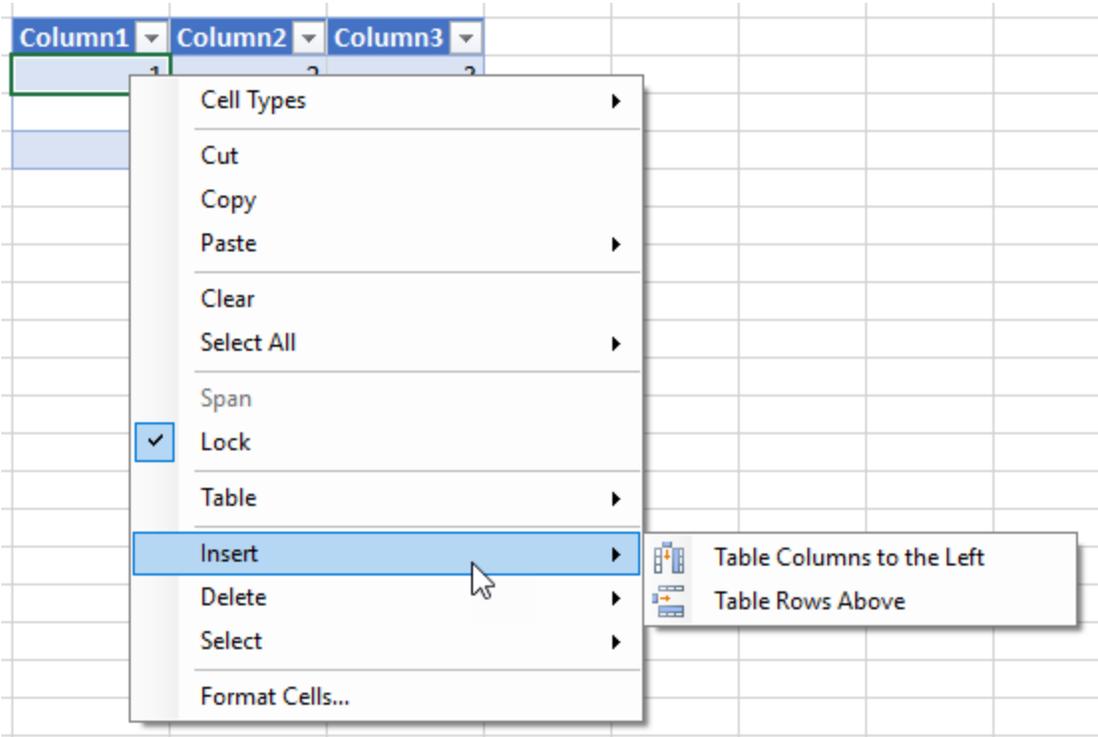
Spread Winforms allows you to insert cells, cut or copied cell ranges in a spreadsheet using the context menu. The **RichClipboard** option must be set to True to perform these operations. It can be accessed from the Property Panel of Spread Designer by selecting 'Spread' from the dropdown and navigating to Behavior > Features > RichClipboard as shown below:



The following GIF shows the behavior of 'Insert' option performed on a cell and a row header. As can be observed, the 'Insert' dialog appears and provides different options to choose whether the existing cells should be shifted to right or bottom or the entire row or column should be moved.

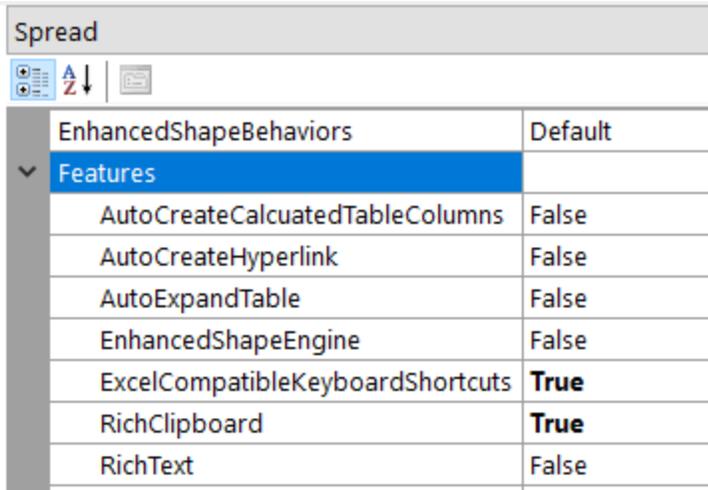


When the active cell is inside a table, the 'Insert' option provides different sub-options depending on the position of the active cell as shown below:



Insert Cut or Copied Cells

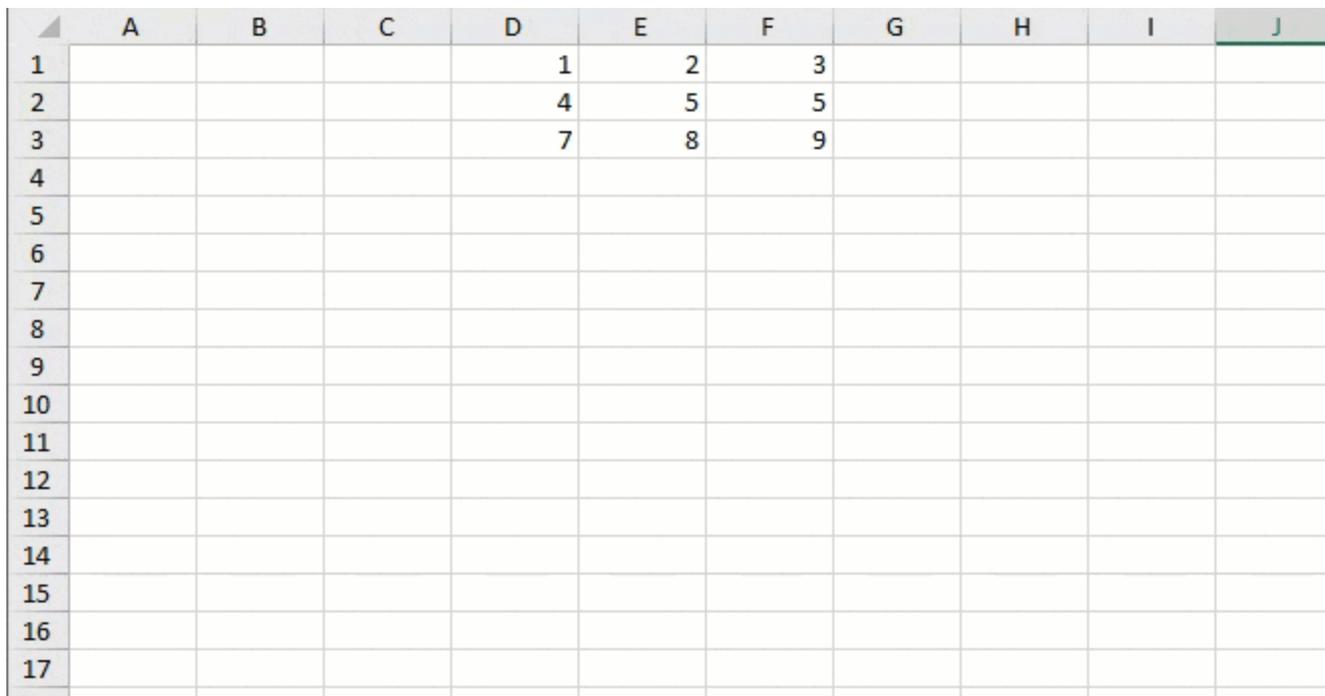
The 'Insert Cut/Copied Cells' option is displayed in the context menu when a cut or copy action is performed on a cell, column, or row range and a target cell, column, or row is right-clicked. The **ExcelCompatibleKeyboardShortcuts** property needs to be set to True to use the 'Insert Cut/Copied Cells' option as shown below:



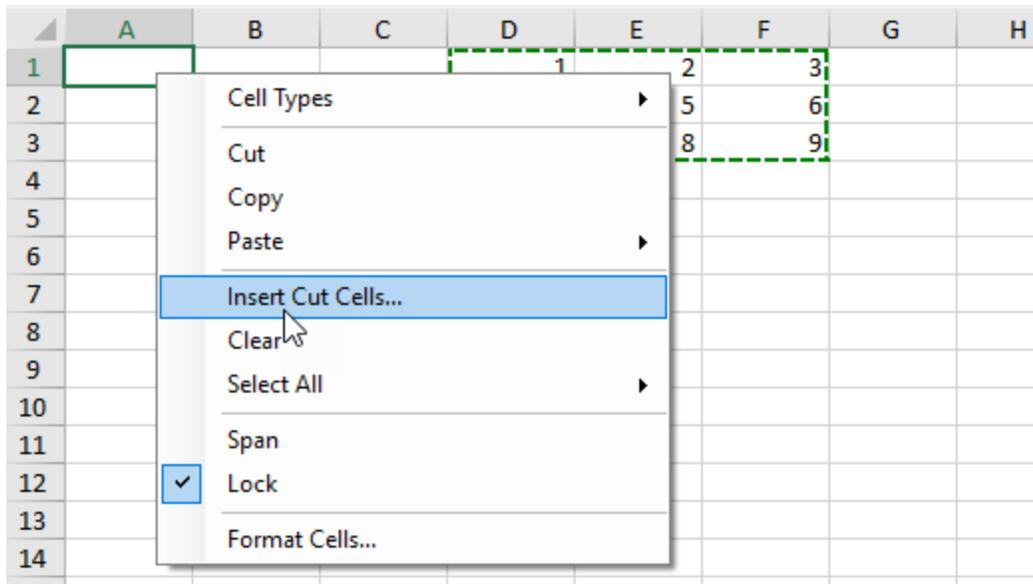
While inserting cut or copied cells, the 'Insert Paste' dialog provides following options:

- **Shift Cells Right:** When a cut or copied cell(s) is inserted, any data item(s) to the right shifts further towards right (the number of columns shifted is equal to the number of inserted cells).
- **Shift Cells Down:** When a cut or copied cell(s) is inserted, any data item(s) below it shifts downwards (the number of rows shifted is equal to the number of inserted cells).

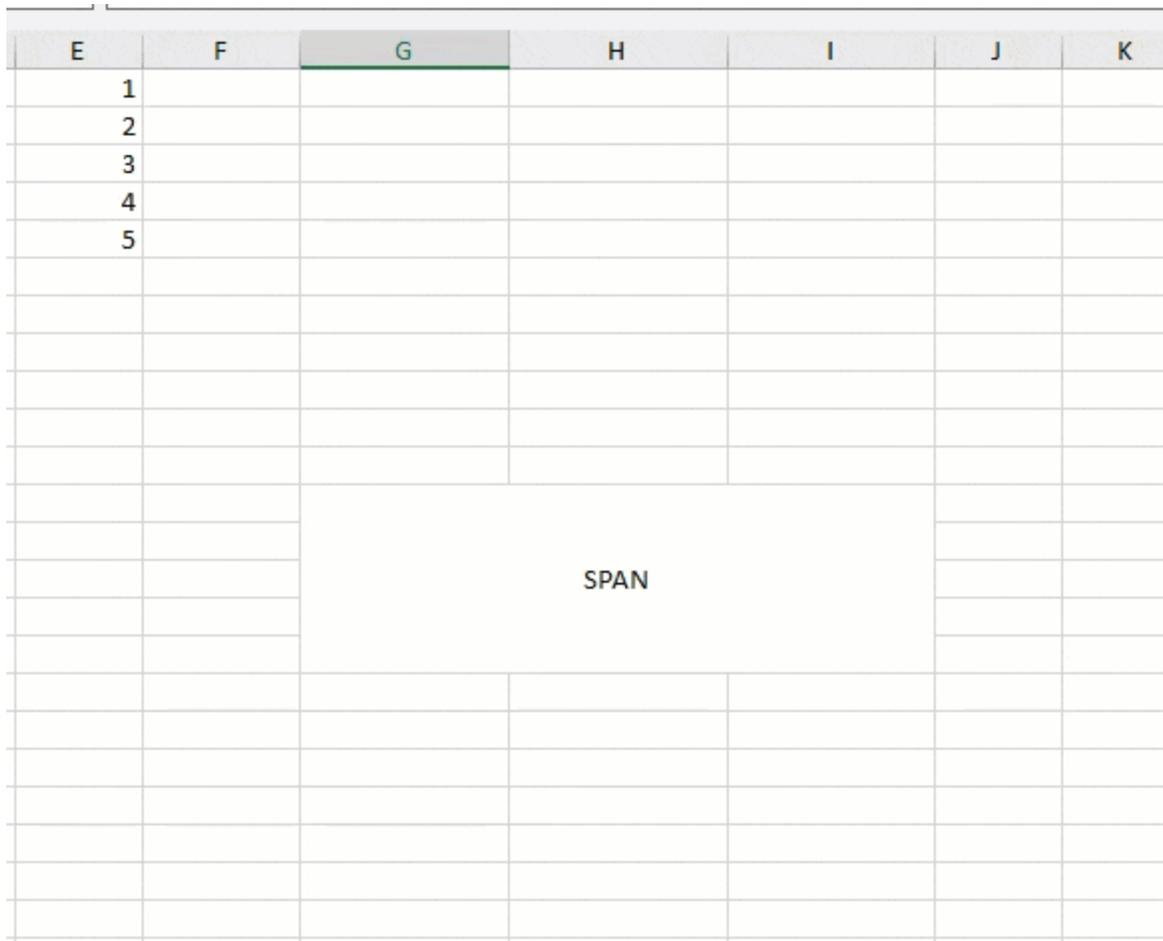
The below GIF shows the behavior of 'Insert Copied Cells' option when the existing data is shifted towards right:



Similarly, the below screenshot shows the 'Insert Cut Cells' option:



The below GIF shows the behavior of inserting copied cells when a spanned area exists adjacent to the target area:



The below GIF shows the behavior of inserting copied cells when a table exists adjacent to the target area:

The screenshot shows an Excel spreadsheet with columns A through I and rows 1 through 22. The data is as follows:

	A	B	C	D	E	F	G	H	I
1	1	2	3						
2	4	5	6						
3	7	8	9						
4									
5									
6									
7									
8									
9					Column1	Column2	Column3		
10					1	2	3		
11					4	5	6		
12					7	8	9		
13									
14									
15									
16									
17									
18									
19									
20									
21									
22									

The filter is applied to columns E, F, and G, with the header row (row 9) and the first three data rows (rows 10-12) visible.

The below GIF shows the behavior of inserting copied cells when a filter exists in the target area:

	A	B	C	D	E	F	G	H	I	J	K
1	1										
2	2										
3	3										
4	4										
5	5										
6	6										
7	7										
8					Name ▾						
9					John						
10					Jack						
11					Ben						
12					Tom						
13					Richard						
14		✚			Mark						
15											
16											
17											
18											
19											
20											
21											
22											

Insert Cut or Copied Cells in Headers

The following GIF shows the behavior of 'Insert Copied Cells' option while inserting copied data in a row header. Please note that the data of copied cells is not replicated throughout the row cells.

	A	B	C	D	E	F	G	H	I	J
1	1	2	3							
2	4	5	6							
3	7	8	9							
4										
5										
6										
7										
8										
9						Column1	Column2	Column3		
10						2	3	1		
11						5	6	4		
12						8	9	7		
13										
14										
15										
16						SPAN				
17										
18										
19										
20										
21										
22										
23										

The following GIF shows the behavior of 'Insert Cut Cells' option while inserting cut data in a row header.

	A	B	C	D	E	F	G	H	I
1	1	2	3						
2	4	5	6						
3	7	8	9						
4									
5									
6									
7									
8									
9									
10						Column1	Column2	Column3	
11						2	3	1	
12						5	6	4	
13						8	9	7	
14									
15									
16									
17									
18									
19									
20									
21									
22									
23									
24									
25									
26									
27									
28									

The similar behavior can be observed while using these options in column headers.

Users can also use the keyboard shortcut (CTRL+SHIFT+"+") to use Insert options from the context menu during runtime. When a cell or a cell range is selected and keyboard shortcut (CTRL+SHIFT+"+") is used, the following options are shown in the 'Insert' dialog in addition to the 'Shift cells right' and 'Shift cells down' options as explained above:

- **Entire Row:** Inserts row(s) equal to the number of rows in the selected range
- **Entire Column:** Inserts column(s) equal to the number of columns in the selected range

	A	B	C	D	E	F	G	H	I	J	K
1				1	2	3					
2				4	5	6					
3				7	8	9					
4											
5											
6											
7											
8											
9											
10											
11											
12											
13											
14											
15											
16											
17											
18											
19											

Also, on using the keyboard shortcut (CTRL+SHIFT+"+") and selecting a range inside a Table, the table rows or columns are inserted based on the selection:

	A	B	C	D	E	F	G	H	I
1	Column1 ▾	Column2 ▾	Column3 ▾						
2	1	2	3						
3	4	5	6						
4	7	8	9						
5									
6									
7									
8									
9									
10									
11									
12									
13									
14									
15									
16									
17									

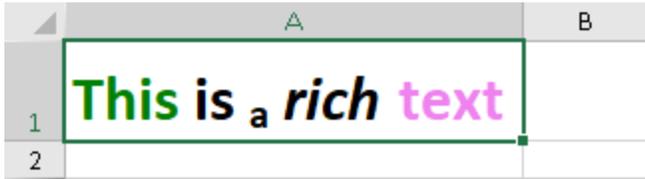
Limitations

- Unlike Excel, Spread does not support replicating data in a complete row or column while inserting copied cells.
- Filter range does not extend when cells are shifted downwards.

Setting Rich Text in a Cell

Spread for Winforms provides the ability to add rich text in a cell. You can add different rich text formatting options such as fonts, text styles, colors, superscripts, and subscripts.

The **RichText Class (on-line documentation)** can be used to set a rich text instance consisting of the string that will have rich text capabilities.



The following code shows how to add rich text in a cell.

C#

```
// Initiating a richtext object
GrapeCity.Spreadsheet.RichText richText = new GrapeCity.Spreadsheet.RichText("This is a
rich text");

// Defining font styles
GrapeCity.Spreadsheet.Font font = GrapeCity.Spreadsheet.Font.Empty;
GrapeCity.Spreadsheet.Font font2 = GrapeCity.Spreadsheet.Font.Empty;
GrapeCity.Spreadsheet.Font font3 = GrapeCity.Spreadsheet.Font.Empty;
GrapeCity.Spreadsheet.Font font4 = GrapeCity.Spreadsheet.Font.Empty;
GrapeCity.Spreadsheet.Font fontAll = GrapeCity.Spreadsheet.Font.Empty;

// Setting font styles
font.VerticalAlign = GrapeCity.Spreadsheet.VerticalTextAlignment.Subscript;
font2.Color =
GrapeCity.Spreadsheet.Color.FromKnownColor(GrapeCity.Core.KnownColor.Violet);
font3.Color =
GrapeCity.Spreadsheet.Color.FromKnownColor(GrapeCity.Core.KnownColor.Green);
font4.Italic = true;
fontAll.Bold = true;
fontAll.Size = 24;

// Applying styles to richtext instance
richText.Format(8, 1, font);
richText.Format(15, 4, font2);
richText.Format(0, 4, font3);
richText.Format(10, 4, font4);
richText.Format(fontAll);

// Setting test activesheet
GrapeCity.Spreadsheet.IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;

// Setting rich text in a cell
TestActiveSheet.Cells["A1"].Value = richText;
```

Visual Basic

```
'Initiating a richtext object
Dim richText As GrapeCity.Spreadsheet.RichText = New
GrapeCity.Spreadsheet.RichText("This is a rich text")
```

```

'Defining font styles
Dim font As GrapeCity.Spreadsheet.Font = GrapeCity.Spreadsheet.Font.Empty
Dim font2 As GrapeCity.Spreadsheet.Font = GrapeCity.Spreadsheet.Font.Empty
Dim font3 As GrapeCity.Spreadsheet.Font = GrapeCity.Spreadsheet.Font.Empty
Dim font4 As GrapeCity.Spreadsheet.Font = GrapeCity.Spreadsheet.Font.Empty
Dim fontAll As GrapeCity.Spreadsheet.Font = GrapeCity.Spreadsheet.Font.Empty

'Setting font styles
font.VerticalAlign = GrapeCity.Spreadsheet.VerticalTextAlignment.Subscript
font2.Color =
GrapeCity.Spreadsheet.Color.FromKnownColor(GrapeCity.Core.KnownColor.Violet)
font3.Color =
GrapeCity.Spreadsheet.Color.FromKnownColor(GrapeCity.Core.KnownColor.Green)
font4.Italic = True
fontAll.Bold = True
fontAll.Size = 24

'Applying styles to richtext instance
richText.Format(8, 1, font)
richText.Format(15, 4, font2)
richText.Format(0, 4, font3)
richText.Format(10, 4, font4)
richText.Format(fontAll)

'Setting active testsheet
Dim TestActiveSheet As GrapeCity.Spreadsheet.IWorksheet =
FpSpread1.AsWorkbook().ActiveSheet

'Setting rich text in a cell
TestActiveSheet.Cells("A1").Value = richText

```

 **Note:** If you edit the rich text in runtime then it behaves as plain text.

ExcelIO for Rich Text

Spreadsheets containing rich text formatting can easily be imported from or exported to Excel by setting **FpSpread.Features.RichText** ('RichText Property' in the on-line documentation) property to True.

The following code shows how to import or export an Excel file containing rich text.

C#

```

// Enable RichText property
fpSpread1.Features.RichText = true;

// Import file containing rich text
fpSpread1.ActiveSheet.OpenExcel("richtext-file.xlsx", 0);

// Export rich text set in cells to a file
fpSpread1.SaveExcel("excelfile.xlsx", FarPoint.Excel.ExcelSaveFlags.UseOOXMLFormat);

```

Visual Basic

```

'Enable RichText property
FpSpread1.Features.RichText = True

```

```
'Import file containing rich text
FpSpread1.ActiveSheet.OpenExcel("richtext-file.xlsx", 0)

'Export rich text set in cells to a file
FpSpread1.SaveExcel("excelfile.xlsx", FarPoint.Excel.ExcelSaveFlags.UseOOXMLFormat)
```

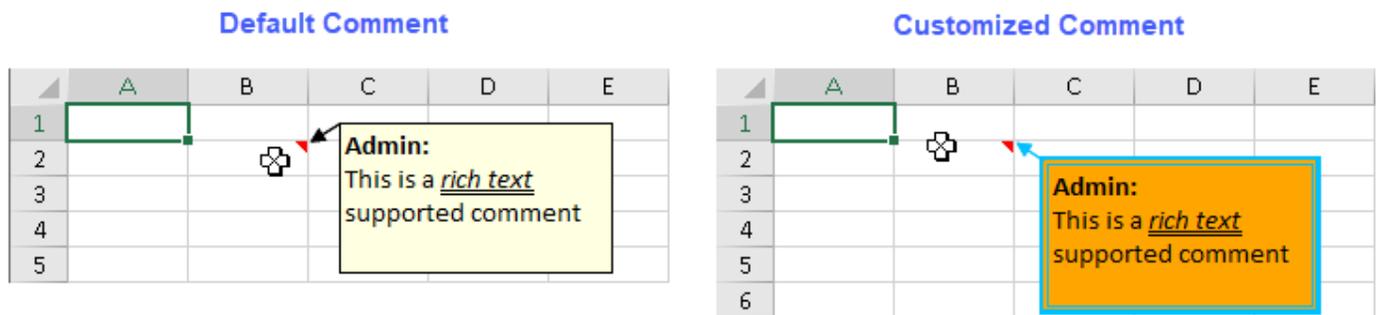
 **Note:** PDF export is not supported.

Adding a Comment to a Cell

Spread for WinForms supports adding comments to cells in a worksheet. You can add a plain text comment or note by referring to the **Adding a Note to a Cell** topic.

Additionally, you can enable the **EnhancedShapeEngine** property to add a string or RichText instances by using the **IRange.AddComment (AddComment Method' in the on-line documentation)** method.

You can modify the added comment by using the **IRange.Comment ('Comment Property' in the on-line documentation)** property. For example, the image below shows a default comment as well as a customized comment. The customized comment is set to always be visible and has shape properties such as background color, shape border style, and border color.



The following code shows how to set comments.

C#

```
// Get activesheet
GrapeCity.Spreadsheet.IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;

// Enable enhanced shape engine
fpSpread1.Features.EnhancedShapeEngine = true;

string username = "Admin" + ":";

// Initilize richtext object
GrapeCity.Spreadsheet.RichText richText = new GrapeCity.Spreadsheet.RichText(username +
"\r\nThis is a rich text \r\nsupported comment");
// Setting style
GrapeCity.Spreadsheet.Font font = GrapeCity.Spreadsheet.Font.Empty;
font.Bold = true;
GrapeCity.Spreadsheet.Font font2 = GrapeCity.Spreadsheet.Font.Empty;
font2.Italic = true;
font2.Underline = GrapeCity.Spreadsheet.UnderlineStyle.Double;
// Adding style to rich text
richText.Format(0, username.Length, font);
richText.Format(17, 9, font2);
```

```
// Adding a comment to cell
GrapeCity.Spreadsheet.IComment comment =
TestActiveSheet.Cells["B2"].AddComment(richText);

// Customizing comment style
TestActiveSheet.Cells["B2"].Comment.Visible = true; // Always show comment
TestActiveSheet.Cells["B2"].Comment.Shape.Fill.BackColor.ARGB = Color.Orange.ToArgb();
// Change comment background color
TestActiveSheet.Cells["B2"].Comment.Shape.Line.Style =
GrapeCity.Drawing.LineStyle.ThickThin; // Change border style
TestActiveSheet.Cells["B2"].Comment.Shape.Line.Weight = 5; // Change border thickness
TestActiveSheet.Cells["B2"].Comment.Shape.Line.ForeColor.ARGB =
Color.DeepSkyBlue.ToArgb(); // Change border color
```

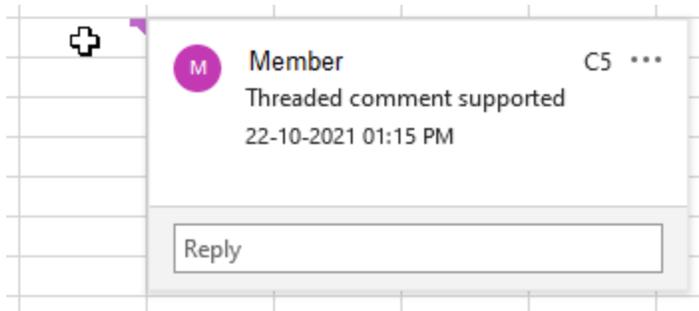
VB

```
'Get activesheet
Dim TestActiveSheet As GrapeCity.Spreadsheet.IWorksheet =
FpSpread1.AsWorkbook().ActiveSheet

'Enable enhanced shape engine
FpSpread1.Features.EnhancedShapeEngine = True
Dim username As String = "Admin" & ":"
'Inititalize richtext object
Dim richText As GrapeCity.Spreadsheet.RichText = New
GrapeCity.Spreadsheet.RichText(username & vbCrLf & "This is a rich text " & vbCrLf &
"supported comment")
'Setting style
Dim font As GrapeCity.Spreadsheet.Font = GrapeCity.Spreadsheet.Font.Empty
font.Bold = True
Dim font2 As GrapeCity.Spreadsheet.Font = GrapeCity.Spreadsheet.Font.Empty
font2.Italic = True
font2.Underline = GrapeCity.Spreadsheet.UnderlineStyle.Double
'Adding style to rich text
richText.Format(0, username.Length, font)
richText.Format(17, 9, font2)
'Adding a comment to cell
Dim comment As GrapeCity.Spreadsheet.IComment =
TestActiveSheet.Cells("B2").AddComment(richText)
'Customizing comment style
TestActiveSheet.Cells("B2").Comment.Visible = True 'Always show comment
TestActiveSheet.Cells("B2").Comment.Shape.Fill.BackColor.ARGB = Color.Orange.ToArgb()
'Change comment background color
TestActiveSheet.Cells("B2").Comment.Shape.Line.Style =
GrapeCity.Drawing.LineStyle.ThickThin 'Change border style
TestActiveSheet.Cells("B2").Comment.Shape.Line.Weight = 5 'Change border thickness
TestActiveSheet.Cells("B2").Comment.Shape.Line.ForeColor.ARGB =
Color.DeepSkyBlue.ToArgb() 'Change border color
```

Threaded Comments

Users can view threaded comments in the Spread UI. This helps in connecting several comments together and presenting a virtual conversation in the workbook.



Using Code

You can add threaded comments by using the [IRange.AddCommentThreaded](#) method.

C#

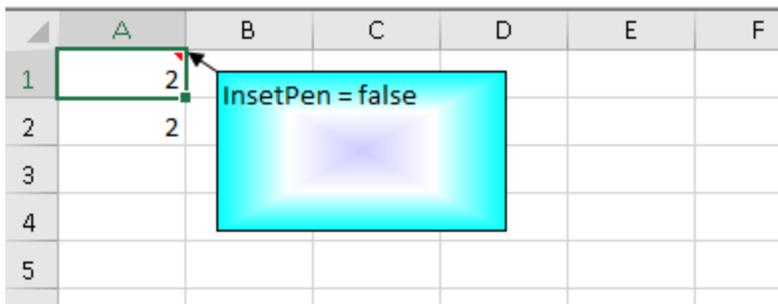
```
// Threaded comments
fpSpread1.LegacyBehaviors = LegacyBehaviors.None;
fpSpread1.Features.EnhancedShapeEngine = true;
fpSpread1.AsWorkbook().TestActiveSheet.Cells["C5"].AddCommentThreaded("Threaded comment supported").AddReply("First reply");
```

VB

```
' Threaded comments
fpSpread1.LegacyBehaviors = LegacyBehaviors.None
fpSpread1.Features.EnhancedShapeEngine = True
fpSpread1.AsWorkbook().TestActiveSheet.Cells("C5").AddCommentThreaded("Threaded comment supported").AddReply("First reply")
```

Gradient Fill Effects

You can add gradient effects to emphasize the comments by applying fill-color and patterns.



Using Code

To add the gradient effects, use the **Fill** property of the **IShapeBase** interface, which gets an object of the **IFillFormat** interface for a specified shape that contains the fill formatting properties for shape.

C#

```
fpSpread1.Features.EnhancedShapeEngine = true;
IShape shape;
shape = TestActiveSheet.Cells["A1"].AddComment("InsetPen = false").Shape;
TestActiveSheet.Cells["A1"].Comment.Visible = true;
shape.Fill.OneColorGradient(GrapeCity.Spreadsheet.Drawing.GradientStyle.FromCenter, 1, 1);
```

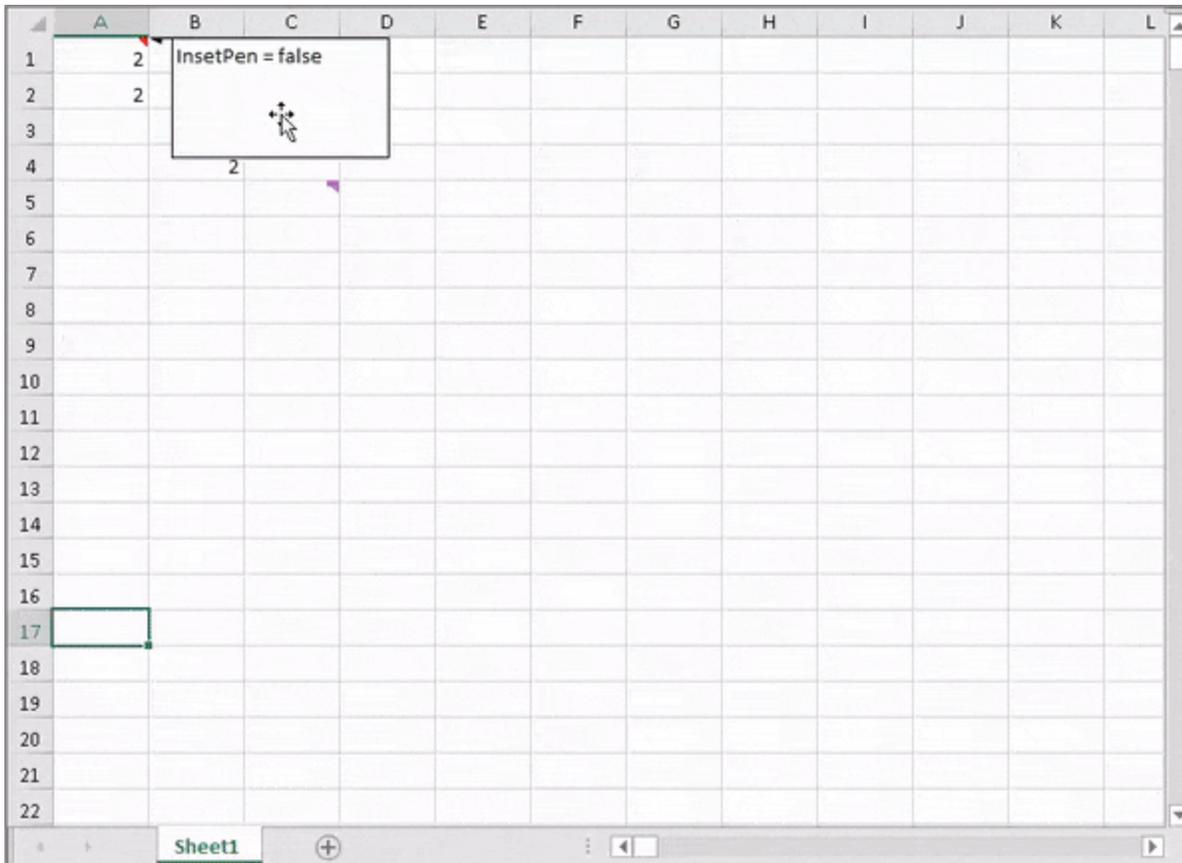
```
shape.Fill.GradientStops.Delete(0);
shape.Fill.GradientStops.Delete(0);
shape.Fill.GradientStops.Delete(0);
shape.Fill.GradientStops.Insert(0xffff00, 0);
shape.Fill.GradientStops.Insert(0xffffffff, 0.5);
shape.Fill.GradientStops.Insert(0x00ffff, 1);
shape.Fill.GradientStops[0].Color.SchemeColor = 23;
shape.Fill.PathShadeType = GrapeCity.Drawing.PathShadeType.Shape;
IShape aa = fpSpread1.AsWorkbook().ActiveSheet.Cells["A1"].Comment.Shape;
```

VB

```
fpSpread1.Features.EnhancedShapeEngine = True
Dim shape As IShape
shape = TestActiveSheet.Cells("A1").AddComment("InsetPen = false").Shape
TestActiveSheet.Cells("A1").Comment.Visible = True
shape.Fill.OneColorGradient(GrapeCity.Spreadsheet.Drawing.GradientStyle.FromCenter, 1,
1)
shape.Fill.GradientStops.Delete(0)
shape.Fill.GradientStops.Delete(0)
shape.Fill.GradientStops.Delete(0)
shape.Fill.GradientStops.Insert(0xffff00, 0)
shape.Fill.GradientStops.Insert(0xffffffff, 0.5)
shape.Fill.GradientStops.Insert(0x00ffff, 1)
shape.Fill.GradientStops(0).Color.SchemeColor = 23
shape.Fill.PathShadeType = GrapeCity.Drawing.PathShadeType.Shape
Dim aa As IShape = fpSpread1.AsWorkbook().ActiveSheet.Cells("A1").Comment.Shape
```

Using Runtime UI

You can add or manipulate gradient fill effects at run time via the **Fill Effects** dialog. The Fill Effects dialog provides options like colors, variants, shading styles, patterns, and a sample preview to view the gradient effect.



To invoke the dialog at runtime, select the comment on the sheet and use the keyboard shortcut keys combination **Ctrl+1**, or select the **Format Comment** option from the built-in context menu.

In the **Format Comments** dialog, navigate to the **Colors and Lines** tab and select the **Fill Effects** option from the Color picker dropdown to view the **Fill Effects** dialog.

 For the keyboard shortcut to work, set the **ExcelCompatibleKeyboardShortcuts** property to true.

You can also invoke the Fill Effects dialog at runtime using the **FillEffects** method of the **BuiltInDialogs** class. To learn more about this class, see **Working with BuiltIn Dialogs**.

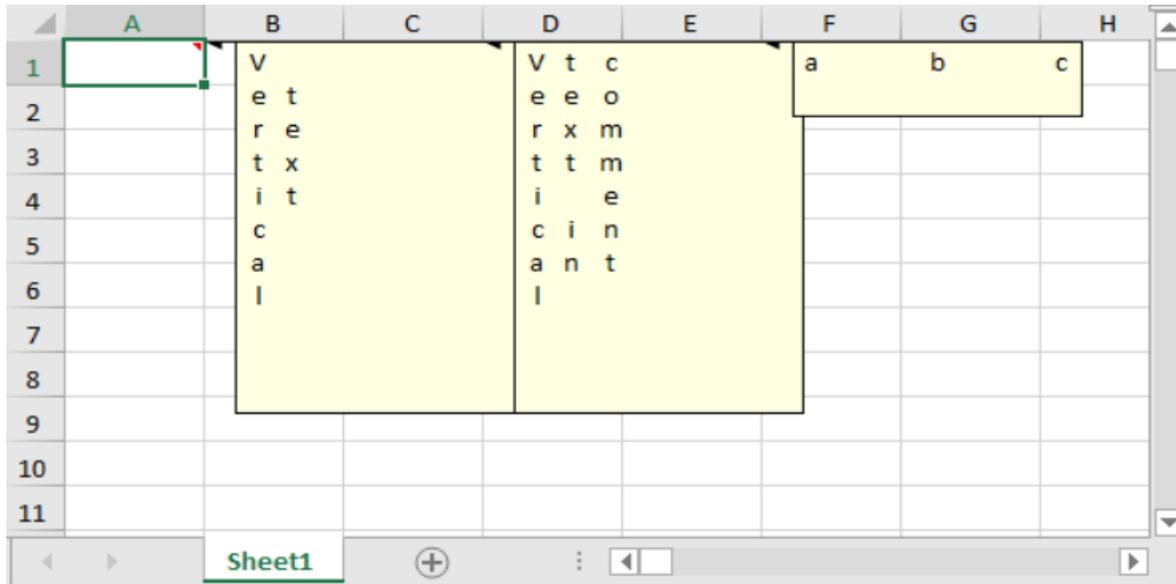
Using Spread Designer

To view comments and add gradient effects to the block in the Spread Designer, follow the below mentioned steps:

1. Set the **EnhancedShapeEngine** and **ExcelCompatibleKeyboardShortcuts** to true in the properties window.
2. Select the comment on which you want to add or modify the gradient effect.
3. Invoke the **Format Comment** dialog.
4. Click the **Fill Effects** button from the **Colors and Lines** tab to open the **Fill Effects** dialog.

Text Orientation

Formatting text in the comments is helpful for customizing the appearance and readability of comments or notes in the worksheet, allowing you to visually display your annotations in a more appealing way. Usually, Spread allows you to orient the text horizontally or vertically. However, using the **HorizontalRotatedFarEast** property of **TextOrientation** (**'TextOrientation Enumeration'** in the on-line documentation) enumeration, you can control the orientation and formatting of text within cell comments. This property allows you to stack text vertically.



 The implementation of this feature takes effect only if the **EnhancedShapeEngine** property is set to true.

Using code

The following example code shows how to change text orientation in cell comments.

C#

```
// Add support paint stacked text direction for cell comment
fpSpread1.Features.EnhancedShapeEngine = true;
var activeSheet = fpSpread1.AsWorkbook().ActiveSheet;
var comment = activeSheet.Cells["A1"].AddComment("Vertical\n text");
comment.Shape.TextFrame.Orientation =
GrapeCity.Spreadsheet.Drawing.TextOrientation.HorizontalRotatedFarEast;
comment.Shape.TextFrame.WordWrap = true;
comment.Shape.Height = 200;
comment.Visible = true;
var comment2 = activeSheet.Cells["C1"].AddComment("Vertical text in comment");
comment2.Shape.TextFrame.Orientation =
GrapeCity.Spreadsheet.Drawing.TextOrientation.HorizontalRotatedFarEast;
comment2.Shape.TextFrame.WordWrap = true;
comment2.Shape.Height = 200;
comment2.Visible = true;
var comment3 = activeSheet.Cells["E1"].AddComment("abc");
comment3.Shape.TextFrame.Orientation =
GrapeCity.Spreadsheet.Drawing.TextOrientation.HorizontalRotatedFarEast;
comment3.Shape.TextFrame.HorizontalAlignment =
GrapeCity.Spreadsheet.HorizontalAlignment.Justify;
comment3.Shape.TextFrame.VerticalAlignment =
GrapeCity.Spreadsheet.VerticalAlignment.Justify;
comment3.Shape.Height = 40;
comment3.Visible = true;
```

VB

```
' Add support paint stacked text direction for cell comment
fpSpread1.Features.EnhancedShapeEngine = True
Dim activeSheet = fpSpread1.AsWorkbook().ActiveSheet
```

```
Dim comment = activeSheet.Cells("A1").AddComment("Vertical" & vbLf & " text")
comment.Shape.TextFrame.Orientation =
GrapeCity.Spreadsheet.Drawing.TextOrientation.HorizontalRotatedFarEast
comment.Shape.TextFrame.WordWrap = True
comment.Shape.Height = 200
comment.Visible = True
Dim comment2 = activeSheet.Cells("C1").AddComment("Vertical text in comment")
comment2.Shape.TextFrame.Orientation =
GrapeCity.Spreadsheet.Drawing.TextOrientation.HorizontalRotatedFarEast
comment2.Shape.TextFrame.WordWrap = True
comment2.Shape.Height = 200
comment2.Visible = True
Dim comment3 = activeSheet.Cells("E1").AddComment("abc")
comment3.Shape.TextFrame.Orientation =
GrapeCity.Spreadsheet.Drawing.TextOrientation.HorizontalRotatedFarEast
comment3.Shape.TextFrame.HorizontalAlignment =
GrapeCity.Spreadsheet.HorizontalAlignment.Justify
comment3.Shape.TextFrame.VerticalAlignment =
GrapeCity.Spreadsheet.VerticalAlignment.Justify
comment3.Shape.Height = 40
comment3.Visible = True
```

At Runtime

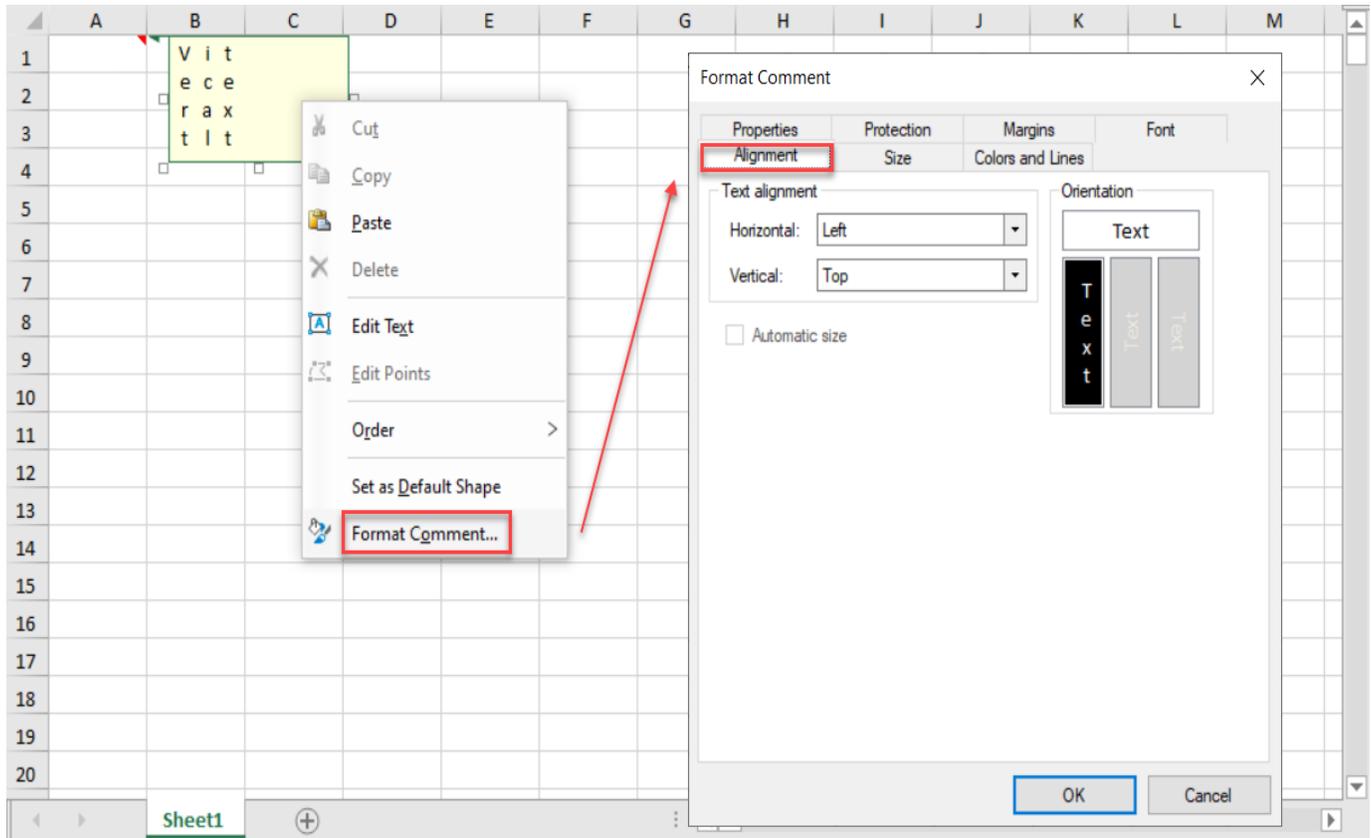
Using the **Format Comment** dialog, you can change the orientation of the comments you have added to cells. To invoke the dialog at runtime, perform the following steps:

1. Run the code below to load a spread control with a comment.

C#

```
fpSpread1.Features.EnhancedShapeEngine = true;
var activeSheet = fpSpread1.AsWorkbook().ActiveSheet;
var comment = activeSheet.Cells["A1"].AddComment("Vertical text");
comment.Shape.TextFrame.Orientation =
GrapeCity.Spreadsheet.Drawing.TextOrientation.HorizontalRotatedFarEast;
comment.Visible = true;
```

2. Right-click on the comment to open up the **Format Comment...** context menu.
3. Select the **Alignment** tab.
4. Select the **Orientation** as per your choice and click **OK**.



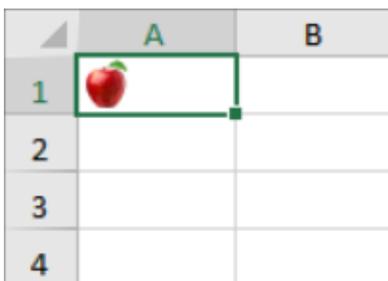
Adding Image in a Cell

Spread for Winforms provides the ability to insert images in a cell as a value. This helps to add images in a cell without converting the cell into a **ImageCellType** class object.

 To add images using ImageCellType, refer to **Setting an Image Cell** topic.

You can use one of the following methods to display images in a cell. These methods help to set the image object to the cell value, set a local image file path, or set a base64-encoded image string in cells:

- Using value - Convert the value itself to an image data type such as image, byte or Stream that stores image's binary data. Use the [System.Drawing.Image](#) class methods to take the image file path as a parameter. The following image shows a picture inserted inside a cell by providing an image file path.



C#

```
IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;
fpSpread1.AsWorkbook().WorkbookSet.CalculationEngine.CalcFeatures =
CalcFeatures.All;
```

```

fpSpread1.LegacyBehaviors = FarPoint.Win.Spread.LegacyBehaviors.None;

// Displaying cell image using value

// Image type
TestActiveSheet.Cells["A1"].Value = Image.FromFile(@"D:\apple.jpg");

// Byte type
// TestActiveSheet.Cells["A3"].Value = File.ReadAllBytes("D:\\apple.jpg");

// Stream type
// TestActiveSheet.Cells["A5"].Value = new FileStream("D:\\apple.jpg",
FileMode.Open);

```

Visual Basic

```

Dim TestActiveSheet As IWorksheet = FpSpread1.AsWorkbook().ActiveSheet
FpSpread1.AsWorkbook().WorkbookSet.CalculationEngine.CalcFeatures =
CalcFeatures.All
FpSpread1.LegacyBehaviors = FarPoint.Win.Spread.LegacyBehaviors.None

'Displaying cell image using value

'Image type
TestActiveSheet.Cells("A1").Value = Image.FromFile("D:\apple.jpg")

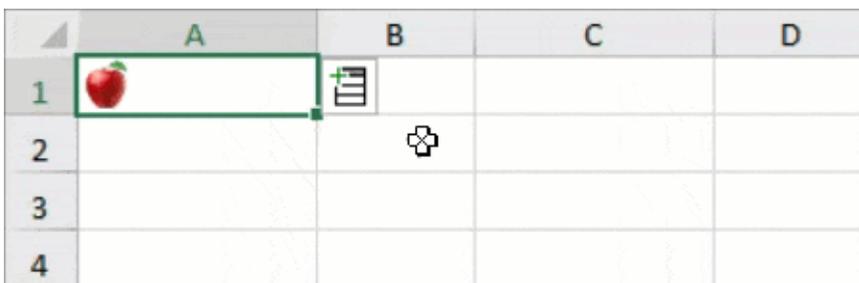
'Byte type
'TestActiveSheet.Cells("A3").Value = File.ReadAllBytes("D:\\apple.jpg")

'Stream type
'TestActiveSheet.Cells("A5").Value = New FileStream("D:\apple.jpg", FileMode.Open)

```

- Using attribute - The **CellValueDataTypeAttribute** ('CellValueDataTypeAttribute Class' in the on-line documentation) class provides different data types such as byte, image, and string. These data types can be used to define a class attribute to get images.

The following GIF illustrates the use of attributes to display an image in a cell as well as show additional information.



C#

```

private void CellImageAsAttribute_Load(object sender, EventArgs e)
{
    IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;

    fpSpread1.AsWorkbook().WorkbookSet.CalculationEngine.CalcFeatures =
CalcFeatures.All;
    fpSpread1.LegacyBehaviors = FarPoint.Win.Spread.LegacyBehaviors.None;

    // Displaying cell image using attribute
    GrapeCity.CalcEngine.RichValue<Country> ct = new

```

```

GrapeCity.CalcEngine.RichValue<Country>(new Country()
{
    Name = "India",
    Capital = "New Delhi",
});
ct.ShowDetailsIcon = true;

TestActiveSheet.Cells["A1"].Value = ct;

TestActiveSheet.Columns[0].ColumnWidth = 100;
TestActiveSheet.Columns[2].ColumnWidth = 100;
}

[System.Reflection.DefaultMember("Data")]
[CellImage("Name")]
public class Country
{
    public string Name { get; set; }
    [DisplayName("Capital Name")]
    public string Capital { get; set; }
    public string ContentType => "image/png";
}

[GrapeCity.CalcEngine.CellValueDataType(GrapeCity.CalcEngine.PrimitiveValueType.Image)]
public Image Data
{
    get
    {
        return Image.FromFile(@"D:\apple.jpg");
    }
}
}

```

Visual Basic

```

Private Sub CellImageAsAttribute_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
    Dim TestActiveSheet As IWorksheet = FpSpread1.AsWorkbook().ActiveSheet

    FpSpread1.AsWorkbook().WorkbookSet.CalculationEngine.CalcFeatures =
CalcFeatures.All
    FpSpread1.LegacyBehaviors = FarPoint.Win.Spread.LegacyBehaviors.None

    'Displaying cell image using attribute
    Dim ct As GrapeCity.CalcEngine.RichValue(Of Country) = New
GrapeCity.CalcEngine.RichValue(Of Country)(New Country() With {
        .Name = "India",
        .Capital = "New Delhi"
    })
    ct.ShowDetailsIcon = True

    TestActiveSheet.Cells("A1").Value = ct

    TestActiveSheet.Columns(0).ColumnWidth = 100
    TestActiveSheet.Columns(2).ColumnWidth = 100
End Sub

<System.Reflection.DefaultMember("Data")>
<CellImage("Name")>

```

```

Public Class Country
    Public Property Name As String
    <DisplayName("Capital Name")>
    Public Property Capital As String

    Public ReadOnly Property ContentType As String
        Get
            Return "image/png"
        End Get
    End Property

    <GrapeCity.CalcEngine.CellValueDataType(GrapeCity.CalcEngine.PrimitiveValueType.Image)>
    Public ReadOnly Property Data As Image
        Get
            Return Image.FromFile("D:\apple.jpg")
        End Get
    End Property
End Class

```

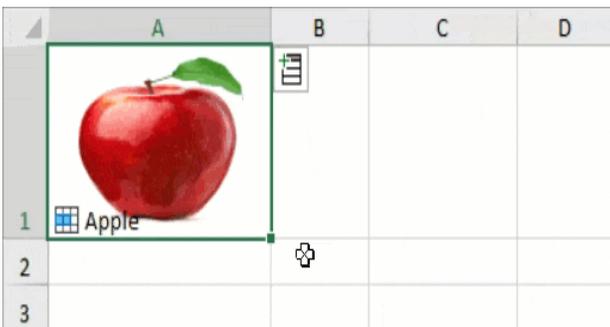
 **Note:** The data type return by string only supports local image file path.

Display Image and Cell Value

If you want to display the image along with a cell value, it can be achieved by using the **CellImageAttribute** ('CellImageAttribute Class' in the on-line documentation) class members. Its constructor method takes the following parameters:

Parameter	Description
<i>member</i>	A string value indicating the member specified cell inline image data.
<i>isField</i>	A Boolean value indicating whether the member is a field. Default is false.

The following GIF illustrates an image displayed along with a cell value according to the image attributes set in Spread.



C#

```

private void CellImageAndValue_Load(object sender, EventArgs e)
{
    IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;
    fpSpread1.AsWorkbook().WorkbookSet.CalculationEngine.CalcFeatures = CalcFeatures.All;
    fpSpread1.LegacyBehaviors = FarPoint.Win.Spread.LegacyBehaviors.None;

    // Displaying both image and cell value
    RichValue<Country> ct = new RichValue<Country>(new Country())
    {

```

```

        Name = "Apple",
    });
    ct.ShowDetailsIcon = true;
    TestActiveSheet.Cells["A1"].Value = ct;

    TestActiveSheet.Rows[0].RowHeight = 100;
    TestActiveSheet.Columns[0].ColumnWidth = 150;
    TestActiveSheet.Columns[2].ColumnWidth = 100;
}

[System.Reflection.DefaultMember("Name")]
[CellImage("Image")]
public class Country
{
    public string Name { get; set; }
    [CellValueDataType(PrimitiveValueType.Image)]
    public string Image
    {
        get
        {
            return @"D:\apple.jpg";
        }
    }
}

```

Visual Basic

```

Private Sub CellImageAndValue_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    Dim TestActiveSheet As IWorksheet = FpSpread1.AsWorkbook().ActiveSheet

    FpSpread1.AsWorkbook().WorkbookSet.CalculationEngine.CalcFeatures = CalcFeatures.All
    FpSpread1.LegacyBehaviors = FarPoint.Win.Spread.LegacyBehaviors.None

    'Displaying both image and cell value
    Dim ct As RichValue(Of Country) = New RichValue(Of Country)(New Country() With {
        .Name = "Apple"
    })
    ct.ShowDetailsIcon = True

    TestActiveSheet.Cells("A1").Value = ct

    TestActiveSheet.Rows(0).RowHeight = 100
    TestActiveSheet.Columns(0).ColumnWidth = 150
    TestActiveSheet.Columns(2).ColumnWidth = 100
End Sub

<System.Reflection.DefaultMember("Name")>
<CellImage("Image")>
Public Class Country
    Public Property Name As String

    <CellValueDataType(PrimitiveValueType.Image)>
    Public ReadOnly Property Image As String
        Get
            Return "D:\apple.jpg"
        End Get
    End Property
End Class

```

 **Note:** This class has a higher priority than `CellValueDataTypeAttribute` ('`CellValueDataTypeAttribute Class`' in the on-line documentation) class.

Spread for WinForms also provides the `GC.IMAGE` function to place an image in a cell. For more information about this function, refer to the [Image Sparkline](#) topic. You can also use the `IMAGE` function, which inserts images into cells from a source location.

Insert/Paste Picture in Cell

Spread for Winforms allows you to use the `InsertPictureInCell` or `PastePictureInCell` members of the `IRange` interface to insert or paste an in-place picture into a cell. Note that it works with flat style mode (no `LegacyBehaviors.Style`) only. You can insert a picture into a cell using ribbonBar and paste the same using the context menu.

Insert Picture in a Cell

Using Code

The following example code inserts a picture using the `IRange.InsertPictureInCell` API.

C#

```
fpSpread1.AsWorkbook().ActiveSheet.Cells[7, 2].InsertPictureInCell(@"picture.jpg");
```

At Runtime

Follow the steps below to insert a picture using the ribbonBar.

1. Run the code below to load a spread control with the ribbonBar.

C#

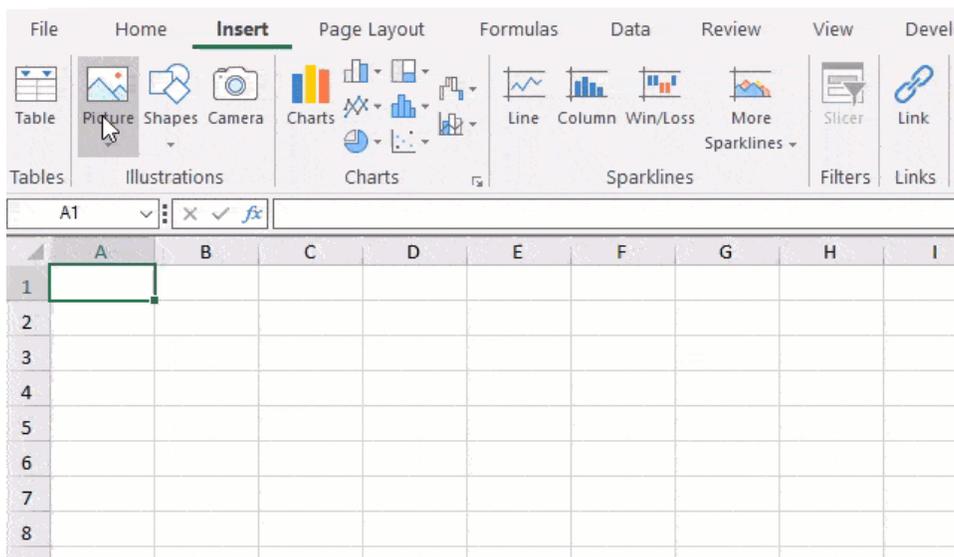
```
ribbonBar1.Attach(fpSpread1);
```

2. Choose the **Insert** option from the menu.
3. In the **Illustrations** group, click on **Picture** and select **Place in Cell**.

 You can add images to the worksheet and have them float on top by using the **Place over Cells** option.

4. Select a picture and click **OK**.

The following GIF illustrates how to insert a picture into a cell using ribbonBar.



Paste Picture in a cell

Using code

The following example code pastes a picture using the **IRange.PastePictureInCell** API.

1. Copy a picture to Clipboard.
2. Run the code below to load a spread control with the ribbonBar having **RichClipboard** property as True.

C#

```
fpSpread1.Features.RichClipboard = true;
fpSpread1.AsWorkbook().ActiveSheet.ActiveCell.PastePictureInCell();
```

At runtime

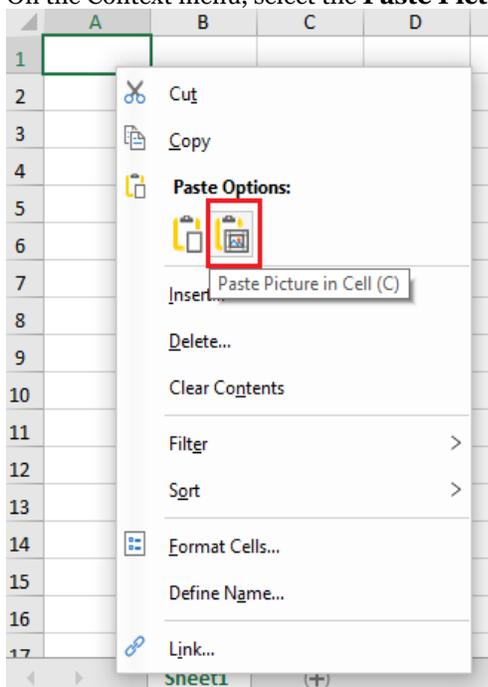
The following steps show how to paste a picture using ribbonBar.

1. Copy a picture to Clipboard.
2. Run the code below to load a spread control with the ribbonBar having **RichClipboard** property as True.

C#

```
fpSpread1.Features.RichClipboard = true;
ribbonBar1.Attach(fpSpread1);
```

3. Right-click on the cell where you want to paste the copied picture.
4. On the Context menu, select the **Paste Picture in Cell(C)** option.



Formatting a Cell Value

Spread for Winforms provides the ability to apply formatting to a cell value itself. This helps to retain the format string of the value throughout the calculation.

The **GrapeCity.CalcEngine.IFormattedCellValue** ('IFormattedCellValue Interface' in the on-line **documentation**) interface is used to set the value and format properties in a cell. It represents the value of a cell with a format string and allows the user to implement their own formatted cell value. The [CalcEngine.FormattedCellValue](#) class

members can also be used to create an object instance to implement the feature.

The following code example shows how to implement formatting to a cell value instead of to a cell.

C#

```
GrapeCity.Spreadsheet.IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;

GrapeCity.CalcEngine.FormattedCellValue fmval =
GrapeCity.CalcEngine.FormattedCellValue.Create(fpSpread1.AsWorkbook(), 123,
"dd/mm/yyyy");
TestActiveSheet.Cells["A1"].Value = fmval;
TestActiveSheet.Cells["A2"].Formula = "A1"; // A2 show value same as A1 : 02/05/1900
```

Visual Basic

```
Dim TestActiveSheet As GrapeCity.Spreadsheet.IWorksheet =
FpSpread1.AsWorkbook().ActiveSheet

Dim fmval As GrapeCity.CalcEngine.FormattedCellValue =
GrapeCity.CalcEngine.FormattedCellValue.Create(FpSpread1.AsWorkbook(), 123,
"dd/mm/yyyy")

TestActiveSheet.Cells("A1").Value = fmval
TestActiveSheet.Cells("A2").Formula = "A1" ' A2 show value same as A1 : 02/05/1900
```

The following behavior should be noted when working with formatted cell values:

- It is not supported in Spread Designer or through ExcelIO.
- The formula textbox shows the editing value the same as the cell.
- Any cell type including customized cell types works with **IFormattedCellValue** as a normal **IReadOnlyPrimitiveValue**.

Cell Types

Cell types define the type of information that appears in a cell, how that information is displayed, and how the user can interact with it. There are two different groups of cell types that can be set for cells in a sheet: ones that are simply related to formatting of text in a cell and ones that display a control or graphic. Spread includes built-in cell types and allows you define custom cell types. Cell types can be assigned to individual cells or entire rows or columns.

These tasks of working with cell types are organized into these broad categories:

- **Understanding Cell Type Basics**
- **Understanding How Cell Types Work**
- **Understanding How Cell Types Display and Format Data**
- **Understanding How Cell Type Affects Model Data**
- **Working with Editable Cell Types**
- **Working with Graphical Cell Types**
- **Understanding Additional Features of Cell Types**

Note that some graphical elements in certain cell types are affected by XP themes (visual styles). Setting the **VisualStyles ('VisualStyles Property' in the on-line documentation)** property of the Spread component to "off" can allow visual customizations of those graphical cell types to work as expected. For more information, refer to **Using XP Themes with the Component**.

For detailed information on classes behind the various built-in cell types, refer to the **FarPoint.Win.Spread.CellType ('FarPoint.Win.Spread.CellType Namespace' in the on-line documentation)** namespace.

For information on setting cell types using the Spread Designer, refer to the **Spread Designer Guide (on-line documentation)**.

Understanding Cell Type Basics

You can specify the way a user interacts with a cell, including how data is entered, displayed, and validated, by specifying the cell type. The cell type defines an editor control for the cell that handles data entry, a formatter control to handle how the data is interpreted, and a renderer control that handles how the data is displayed in the cell. Examples of cell types are check box cell, date-time cell, or a simple text cell.

Cell types can be set for individual cells, columns, rows, a range of cells, or an entire sheet. For any cell type there are properties of a cell that can be set. In general, working with cell types includes defining the cell type, setting the properties, and applying that cell type to cells.

Pay attention to data types when working with cell types. For several cell types, including **ComboBoxCellType ('ComboBoxCellType Class' in the on-line documentation)**, **DateTimeCellType ('DateTimeCellType Class' in the on-line documentation)**, and **MultipleOptionCellType ('MultiOptionCellType Class' in the on-line documentation)**, there is an **EditorValue** property. Make sure the **Value** property you are setting in the cell matches the type specified by the **EditorValue** property.

Editor, Formatter, and Renderer

A cell type consists of an editor, a renderer, and a formatter. The editor is a control instance that is created and placed in the location of the cell when the cell goes into edit mode. The editor is responsible for creating and managing the cell's edit control when in edit mode. The formatter is responsible for converting the cell's value to and from text (for example when getting or setting a cell's **Text ('Text Property' in the on-line documentation)** property). The renderer paints the control inside the cell rectangle when the editor is not there or when the cell is not in edit mode.

In most cases, you want the cell to look the same whether you are in edit mode or not in edit mode. In these cases, you would create a single cell type and assign it to the cell's **CellType ('CellType Property' in the on-line documentation)** property. This single cell type is used as the cell's editor, renderer, and formatter. If you want the cell

to appear differently depending on whether you are in edit mode or not in edit mode, then you can create two different cell types and assign one cell type as the cell's editor and the other cell type as the cell's renderer. In this case, you probably also want to assign one of the cell types as the cell's formatter. For more information, refer to the **ICellType** ('**ICellType Interface**' in the on-line documentation) interface.

EditBaseCellType

The design of cell editing requires that the cell type return an editor control that is then placed over the cell. The editor control can be text based (for example, text box) or graphics based (for example, check box). The editor control can drop down lists (for example, combo box) or pop up dialogs (for example, date picker). The **EditBaseCellType** ('**EditBaseCellType Class**' in the on-line documentation) class is a class from which the built-in text based cell types (for example, general, text, number, data-time, and so on) are derived. The class can also be used to derive custom cell types that are text based. The **ISubEditor** ('**ISubEditor Interface**' in the on-line documentation) interface is used to combine a text-based editor with a drop-down list (for example, combo box) or pop-up dialog (for example date picker). The data model can hold any value, including colors. The cell type is always passed the raw value from the data model.

Header Cells

While you can assign a cell type to the cells in the row header or column header, the cell type is only used for painting purposes; the component renders header cells but does not allow editing. In-cell editing is limited to cells in the data area. If you want to have something editable that acts like a header, you can hide (turn off) the column header, freeze the first row of the spreadsheet, then use the frozen row to appear as header cells.

Understanding How Cell Types Work

You can specify the cell type for an individual cell, a range of cells, columns, rows, or an entire sheet using named styles. **Object Parentage** is also applied to the cell type. The closer to the cell level, the higher the precedence.

Using Code

Create the cell type you want to apply and set it in the CellType property of named style and each object. CellType property can be set for the following objects:

Objects	Class	Property
Cell	Cell Class (on-line documentation)	CellType Property (on-line documentation)
Column	Column Class (on-line documentation)	CellType Property (on-line documentation)
Row	Row Class (on-line documentation)	CellType Property (on-line documentation)
Alternating rows	AlternatingRow Class (on-line documentation)	CellType Property (on-line documentation)
Named style	NamedStyle Class (on-line documentation)	CellType Property (on-line documentation)

You can also determine the cell type of the active cell using **GetCellType** ('**GetCellType Method**' in the on-line documentation) method of **SheetView** ('**SheetView Class**' in the on-line documentation) class. You can check the cell type using the typeof operator in Visual Basic and is operator in C#.

Example

This example code sets the cell type for rows, columns and specific cells.

C#

```
// Set general cell in the entire first row.
fpSpread1.ActiveSheet.Rows[0].CellType = new
FarPoint.Win.Spread.CellType.GeneralCellType();

// Set button cell for the entire second column.
FarPoint.Win.Spread.CellType.ButtonCellType buttonCell = new
FarPoint.Win.Spread.CellType.ButtonCellType();
buttonCell.Text = "Button";
fpSpread1.ActiveSheet.Columns[1].CellType = buttonCell;

// Set date-time cell only for the first row and first column.
FarPoint.Win.Spread.CellType.DateTimeCellType datecell = new
FarPoint.Win.Spread.CellType.DateTimeCellType();
datecell.DateTimeFormat = FarPoint.Win.Spread.CellType.DateTimeFormat.ShortDate;
fpSpread1.ActiveSheet.Cells[0, 0].CellType = datecell;
fpSpread1.ActiveSheet.Cells[0, 0].Value = System.DateTime.Now;
```

Visual Basic

```
' Set general cell in the entire first row.
FpSpread1.ActiveSheet.Rows(0).CellType = New
FarPoint.Win.Spread.CellType.GeneralCellType()

' Set button cell for the entire second column.
Dim buttonCell As New FarPoint.Win.Spread.CellType.ButtonCellType()
buttonCell.Text = "Button"
FpSpread1.ActiveSheet.Columns(1).CellType = buttonCell

' Set date-time cell only for the first row and first column.
Dim datecell As New FarPoint.Win.Spread.CellType.DateTimeCellType()
datecell.DateTimeFormat = FarPoint.Win.Spread.CellType.DateTimeFormat.ShortDate
FpSpread1.ActiveSheet.Cells(0, 0).CellType = datecell
FpSpread1.ActiveSheet.Cells(0, 0).Value = System.DateTime.Now
```

Understanding How Cell Types Display and Format Data

The value of the **Text** property contains the formatted data as displayed in the cell; the value of the **Value** property contains the unformatted data as saved in the model. You can use the **SheetView** (**'SheetView Class' in the on-line documentation**), **GetText** (**'GetText Method' in the on-line documentation**) and **GetValue** (**'GetValue Method' in the on-line documentation**) methods to obtain the contents of the cell, regardless of cell type.

The following table lists the editable cell types, and how each cell type works with the data, whether formatted (**Text**) or unformatted (**Value**).

Editable Cell Type	Sample Input	Formatted Data	Unformatted Data
CurrencyCellType ('CurrencyCellType Class' in the on-line documentation)	"\$10,000.00"	"\$10,000.00"	10000.00
DateTimeCellType ('DateTimeCellType Class' in the on-line documentation)	"10/29/2002"	"10/29/2002"	DateTime object of Tuesday, October 29, 2002 12:00:00 AM

GcCharMaskCellType ('GcCharMaskCellType Class' in the on-line documentation)	"123-45-6789"	"123-45-6789"	"123456789"
GcDateTimeCellType ('GcDateTimeCellType Class' in the on-line documentation)	"10/29/2002"	"10/29/2002"	DateTime object of Tuesday, October 29, 2002 12:00:00 AM
GcMaskCellType ('GcMaskCellType Class' in the on-line documentation)	"123-45-6789"	"123-45-6789"	"123456789"
GcNumberCellType ('GcNumberCellType Class' in the on-line documentation)	"10000.00"	"10000.00"	10000.00
GcTextBoxCellType ('GcTextBoxCellType Class' in the on-line documentation)	Any text	String of that text	String of that text
GcTimeSpanCellType ('GcTimeSpanCellType Class' in the on-line documentation)	"1.22:50:40"	"1.22:50:40"	TimeSpan object {1.22:50:40}
GeneralCellType ('GeneralCellType Class' in the on-line documentation)	Any data	String of that data	Depends on whether DateTime, Boolean, or Text returns the DateTime object, the Boolean value, or the Text value
MaskCellType ('MaskCellType Class' in the on-line documentation)	"123-45-6789"	"123-45-6789"	"123456789"
NumberCellType ('NumberCellType Class' in the on-line documentation)	"10000.00"	"10000.00"	10000.00
PercentCellType ('PercentCellType Class' in the on-line documentation)	"15%"	"15%"	0.15
RegularExpressionCellType ('RegularExpressionCellType Class' in the on-line documentation)	"99-999-9999"	"99-999-9999"	"99-999-9999"
TextCellType ('TextCellType Class' in the on-line documentation)	Any text	String of that text	String of that text

The following table lists the graphical cell types, and how each cell type works with the **Text** and **Value** properties.

Graphical Cell Type	Sample Input	Text Data	Value Data
BarcodeCellType ('BarcodeCellType Class' in the on-line documentation)	Picture as data	N/A	N/A
ButtonCellType ('ButtonCellType Class' in the on-line documentation) two-state	True	"1"	True
	False	"0"	False
	Not set looks false	Empty string	False
CheckBoxCellType ('CheckBoxCellType Class' in the on-line documentation) two-state	True (checked)	"True"	1
	False	"False"	0

	(unchecked)		
	Not set looks false	Empty string	False
CheckBoxCellType ('CheckBoxCellType Class' in the on-line documentation) three-state	True (checked)	"1"	1
	False (unchecked)	"0"	0
	Indeterminate (gray)	"2"	2
	Not set looks false	Empty string	0
ColorPickerCellType ('BarCodeCellType Class' in the on-line documentation)	Selected color	Color name	Color name
ComboBoxCellType ('ComboBoxCellType Class' in the on-line documentation) and MultiColumnComboBoxCellType ('MultiColumnComboBoxCellType Class' in the on-line documentation)	Any item	Text of selected item	Text of selected item or index of selected item (see the EditorValue ('EditorValue Property' in the on-line documentation) property)
	Nothing selected	Empty string	Null
GcComboBoxCellType ('GcComboBoxCellType Class' in the on-line documentation)	Any item	Text of selected item	Text of selected item or index of selected item (see the EditorValue ('EditorValue Property' in the on-line documentation) property)
	Nothing selected	Empty string	Null
HyperLinkCellType ('HyperLinkCellType Class' in the on-line documentation)	any text	Array of Boolean values: "True" (for visited link) or "False" (for not visited)	Array of Boolean values: "True" (for visited link) or "False" (for not visited)
ImageCellType ('ImageCellType Class' in the on-line documentation)	Picture as data	N/A	N/A
ListBoxCellType ('ListBoxCellType Class' in the on-line documentation)	Array	Array	Array
MultiOptionCellType ('MultiOptionCellType Class' in the on-line documentation)	Any item selected	Text of selected item	Index of selected item (numeric)
	Nothing selected	Empty string	Null
ProgressCellType ('ProgressCellType Class' in the on-line documentation)	15, between 10 and 20	"50%" (string representation)	15 (actual value)

		of numeric value)	
RichTextCellType ('RichTextCellType Class' in the on-line documentation)	String in rich text format	String in rich text format	String in rich text format
SliderCellType ('SliderCellType Class' in the on-line documentation)	4, between 0 and 10	"4" (string representation of numeric value)	4

For information on other aspects of cell display, refer to **Resizing a Cell to Fit the Data**.

Understanding How Cell Type Affects Model Data

The cell type affects how the values are stored in the model.

The following table lists the editable cell types and the data type of the value in the cell that is written to the data model.

Editable Cell Type	Data Type Written to Model
CurrencyCellType ('CurrencyCellType Class' in the on-line documentation)	Decimal
DateTimeCellType ('DateTimeCellType Class' in the on-line documentation)	Date-Time Object
GcCharMaskCellType ('GcCharMaskCellType Class' in the on-line documentation)	String
GcDateTimeCellType ('GcDateTimeCellType Class' in the on-line documentation)	Date-Time Object
GcMaskCellType ('GcMaskCellType Class' in the on-line documentation)	String
GcNumberCellType ('GcNumberCellType Class' in the on-line documentation)	Double
GcTextBoxCellType ('GcTextBoxCellType Class' in the on-line documentation)	String
GcTimeSpanCellType ('GcTimeSpanCellType Class' in the on-line documentation)	TimeSpan Object
GeneralCellType ('GeneralCellType Class' in the on-line documentation)	Depends whether Date-Time, Boolean, or String
MaskCellType ('MaskCellType Class' in the on-line documentation)	String
NumberCellType ('NumberCellType Class' in the on-line documentation)	Double
PercentCellType ('PercentCellType Class' in the on-line documentation)	Double
RegularExpressionCellType ('RegularExpressionCellType Class' in the on-line documentation)	String
TextCellType ('TextCellType Class' in the on-line documentation)	String

The following table lists the graphical cell types and the data type of the value in the cell that is written to the data model.

Graphical Cell Type

BarcodeCellType ('BarcodeCellType Class' in the on-line documentation)

ButtonCellType ('ButtonCellType Class' in the on-line documentation) two-state

ButtonCellType ('ButtonCellType Class' in the on-line documentation) one-state

CheckBoxCellType ('CheckBoxCellType Class' in the on-line documentation) three-state

CheckBoxCellType ('CheckBoxCellType Class' in the on-line documentation) two-state

ColorPickerCellType ('BarcodeCellType Class' in the on-line documentation)

ComboBoxCellType ('ComboBoxCellType Class' in the on-line documentation) and **MultiColumnComboBoxCellType** ('MultiColumnComboBoxCellType Class' in the on-line documentation)

GcComboBoxCellType ('GcComboBoxCellType Class' in the on-line documentation)

HyperLinkCellType ('HyperLinkCellType Class' in the on-line documentation)

ImageCellType ('ImageCellType Class' in the on-line documentation)

ListBoxCellType ('ListBoxCellType Class' in the on-line documentation)

MultiOptionCellType ('MultiOptionCellType Class' in the on-line documentation)

ProgressCellType ('ProgressCellType Class' in the on-line documentation)

RichTextCellType ('RichTextCellType Class' in the on-line documentation)

SliderCellType ('SliderCellType Class' in the on-line documentation)

Data Type Written to Model

Value of barcode

Integer

Null

Integer (0 = false, 1= true, 2 = indeterminate)

Boolean

Null

Depends on value of **EditorValue** property. String, if EditorValue = String or item data. Integer, if EditorValue = index

Depends on value of **EditorValue** property. String, if EditorValue = String or item data. Integer, if EditorValue = index

Array of Boolean values (whether each link is clicked or unclicked)

System.Drawing.Color or Null

Array

Depends on value of **EditorValue** property. String, if EditorValue = String or item data, Integer, if EditorValue = index

Double

String

Integer

Working with Editable Cell Types

You can work with the editable cell types as described in the following topics:

- **Setting a General Cell**
- **Setting a Text Cell**
- **Setting a Date-Time Cell**
- **Setting a Number Cell**

- **Setting a Currency Cell**
- **Setting a Mask Cell**
- **Setting a Percent Cell**
- **Setting a Regular Expression Cell**
- **Setting a Rich Text Cell**

For other cell types, refer to **Working with Graphical Cell Types**.

Setting a General Cell

The general cell is the default cell type for the cells in the sheets. Unless you specify another cell type, the component assigns the general cell type to the cells. The general cell can be used as is for entering text or numbers where formatting is not critical or the type of data is not tied to a specific data type. For specific cell types where formatting is important, see the **CurrencyCellType** ('**CurrencyCellType Class**' in the on-line documentation), **DateTimeCellType** ('**DateTimeCellType Class**' in the on-line documentation), **NumberCellType** ('**NumberCellType Class**' in the on-line documentation), and **PercentCellType** ('**PercentCellType Class**' in the on-line documentation) cells.

You use the **GeneralCellType** ('**GeneralCellType Class**' in the on-line documentation) class to set the general cell and its properties.

With the general cell you can format the displayed values regardless of the user input. The general cell type includes a formatter that takes the data entered by the user and assigns it one of the known formats and data types. Therefore, you need not worry about setting cell types because the general cell type handles inputs of many kinds.

The openness of the general cell can be restricted if you want to allow the user to enter data in any acceptable format, but want it to be formatted and displayed in a specific way. To allow the user to enter data in any acceptable format and format and display the data in a specific way, adjust the formatter for the general cell type. To do this, specify a format string for the general cell and the general formatter parses the user-entered data, but when the data is displayed, your custom format is used rather than the format used by the end user. You can use the **FormatString** property. Here is an example:

Visual Basic

```
Dim gnrCell As New FarPoint.Win.Spread.GeneralCellType
gnrCell.FormatString = "#,###.00"
fpSpread1.Sheets(0).Cells(1, 1).CellType = gnrCell
```

For more information on the properties and methods of this cell type, refer to the **GeneralCellType** ('**GeneralCellType Class**' in the on-line documentation) class.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet in which the cells appear.
5. In the property list, select the **Cells** property and then click the button to display the **Cell, Column, and Row Editor**.
6. Select the cells for which you want to set the cell type.
7. In the property list, select the **CellType** property and choose the **General** cell type.
8. Expand the list of properties under the **CellType** property. Select and set these specific properties as needed.
9. Click **OK** to close the **Cell, Column, and Row Editor**.
10. Click **OK** to close the **SheetView Collection Editor**.

Using Code

1. Define the general cell by creating an instance of the **GeneralCellType** ('GeneralCellType Class' in the **on-line documentation**) class.
2. Set properties for the class.
3. Assign the general cell type to a cell or range of cells by setting the **CellType** ('CellType Property' in the **on-line documentation**) property for a cell, column, row, or style to the **GeneralCellType** ('GeneralCellType Class' in the **on-line documentation**) object.

Example

This example sets a cell to be a general cell.

C#

```
FarPoint.Win.Spread.CellType.GeneralCellType gnrlcell = new
FarPoint.Win.Spread.CellType.GeneralCellType();
fpSpread1.ActiveSheet.Cells[1, 1].CellType = gnrlcell;
;
```

VB

```
Dim gnrlcell As New FarPoint.Win.Spread.CellType.GeneralCellType()
fpSpread1.ActiveSheet.Cells(1, 1).CellType = gnrlcell
```

Using the Spread Designer

1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the **General** cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Select and set those properties as needed.
Or right-click on the cell or cells and select **Cell Type**. From the list, select **General**. In the **CellType** editor, set the properties you need. Click **Apply**.
3. From the File menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting a Text Cell

You can create a text cell that allows only text to be displayed or treats the contents of a cell as only text.

You can also specify if the text shows up as all lower case, upper case, or normal with the **CharacterCasing** ('CharacterCasing Property' in the **on-line documentation**) property. The **CharacterSet** ('CharacterSet Property' in the **on-line documentation**) property allows you to specify numbers only, letters only, numbers and letters, or any ASCII characters.

You use the **TextCellType** ('TextCellType Class' in the **on-line documentation**) class to set the text cell and its properties.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet in which the cells appear.
5. In the property list, select the **Cells** property and then click the button to display the **Cell, Column, and Row**

Editor.

6. Select the cells for which you want to set the cell type.
7. In the property list, select the **CellType** property and choose the **Text** cell type.
8. Expand the list of properties under the **CellType** property. Select and set these specific properties as needed.
9. Click **OK** to close the **Cell, Column, and Row Editor**.
10. Click **OK** to close the **SheetView Collection Editor**.

Using Code

1. Define the text cell by creating an instance of the **TextCellType ('TextCellType Class' in the on-line documentation)** class.
2. Set properties for the class.
3. Assign the text cell type to a cell or range of cells by setting the **CellType** property for a cell, column, row, or style to the **TextCellType ('TextCellType Class' in the on-line documentation)** object.

Example

This example creates a text cell with a maximum length.

C#

```
FarPoint.Win.Spread.CellType.TextCellType tcell = new
FarPoint.Win.Spread.CellType.TextCellType();
tcell.CharacterCasing = CharacterCasing.Upper;
tcell.CharacterSet = FarPoint.Win.Spread.CellType.CharacterSet.Ascii;
tcell.MaxLength = 30;
tcell.Multiline = true;
fpSpread1.ActiveSheet.Cells[0, 0].Text = "This is a text cell.";
fpSpread1.ActiveSheet.Cells[0, 0].CellType = tcell;
```

VB

```
Dim tcell As New FarPoint.Win.Spread.CellType.TextCellType()
tcell.CharacterCasing = CharacterCasing.Upper
tcell.CharacterSet = FarPoint.Win.Spread.CellType.CharacterSet.Ascii
tcell.MaxLength = 40
tcell.Multiline = True
fpSpread1.ActiveSheet.Cells(0, 0).Text = "This is a text cell."
fpSpread1.ActiveSheet.Cells(0, 0).CellType = tcell
```

Using the Spread Designer

1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the **Text** cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Select and set those properties as needed. Or right-click on the cell or cells and select **Cell Type**. From the list, select **Text**. In the **CellType** editor, set the properties you need. Click **Apply**.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting a Date-Time Cell

You can set a cell to display date and time and only allow user inputs of date and time using the date-time cell. You

determine the format of the date and time to display.

Date and Time
Monday, September 26, 2005 3:36:45 pm
Tuesday, September 27, 2005 3:36:45 pm
Friday, September 23, 2005 3:36:45 pm

You use the **DateTimeCellType** ('**DateTimeCellType Class**' in the on-line documentation) class to set the date-time cell and its properties.

The default values use the Regional Settings or Regional Options in the Windows environment. You can specify the format using several properties. For a complete list of date and time formats, refer to the **DateTimeFormat** ('**DateTimeFormat Enumeration**' in the on-line documentation) enumeration and the **DateTimeFormat** ('**DateTimeFormat Property**' in the on-line documentation) property. If a date time cell displays dates and times in long date and time format, and the current date and time is "10/29/2002 11:10:01", the **Text** property returns "Tuesday, October 29, 2002 11:10:01 AM" as the formatted data of the cell. The **Value** property returns the date-time object of that date and time.

The date-time cell also has an **EditorValue** ('**EditorValue Property**' in the on-line documentation) property that allows you to determine what is written to the data model.

By default, in a date-time cell, if you double-click on the cell in edit mode at run-time, a pop-up calendar (or clock) appears. You can determine whether to allow this, and you can specify the text that displays on the **OK** and **Cancel** buttons. For more information, refer to **Customizing the Pop-Up Date-Time Control**.

Spread uses the **TimeDefault** ('**TimeDefault Property**' in the on-line documentation) property to fill in the time portion that is not set into the cell. When you use the popup calendar to set the date for the cell, the time is set to midnight. If you want a different time, you would need to use the **SubEditorClosed** event and change the value in the cell. (You can also create your own sub-editor to create a clock and calendar form to pop up for the cell.) You can look in **ISubEditor** interface for more information on how to implement this. As for the value, the **Value** ('**Value Property**' in the on-line documentation) property returns is a **DateTime** object that encapsulates both the date and time. Query the **TimeOfDay** property from the returned **DateTime** object to get the time of day.

For more information on the properties and methods of this cell type, refer to the **DateTimeCellType** ('**DateTimeCellType Class**' in the on-line documentation) class.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet in which the cells appear.
5. In the property list, select the **Cells** property and then click the button to display the **Cell, Column, and Row Editor**.
6. Select the cells for which you want to set the cell type.
7. In the property list, select the **CellType** property and choose the **DateTime** cell type.
8. Expand the list of properties under the **CellType** property. Select and set these specific properties as needed.
9. Click **OK** to close the **Cell, Column, and Row Editor**.
10. Click **OK** to close the **SheetView Collection Editor**.

Using Code

1. Define the date-time cell by creating an instance of the **DateTimeCellType** ('**DateTimeCellType Class**' in the on-line documentation) class.

2. Specify the message to display if invalid.
3. Specify the format of the date to display.
4. Assign the date-time cell type to a cell or range of cells by setting the **CellType** property for a cell, column, row, or style to the **DateTimeCellType** (**'DateTimeCellType Class' in the on-line documentation**) object.

Example

Display the date as Tuesday, March 04 (day of week, month and number of day) in the second row, second column cell.

C#

```
FarPoint.Win.Spread.CellType.DateTimeCellType datecell = new
FarPoint.Win.Spread.CellType.DateTimeCellType();
datecell.DateSeparator = " | ";
datecell.TimeSeparator = ".";
datecell.DateTimeFormat =
FarPoint.Win.Spread.CellType.DateTimeFormat.ShortDateWithTime;
datecell.MaximumDate = new System.DateTime(2100, 1, 1);
datecell.MinimumDate = new System.DateTime(1990, 12, 31);
datecell.MaximumTime = new System.TimeSpan(15, 59, 59);
datecell.MinimumTime = new System.TimeSpan(11, 0, 0);
fpSpread1.ActiveSheet.Columns[1].Width = 175;
fpSpread1.ActiveSheet.Cells[1, 1].CellType = datecell;
fpSpread1.ActiveSheet.Cells[1, 1].Value = System.DateTime.Now;
```

VB

```
Dim datecell As New FarPoint.Win.Spread.CellType.DateTimeCellType()
datecell.DateSeparator = " | "
datecell.TimeSeparator = "."
datecell.DateTimeFormat = FarPoint.Win.Spread.CellType.DateTimeFormat.ShortDateWithTime
datecell.MaximumDate = new System.DateTime(2100, 1, 1)
datecell.MinimumDate = new System.DateTime(1990, 12, 31)
datecell.MaximumTime = new System.TimeSpan(15, 59, 59)
datecell.MinimumTime = new System.TimeSpan(11, 0, 0)
fpSpread1.ActiveSheet.Columns(1).Width = 175
fpSpread1.ActiveSheet.Cells(1, 1).CellType = datecell
fpSpread1.ActiveSheet.Cells(1, 1).Value = System.DateTime.Now
```

Using the Spread Designer

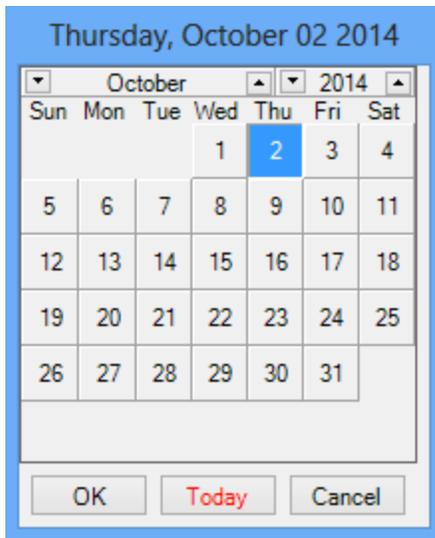
1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the **DateTime** cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Select and set those properties as needed.
Or right-click on the cell or cells and select **Cell Type**. From the list, select **DateTime**. In the **CellType** editor, set the properties you need. Click **Apply**.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Customizing the Pop-Up Date-Time Control

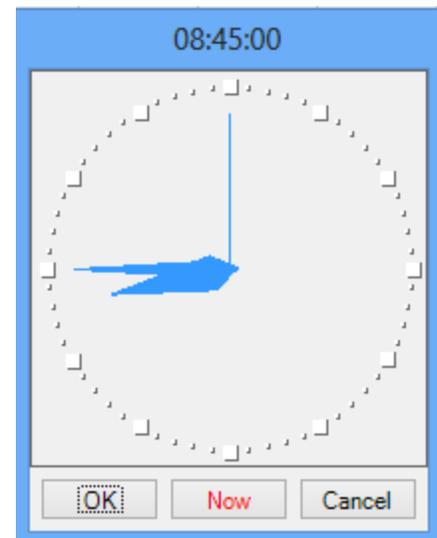
If you press F4 or double-click a date-time cell when it is in edit mode, a pop-up calendar (or pop-up clock) appears, as shown in the following figures. The calendar control appears if you have the date-time cell set to any format besides TimeOnly format. The clock control appears if you have the format set to TimeOnly. The date you choose from the

calendar (or the time you choose from the clock) is placed in the date-time cell. If you want the present date and time, in the calendar control click **Today**; if you want the present time, in the clock control click **Now**.

Pop-Up Calendar Control



Pop-Up Clock Control



 You can also display the clock control if the **DateTimeFormat** ('**DateTimeFormat Property**' in the on-line documentation) property is set to UserDefined and the **UserDefinedFormat** ('**UserDefinedFormat Property**' in the on-line documentation) property uses a time setting such as HH:mm.

You can specify normal and abbreviated day names, normal and abbreviated month names, and the text for the buttons at the bottom of the control. Use the **SetCalendarText** ('**SetCalendarText Method**' in the on-line documentation) method of the **DateTimeCellType** ('**DateTimeCellType Class**' in the on-line documentation) class to set these.

Note that the text appears centered on the button. If you set custom text for the buttons, try to limit your text to eight or nine characters in length. The button will display ten characters, but the first and last appear very close to the edges.

When using the controls, you must click the **OK** or **Cancel** button to close the control. The **Today** (or **Now**) button simply sets the value in the cell to the current date (or time).

For more information on setting the format of the date-time cell, refer to the **DateTimeFormat** ('**DateTimeFormat Enumeration**' in the on-line documentation) enumeration.

For information on the editable cell types, refer to **Working with Editable Cell Types**.

For information on other features of cell types, refer to **Understanding Additional Features of Cell Types**.

Using Code

1. Define the date-time cell by creating an instance of the **DateTimeCellType** ('**DateTimeCellType Class**' in the on-line documentation) class.
2. Specify the values for the buttons or for all the day and month names and the buttons.
3. Assign the date-time cell type to a cell or range of cells by setting the **CellType** property for a cell, column, row, or style to the **DateTimeCellType** ('**DateTimeCellType Class**' in the on-line documentation) object.

Example

The following example sets the text of the buttons and specifies day and month names in array lists.

C#

```

FarPoint.Win.Spread.CellType.DateTimeCellType datecell = new
FarPoint.Win.Spread.CellType.DateTimeCellType();
string[] daynames = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday"};
string[] months = {"January", "February", "March", "April", "May", "June", "July",
"August", "September", "October", "November", "December"};
string[] dayabbrev = {"Su", "My", "Ty", "Wy", "Th", "Fy", "Sy"};
string[] mthabbrev = {"Jy", "Fy", "Mh", "Al", "My", "Jn", "Jl", "At", "Sr", "Or", "Nr", "Dr"};
string okbuttn = "Fine";
string cancelb = "Quit";
datecell.DateTimeFormat = FarPoint.Win.Spread.CellType.DateTimeFormat.UserDefined;
datecell.UserDefinedFormat = "dddd MMMM d, yyyy";
datecell.SetCalendarText(daynames, months, dayabbrev, mthabbrev, okbuttn, cancelb);
fpSpread1.ActiveSheet.Cells[1, 1].CellType = datecell;
fpSpread1.ActiveSheet.Cells[1, 1].Value = System.DateTime.Now;
fpSpread1.ActiveSheet.Columns[1].Width = 130;

```

VB

```

FarPoint.Win.Spread.CellType.DateTimeCellType datecell = new
FarPoint.Win.Spread.CellType.DateTimeCellType();
string[] daynames = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday"};
string[] months = {"January", "February", "March", "April", "May", "June", "July",
"August", "September", "October", "November", "December"};
string[] dayabbrev = {"Su", "My", "Ty", "Wy", "Th", "Fy", "Sy"};
string[] mthabbrev = {"Jy", "Fy", "Mh", "Al", "My", "Jn", "Jl", "At", "Sr", "Or", "Nr", "Dr"};
string okbuttn = "Fine";
string cancelb = "Quit";
datecell.DateTimeFormat = FarPoint.Win.Spread.CellType.DateTimeFormat.UserDefined;
datecell.UserDefinedFormat = "dddd MMMM d, yyyy";
datecell.SetCalendarText(daynames, months, dayabbrev, mthabbrev, okbuttn, cancelb);
fpSpread1.ActiveSheet.Cells[1, 1].CellType = datecell;
fpSpread1.ActiveSheet.Cells[1, 1].Value = System.DateTime.Now;
fpSpread1.ActiveSheet.Columns[1].Width = 130;

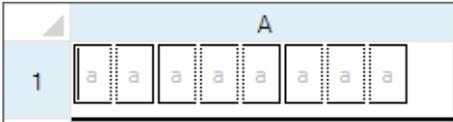
```

Using the Spread Designer

1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the **DateTime** cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Set properties such as **ShortDayNames** and **ShortMonthNames** to change the calendar text. Set other properties as needed.
Or right-click on the cell or cells and select **Cell Type**. From the list, select the cell type. In the **CellType** editor, set the properties you need. Click **Apply**.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting a GcCharMask Cell

The GcCharMask cell uses character boxes for the mask. The user can customize the appearance of the boxes and the input value.



The following image displays the characters you can use for the mask (**FormatString** ('**FormatString Property**' in the **on-line documentation**) property).

Pattern	Description
<input type="checkbox"/> A	Upper case alphabet (A~Z)
<input type="checkbox"/> A	DBCS upper case alphabet (A~Z)
<input type="checkbox"/> a	Lower case alphabet (a~z)
<input type="checkbox"/> a	DBCS lower case alphabet (a~z)
<input type="checkbox"/> K	Katakana
<input type="checkbox"/> K	DBCS katakana
<input type="checkbox"/> 9	Numbers
<input type="checkbox"/> 9	DBCS numbers
<input type="checkbox"/> #	Numbers and number related symbols (0~9, + - \$ % \ , .)
<input type="checkbox"/> #	DBCS numbers and number related symbols (0~9, + - \$ % \ , .)
<input type="checkbox"/> @	Symbols
<input type="checkbox"/> @	DBCS symbols
<input type="checkbox"/> B	Binary numbers
<input type="checkbox"/> B	DBCS binary numbers
<input type="checkbox"/> X	Hexadecimal (0~9, A~F, a-f)
<input type="checkbox"/> X	DBCS Hexadecimal (0~9, A~F, a-f)
<input type="checkbox"/> J	Hiragana
<input type="checkbox"/> Z	All DBCS characters
<input type="checkbox"/> H	All SBCS characters
<input type="checkbox"/> D	All DBCS characters except for surrogates
<input type="checkbox"/> M	All Shift-JIS characters
<input type="checkbox"/> I	All JIS X 0208 characters
<input type="checkbox"/> N	Only large katakana
<input type="checkbox"/> N	Only DBCS large katakana
<input type="checkbox"/> G	Only large hiragana
<input type="checkbox"/> T	Surrogate char
<input type="checkbox"/> ^	Not include char

You can specify a built-in collection of character boxes with the **CharBoxInfoCollection** ('**CharBoxInfoCollection Class**' in the **on-line documentation**) class or you can customize the number and layout of the boxes with the **LiteralBoxInfo** ('**LiteralBoxInfo Class**' in the **on-line documentation**), **InputBoxInfo** ('**InputBoxInfo Class**' in the **on-line documentation**), and **SeparatorBoxInfo** ('**SeparatorBoxInfo Class**' in the **on-line documentation**) classes. Use the **AddRange** ('**AddRange Method**' in the **on-line documentation**) method to add the built-in boxes or the custom boxes in the preferred order.

For a complete list of properties and methods for the GcCharMask cell, refer to the **GcCharMaskCellType** ('**GcCharMaskCellType Class**' in the **on-line documentation**) class.

Using Code

1. Define the GcCharMask cell by creating an instance of the **GcCharMaskCellType** ('**GcCharMaskCellType Class**' in the **on-line documentation**) class.
2. Set properties for the class.
3. Assign the cell type to a cell or range of cells by setting the **CellType** property for a cell, column, row, or style to the **GcCharMaskCellType** object.

Example

This example creates a GcCharMask cell.

C#

```
GrapeCity.Win.Spread.InputMan.CellType.GcCharMaskCellType gc = new
GrapeCity.Win.Spread.InputMan.CellType.GcCharMaskCellType();
gc.AcceptsArrowKeys = FarPoint.Win.SuperEdit.AcceptsArrowKeys.CtrlArrows;
gc.AcceptsCrLf = GrapeCity.Win.Spread.InputMan.CellType.CrLfMode.Filter;
gc.AllowSpace = GrapeCity.Win.Spread.InputMan.CellType.AllowSpace.Wide;

gc.CharBoxes.AddRange(GrapeCity.Win.Spread.InputMan.CellType.CharBoxInfoCollection.Number1);
gc.CharBoxSpacing = 2;
gc.ClipContent = GrapeCity.Win.Spread.InputMan.CellType.ClipContent.ExcludeLiterals;

gc.ExitOnLastChar = true;
gc.FocusPosition =
GrapeCity.Win.Spread.InputMan.CellType.EditorBaseFocusCursorPosition.FirstInputPosition;
gc.FormatString = "a";
gc.PaintByControl = true;
gc.RecommendedValue = "aaaaaaaaaa";
gc.ShowRecommendedValue = true;
gc.UseSpreadDropDownButtonRender = true;
fpSpread1.ActiveSheet.Cells[0, 0].CellType = gc;
fpSpread1.ActiveSheet.Columns[0].Width = 200;
fpSpread1.ActiveSheet.Rows[0].Height = 40;
```

VB

```
Dim GC As New GrapeCity.Win.Spread.InputMan.CellType.GcCharMaskCellType()
GC.AcceptsArrowKeys = FarPoint.Win.SuperEdit.AcceptsArrowKeys.CtrlArrows
GC.AcceptsCrLf = GrapeCity.Win.Spread.InputMan.CellType.CrLfMode.Filter
GC.AllowSpace = GrapeCity.Win.Spread.InputMan.CellType.AllowSpace.Wide

GC.CharBoxes.AddRange(GrapeCity.Win.Spread.InputMan.CellType.CharBoxInfoCollection.Number1)
GC.CharBoxSpacing = 2
GC.ClipContent = GrapeCity.Win.Spread.InputMan.CellType.ClipContent.ExcludeLiterals

GC.ExitOnLastChar = True
GC.FocusPosition =
GrapeCity.Win.Spread.InputMan.CellType.EditorBaseFocusCursorPosition.FirstInputPosition
GC.FormatString = "a"
GC.PaintByControl = True
GC.RecommendedValue = "aaaaaaaaaa"
GC.ShowRecommendedValue = True
GC.UseSpreadDropDownButtonRender = True
fpSpread1.ActiveSheet.Cells(0, 0).CellType = GC
fpSpread1.ActiveSheet.Columns(0).Width = 200
fpSpread1.ActiveSheet.Rows(0).Height = 40
```

Using the Spread Designer

1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the **GcCharMask** cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Select and set those properties as needed.
Or right-click on the cell or cells and select **Cell Type**. From the list, select **GcCharMask**. In the **CellType** editor, set the properties you need. Click **Apply**.

- From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting a GcDateTime Cell

You can use the GcDateTime cell to display date and time values. The GcDateTime cell allows the user to pick a date from a calendar drop-down or type in the cell. This cell is part of the GrapeCity.Win.PluginInputMan assembly.

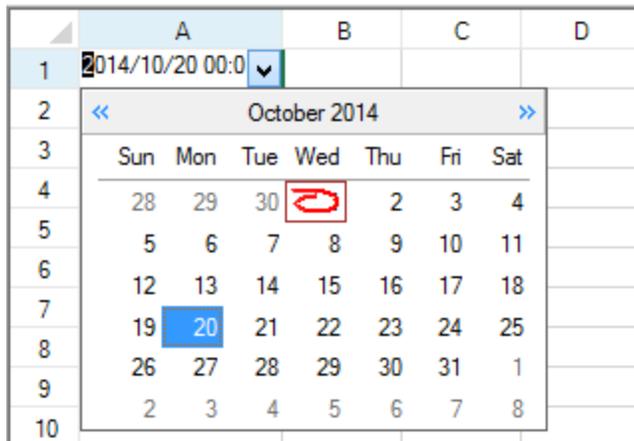
The GcDateTime cell supports different calendar styles from the DateTime cell. You can specify which fields to display with the **DisplayFields ('DisplayFields Property' in the on-line documentation)** property.

You can specify the focus position when the cell gets focus with the **FocusPosition ('FocusPosition Property' in the on-line documentation)** property and you can specify whether focus leaves the cell after typing the last character in the date value with the **ExitOnLastChar ('ExitOnLastChar Property' in the on-line documentation)** property.

You can use the **ShortcutKeys ('ShortcutKeys Property' in the on-line documentation)** property to map keys to actions for the GcDateTime and GcTextBox cells. In edit mode, these shortcut keys have precedence over the Spread input maps. The cell uses the Spread input maps when not in edit mode.

You can specify the type of drop-down to display with the **DropDownType ('DropDownType Property' in the on-line documentation)** property. The drop-down picker type is easier to use in a touch environment.

For a complete list of properties, see the **GcDateTimeCellType ('GcDateTimeCellType Class' in the on-line documentation)** class.



Using the Properties Window

- At design time, in the **Properties** window, select the Spread component.
- Select the **Sheets** property.
- Click the button to display the **SheetView Collection Editor**.
- In the **Members** list, select the sheet in which the cells appear.
- In the property list, select the **Cells** property and then click the button to display the **Cell, Column, and Row Editor**.
- Select the cells for which you want to set the cell type.
- In the property list, select the **CellType** property and choose the **GcDateTime** cell type.
- Expand the list of properties under the **CellType** property. Select and set these specific properties as needed.
- Click **OK** to close the **Cell, Column, and Row Editor**.
- Click **OK** to close the **SheetView Collection Editor**.

Using Code

1. Define the text cell by creating an instance of the **GcDateTimeCellType** ('**GcDateTimeCellType Class**' in the on-line documentation) class.
2. Set properties for the class.
3. Assign the cell type to a cell or range of cells by setting the **CellType** property for a cell, column, row, or style to the **GcDateTimeCellType** object.

Example

This example creates a GcDateTime cell.

C#

```
GrapeCity.Win.Spread.InputMan.CellType.GcDateTimeCellType inputcell = new
GrapeCity.Win.Spread.InputMan.CellType.GcDateTimeCellType();
inputcell.EditMode = GrapeCity.Win.Spread.InputMan.CellType.EditMode.Overwrite;
fpSpread1.Sheets[0].Cells[0, 0].CellType = inputcell;
```

VB

```
Dim inputcell As New GrapeCity.Win.Spread.InputMan.CellType.GcDateTimeCellType
inputcell.EditMode = GrapeCity.Win.Spread.InputMan.CellType.EditMode.Overwrite
fpSpread1.Sheets(0).Cells(0, 0).CellType = inputcell
```

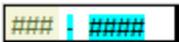
Using the Spread Designer

1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the **Text** cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Select and set those properties as needed. Or right-click on the cell or cells and select **Cell Type**. From the list, select **GcDateTime**. In the **CellType** editor, set the properties you need. Click **Apply**.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting a GcMask Cell

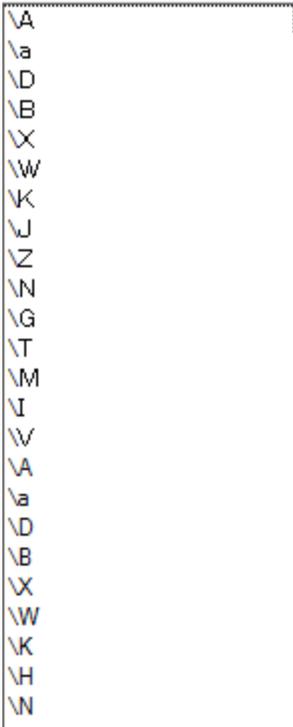
You can use a GcMask cell to limit the characters a user can type in a cell. You can specify literal characters and the pattern for the characters the user is allowed to enter.

The following image displays a pattern of three characters, followed by a single literal, and then a pattern of four characters.



Use the **MaskPatternFieldInfo** ('**MaskPatternFieldInfo Class**' in the on-line documentation) class to specify the pattern you want to use for the characters in the cell. Use the **MaskLiteralFieldInfo** ('**MaskLiteralFieldInfo Class**' in the on-line documentation) to specify the literal characters for the mask. Use the **MaskFieldInfo** ('**MaskFieldInfo Class**' in the on-line documentation) class to specify the order of the pattern and literal objects. You can specify a collection of pre-defined values with the **MaskEnumerationFieldInfo** ('**MaskEnumerationFieldInfo Class**' in the on-line documentation) class.

The following image displays the available pattern characters as shown in the designer.



The following table lists the single-byte character-set available for the pattern:

Character(s)	Description
\A	Matches any upper case alphabetical letter [A-Z].
\a	Matches any lower case alphabetical letter [a-z].
\D	Matches any decimal digit [0-9].
\B	Matches binary digit [0-1].
\X	Matches hexadecimal value [0-9A-Fa-f].
\W	Matches any word character [a-zA-Z_0-9].
\K	Matches SBCS Katakana
\H	Matches all SBCS characters.
\N	Matches all SBCS big Katakana.

The following table lists the double-byte character-set available for the pattern:

Character(s)	Description
\A	Matches any upper case DBCS alphabetical character [A-Z].
\a	Matches any lower case DBCS alphabetical character [a-z].
\D	Matches any DBCS decimal digit [0-9].
\B	Matches DBCS binary digits [0-1].
\X	Matches DBCS hexadecimal digits [0-9 A-F a-f].
\W	Matches any DBCS word character [a-z A-Z_0-9].
\K	DBCS Katakana

\J	Hiragana
\Z	All DBCS characters.
\N	Matches DBCS big Katakana.
\G	Matches DBCS big Hiragana.
\T	Matches four-bit characters.

For a complete list of properties and methods for the GcMask cell, refer to the **GcMaskCellType ('GcMaskCellType Class' in the on-line documentation)** class.

Using Code

1. Define the GcMask cell by creating an instance of the **GcMaskCellType ('GcMaskCellType Class' in the on-line documentation)** class.
2. Set the pattern or literal characters.
3. Set properties for the class.
4. Assign the cell type to a cell or range of cells by setting the **CellType** property for a cell, column, row, or style to the **GcMaskCellType** object.

Example

This example creates a GcMask cell.

C#

```
GrapeCity.Win.Spread.InputMan.CellType.GcMaskCellType gcMask = new
GrapeCity.Win.Spread.InputMan.CellType.GcMaskCellType();

        GrapeCity.Win.Spread.InputMan.CellType.Fields.MaskPatternFieldInfo mpf
= new GrapeCity.Win.Spread.InputMan.CellType.Fields.MaskPatternFieldInfo("\\D{3}", 3,
3);

        mpf.AutoConvert = true;
        mpf.BackColor = Color.Beige;
        mpf.ForeColor = Color.DarkOliveGreen;
        mpf.Font = SystemFonts.DefaultFont;
        mpf.Name = "MaskPatternFieldInfo";
        mpf.Padding = new System.Windows.Forms.Padding(3);

        GrapeCity.Win.Spread.InputMan.CellType.Fields.MaskLiteralFieldInfo mlf
= new GrapeCity.Win.Spread.InputMan.CellType.Fields.MaskLiteralFieldInfo("-");
        mlf.BackColor = Color.Aqua;
        mlf.ForeColor = Color.Black;
        mlf.Margin = new System.Windows.Forms.Padding(4);
        mlf.Name = "MaskLiteralFieldInfo";

        GrapeCity.Win.Spread.InputMan.CellType.Fields.MaskPatternFieldInfo mlf2
= new GrapeCity.Win.Spread.InputMan.CellType.Fields.MaskPatternFieldInfo("\\D{4}", 4,
4);

        mlf2.BackColor = Color.Aqua;
        mlf2.ForeColor = Color.Black;
        mlf2.Margin = new System.Windows.Forms.Padding(4);
        mlf2.Name = "MaskLiteralFieldInfo2";

        gcMask.Fields.AddRange(new
GrapeCity.Win.Spread.InputMan.CellType.Fields.MaskFieldInfo[] { mpf, mlf, mlf2});
```

```

gcMask.PaintByControl = true;
gcMask.PromptChar = '#';
gcMask.RecommendedValue = "1234567";
gcMask.ShowRecommendedValue = true;

fpSpread1.ActiveSheet.Cells[0, 0].CellType = gcMask;

```

VB

```

Dim gcMask As New GrapeCity.Win.Spread.InputMan.CellType.GcMaskCellType()

    Dim mpf As New
GrapeCity.Win.Spread.InputMan.CellType.Fields.MaskPatternFieldInfo("\D{3}", 3, 3)
    mpf.AutoConvert = True
    mpf.BackColor = Color.Beige
    mpf.ForeColor = Color.DarkOliveGreen
    mpf.Font = SystemFonts.DefaultFont
    mpf.Name = "MaskPatternFieldInfo"
    mpf.Padding = New System.Windows.Forms.Padding(3)

    Dim mlf As New
GrapeCity.Win.Spread.InputMan.CellType.Fields.MaskLiteralFieldInfo("-")
    mlf.BackColor = Color.Aqua
    mlf.ForeColor = Color.Black
    mlf.Margin = New System.Windows.Forms.Padding(4)
    mlf.Name = "MaskLiteralFieldInfo"

    Dim mlf2 As New
GrapeCity.Win.Spread.InputMan.CellType.Fields.MaskPatternFieldInfo("\D{4}", 4, 4)
    mlf2.BackColor = Color.Aqua
    mlf2.ForeColor = Color.Black
    mlf2.Margin = New System.Windows.Forms.Padding(4)
    mlf2.Name = "MaskLiteralFieldInfo2"

    gcMask.Fields.AddRange(New
GrapeCity.Win.Spread.InputMan.CellType.Fields.MaskFieldInfo() {mpf, mlf, mlf2})
    gcMask.PaintByControl = True
    gcMask.PromptChar = "#"
    gcMask.RecommendedValue = "1234567"
    gcMask.ShowRecommendedValue = True

fpSpread1.ActiveSheet.Cells(0, 0).CellType = gcMask

```

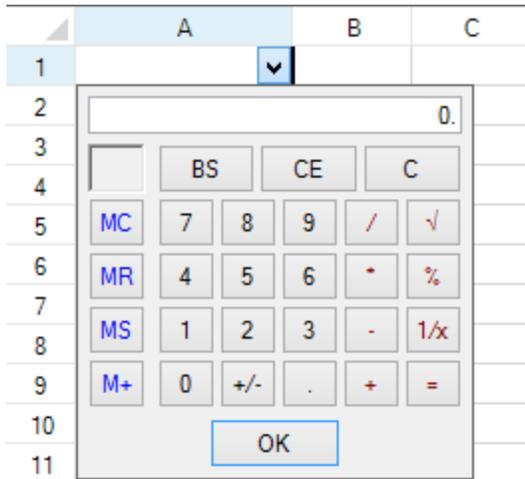
Using the Spread Designer

1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the **GcMask** cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Select and set those properties as needed.
Or right-click on the cell or cells and select **Cell Type**. From the list, select **GcMask**. In the **CellType** editor, set the properties you need. Click **Apply**.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting a GcNumber Cell

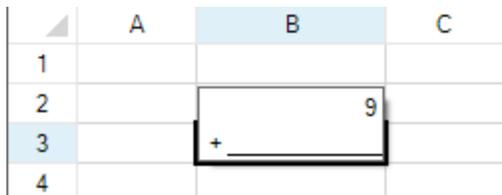
You can create a number cell that displays a side button and calculator. The **GcNumberCellType** (**'GcNumberCellType Class' in the on-line documentation**) cell is part of the GrapeCity.Win.PluginInputMan assembly.

Select the side button to display the drop-down calculator as shown in the following image. Select **OK** to close the calculator.



You can specify whether to display 0 if the cell value is null with the **AllowDeleteToNull** (**'AllowDeleteToNull Property' in the on-line documentation**) property.

You can display the pop-up calculator using the Ctrl key and the add, subtract, multiply, or divide key on the number pad while the cell is in edit mode. Press Enter to finish the calculation and accept the value. The following image displays the pop-up calculator.



For a complete list of properties, see the **GcNumberCellType** (**'GcNumberCellType Class' in the on-line documentation**) class.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet in which the cells appear.
5. In the property list, select the **Cells** property and then click the button to display the **Cell, Column, and Row Editor**.
6. Select the cells for which you want to set the cell type.
7. In the property list, select the **CellType** property and choose the **GcNumber** cell type.
8. Expand the list of properties under the **CellType** property. Select and set these specific properties as needed.
9. Click **OK** to close the **Cell, Column, and Row Editor**.
10. Click **OK** to close the **SheetView Collection Editor**.

Using Code

1. Define the cell by creating an instance of the **GcNumberCellType** ('**GcNumberCellType Class**' in the **on-line documentation**) class.
2. Set properties for the class.
3. Assign the cell type to a cell or range of cells by setting the **CellType** property for a cell, column, row, or style to the **GcNumberCellType** object.

Example

This example creates a GcNumber cell.

C#

```
GrapeCity.Win.Spread.InputMan.CellType.GcNumberCellType ncell = new
GrapeCity.Win.Spread.InputMan.CellType.GcNumberCellType();
fpSpread1.Sheets[0].Cells[0, 0].CellType = ncell;
```

VB

```
Dim ncell As New GrapeCity.Win.Spread.InputMan.CellType.GcNumberCellType()
fpSpread1.Sheets(0).Cells(0, 0).CellType = ncell
```

Using the Spread Designer

1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the **GcNumber** cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Select and set those properties as needed.
Or right-click on the cell or cells and select **Cell Type**. From the list, select **GcNumber**. In the **CellType** editor, set the properties you need. Click **Apply**.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting a GcTextBox Cell

You can create a text cell that displays text and allows you to specify patterns of allowed characters. The **GcTextBoxCellType** ('**GcTextBoxCellType Class**' in the **on-line documentation**) cell is part of the GrapeCity.Win.PluginInputMan assembly.

You can specify an automatic complete mode and a custom source with the **AutoCompleteMode** ('**AutoCompleteMode Property**' in the **on-line documentation**) and **AutoCompleteCustomSource** ('**AutoCompleteCustomSource Property**' in the **on-line documentation**) properties. You can also specify maximum limits for the cell with the **MaxLength** ('**MaxLength Property**' in the **on-line documentation**) property.

You can use the **ShortcutKeys** ('**ShortcutKeys Property**' in the **on-line documentation**) property to map keys to actions for the GcDateTime and GcTextBox cells. In edit mode, these shortcut keys have precedence over the Spread input maps. The cell uses the Spread input maps when not in edit mode.

The **FormatString** ('**FormatString Property**' in the **on-line documentation**) property allows you to specify specific characters that are allowed in the cell. The following Spread Designer table displays the available characters.

Pattern	Description
<input type="checkbox"/> A	Upper case alphabet (A~Z)
<input type="checkbox"/> A	DBCS upper case alphabet (A~Z)
<input type="checkbox"/> a	Lower case alphabet (a~z)
<input type="checkbox"/> a	DBCS lower case alphabet (a~z)
<input type="checkbox"/> K	Katakana
<input type="checkbox"/> K	DBCS katakana
<input type="checkbox"/> 9	Numbers
<input type="checkbox"/> 9	DBCS numbers
<input type="checkbox"/> #	Numbers and number related symbols (0~9, + - \$ % \ .)
<input type="checkbox"/> #	DBCS numbers and number related symbols (0~9, + - \$ % \ .)
<input type="checkbox"/> @	Symbols
<input type="checkbox"/> @	DBCS symbols
<input type="checkbox"/> B	Binary numbers
<input type="checkbox"/> B	DBCS binary numbers
<input type="checkbox"/> X	Hexadecimal (0~9, A~F, a-f)
<input type="checkbox"/> X	DBCS Hexadecimal (0~9, A~F, a-f)
<input type="checkbox"/> J	Hiragana
<input type="checkbox"/> Z	All DBCS characters
<input type="checkbox"/> H	All SBCS characters
<input type="checkbox"/> D	All DBCS characters except for surrogates
<input type="checkbox"/> M	All Shift-JIS characters
<input type="checkbox"/> I	All JIS X 0208 characters
<input type="checkbox"/> N	Only large katakana
<input type="checkbox"/> N	Only DBCS large katakana
<input type="checkbox"/> G	Only large hiragana
<input type="checkbox"/> T	Surrogate char
<input type="checkbox"/> ^	Not include char

For a complete list of properties, see the **GcTextBoxCellType** ('GcTextBoxCellType Class' in the on-line **documentation**) class.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet in which the cells appear.
5. In the property list, select the **Cells** property and then click the button to display the **Cell, Column, and Row Editor**.
6. Select the cells for which you want to set the cell type.
7. In the property list, select the **CellType** property and choose the **GcTextBox** cell type.
8. Expand the list of properties under the **CellType** property. Select and set these specific properties as needed.
9. Click **OK** to close the **Cell, Column, and Row Editor**.
10. Click **OK** to close the **SheetView Collection Editor**.

Using Code

1. Define the text cell by creating an instance of the **GcTextBoxCellType** ('GcTextBoxCellType Class' in the **on-line documentation**) class.
2. Set properties for the class.
3. Assign the cell type to a cell or range of cells by setting the **CellType** property for a cell, column, row, or style to the **GcTextBoxCellType** object.

Example

This example creates a GcTextBox cell and cuts CrLf characters in copied, cut, or pasted strings.

C#

```
GrapeCity.Win.Spread.InputMan.CellType.GcTextBoxCellType inputcell1 = new
GrapeCity.Win.Spread.InputMan.CellType.GcTextBoxCellType();
inputcell1.Multiline = true;
inputcell1.AcceptsCrLf = GrapeCity.Win.Spread.InputMan.CellType.CrLfMode.Cut;
fpSpread1.Sheets[0].Cells[1, 1].CellType = inputcell1;
```

VB

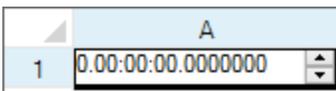
```
Dim inputcell1 As New GrapeCity.Win.Spread.InputMan.CellType.GcTextBoxCellType
inputcell1.Multiline = True
inputcell1.AcceptsCrLf = GrapeCity.Win.Spread.InputMan.CellType.CrLfMode.Cut
fpSpread1.Sheets(0).Cells(1, 1).CellType = inputcell1
```

Using the Spread Designer

1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the **Text** cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Select and set those properties as needed. Or right-click on the cell or cells and select **Cell Type**. From the list, select **GcTextBox**. In the **CellType** editor, set the properties you need. Click **Apply**.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting a GcTimeSpan Cell

You can use the GcTimeSpan cell to display time values in a cell.



You can specify literals and a pattern for the characters the user is allowed to enter.

The following table lists the characters you can use to create a pattern:

Keyword	Description
d	Represents the day field.
h	Represents the hour field.
m	Represents the minute field.
s	Represents the second field.

You can use the following classes with the GcTimeSpan cell to create display fields (literals) and input fields:

- **TimeSpanFieldCollectionInfo** ('TimeSpanFieldCollectionInfo Class' in the on-line documentation)
- **TimeSpanDisplayFieldCollectionInfo** ('TimeSpanDisplayFieldCollectionInfo Class' in the on-line documentation)

For a complete list of properties and methods for the GcTimeSpan cell, refer to the **GcTimeSpanCellType** ('GcTimeSpanCellType Class' in the on-line documentation) class.

Using Code

1. Define the cell by creating an instance of the **GcTimeSpanCellType** ('GcTimeSpanCellType Class' in the on-line documentation) class.
2. Set the pattern or literal characters.
3. Set properties for the class.
4. Assign the cell type to a cell or range of cells by setting the **CellType** property for a cell, column, row, or style to the **GcTimeSpan** object.

Example

This example creates a GcTimeSpan cell.

C#

```
GrapeCity.Win.Spread.InputMan.CellType.GcTimeSpanCellType GC = new
GrapeCity.Win.Spread.InputMan.CellType.GcTimeSpanCellType();
GC.AcceptsArrowKeys = FarPoint.Win.SuperEdit.AcceptsArrowKeys.AllArrows;
GC.EditMode = GrapeCity.Win.Spread.InputMan.CellType.EditMode.Overwrite;
GC.ExitOnLastChar = true;

GC.Fields.Clear();
GC.Fields.AddRange("d.hh:mm:ss,7,,,,-,");
GC.DisplayFields.Clear();
GC.DisplayFields.AddRange("d.hh:mm:ss,7,,,,-,");
GC.DefaultActiveField = GC.Fields[1];

GC.FocusPosition =
GrapeCity.Win.Spread.InputMan.CellType.FieldsEditorFocusCursorPosition.SelectAll;
GC.MaxMinBehavior = GrapeCity.Win.Spread.InputMan.CellType.MaxMinBehavior.Clear;
GC.NegativeColor = Color.Chocolate;
GC.PaintByControl = true;
GC.ShowRecommendedValue = true;

GC.SideButtons.Add(new GrapeCity.Win.Spread.InputMan.CellType.SpinButtonInfo());
GC.Spin.AllowSpin = true;
GC.Spin.Increment = 1;
GC.Spin.SpinOnKeys = true;
GC.Spin.SpinOnWheel = true;
GC.Spin.Wrap = true;

GC.UseNegativeColor = true;
GC.ValidateMode = GrapeCity.Win.Spread.InputMan.CellType.ValidateMode.ValidateNone;
GC.ValueSign = GrapeCity.Win.Spread.InputMan.CellType.ValueSignControl.Positive;
GC.UseSpreadDropDownButtonRender = true;

fpSpread1.ActiveSheet.Cells[0, 0].CellType = GC;
```

VB

```
Dim GC As New GrapeCity.Win.Spread.InputMan.CellType.GcTimeSpanCellType()
GC.AcceptsArrowKeys = FarPoint.Win.SuperEdit.AcceptsArrowKeys.AllArrows
GC.EditMode = GrapeCity.Win.Spread.InputMan.CellType.EditMode.Overwrite
GC.ExitOnLastChar = True

GC.Fields.Clear()
GC.Fields.AddRange("d.hh:mm:ss,7,,,,-")
GC.DisplayFields.Clear()
GC.DisplayFields.AddRange("d.hh:mm:ss,7,,,,-")
GC.DefaultActiveField = GC.Fields(1)

GC.FocusPosition =
GrapeCity.Win.Spread.InputMan.CellType.FieldsEditorFocusCursorPosition.SelectAll
GC.MaxMinBehavior = GrapeCity.Win.Spread.InputMan.CellType.MaxMinBehavior.Clear
GC.NegativeColor = Color.Chocolate
GC.PaintByControl = True
GC.ShowRecommendedValue = True

GC.SideButtons.Add(New GrapeCity.Win.Spread.InputMan.CellType.SpinButtonInfo())
GC.Spin.AllowSpin = True
GC.Spin.Increment = 1
GC.Spin.SpinOnKeys = True
GC.Spin.SpinOnWheel = True
GC.Spin.Wrap = True

GC.UseNegativeColor = True
GC.ValidateMode = GrapeCity.Win.Spread.InputMan.CellType.ValidateMode.ValidateNone
GC.ValueSign = GrapeCity.Win.Spread.InputMan.CellType.ValueSignControl.Positive
GC.UseSpreadDropDownButtonRender = True

fpSpread1.ActiveSheet.Cells(0, 0).CellType = GC
```

Using the Spread Designer

1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the **GcTimeSpan** cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Select and set those properties as needed.
Or right-click on the cell or cells and select **Cell Type**. From the list, select **GcTimeSpan**. In the **CellType** editor, set the properties you need. Click **Apply**.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting a Number Cell

You can use a number cell for entering double-precision floating point numbers as well as fractions. You can display decimal numbers, integers, or fractions. The topics below discuss the various aspects of number cell formatting and calculation.

You use the **NumberCellType** (**'NumberCellType Class' in the on-line documentation**) class to set the number cell and its properties. Use the **CurrencyCellType** (**'CurrencyCellType Class' in the on-line documentation**) class to set the currency cell and its properties.

Setting Precision

Numbers are typically calculated and stored using the Double data type which provides an accuracy of about 15 digits. The cell can be formatted to display as many or as few digits as you want. For example, the following code would sum the values in the cell range A1:A5 and place the result in cell A6. The value stored in cell A6 would have full accuracy (up to the limits of the Double data type), but the text displayed in cell A6 would show the value rounded to the nearest tenths place (one decimal place).

C#

```
NumberCellType ct = new NumberCellType();
ct.DecimalPlaces = 1;
spread.Sheets[0].Cells[5,0].CellType = ct;
spread.Sheets[0].Cells[5,0].Formula = "SUM(A1:A5)";
```

Number cells supports 15 significant digits of precision. This is a total of all digits, integral and fractional. For example, when you have 10 fractional digits, you limit the number of integer digits to the left of the decimal to 5 digits. Also, there is the possibility of floating point errors with the Double data type. For more accurate precision of large numbers or numbers with large fractional portions, consider using a currency cell which uses the Decimal data type and is not prone to floating point errors.

Formatting Numbers

You can customize the number cell to display the number as an integer or decimal with several formatting features as summarized in this table of properties. An example of the use of these properties is provided after the table.

Property	Description
DecimalPlaces ('DecimalPlaces Property' in the on-line documentation)	Sets the number of decimal places in the display of the number, for a decimal number.
DecimalSeparator ('DecimalSeparator Property' in the on-line documentation)	Sets the decimal character for the display of a decimal number.
FixedPoint ('FixedPoint Property' in the on-line documentation)	Sets whether to display zeros as placeholders in the decimal portion of the number for a fixed-point numeric display.
LeadingZero ('LeadingZero Property' in the on-line documentation)	Sets whether leading zeros are displayed.
MaximumValue ('MaximumValue Property' in the on-line documentation)	Sets the maximum value allowed for user input.
MinimumValue ('MinimumValue Property' in the on-line documentation)	Sets the minimum value allowed for user input.
NegativeFormat ('NegativeFormat Property' in the on-line documentation)	Sets how the value is formatted for negative values.
NegativeRed ('NegativeRed Property' in the on-line documentation)	Sets whether negative numeric values are displayed in red.
OverflowCharacter ('OverflowCharacter Property' in the on-line documentation)	Sets the character to use to replace the value if it does not fit the width of the display.
Separator ('Separator Property' in the on-line documentation)	Sets the string used to separate thousands in a numeric value.
ShowSeparator ('ShowSeparator Property' in the on-line documentation)	Sets whether to display the thousands separator string.

A complete list of formatting properties can be found in the **NumberCellType** ('NumberCellType Class' in the on-

line documentation) class. You can use code, the Properties Window, or the Spread Designer to set these properties.

Displaying Fractions

The number cell can display values in a fraction format, so 0.01 can be displayed as 1/100. Set the **FractionMode** (**'FractionMode Property' in the on-line documentation**) property of the number cell to display values in the fraction format. You can type values in the cell as 0.01 or you can type 1/100 in the cell; both display as 1/100. The precision of the fraction can be set using the **FractionDenominatorPrecision** (**'FractionDenominatorPrecision Enumeration' in the on-line documentation**) enumeration (such as to display fractions as quarters, 1/4, etc.) or the **FractionDenominatorDigits** (**'FractionDenominatorDigits Property' in the on-line documentation**) to set the number of digits in the denominator, for 10s, 100s or 1000s or more. This table lists the fraction-related properties of the number cell.

Property	Description
FractionMode ('FractionMode Property' in the on-line documentation)	Sets whether values are represented as fractions.
FractionConvertWholeNumbers ('FractionConvertWholeNumbers Property' in the on-line documentation)	Sets whether to convert whole numbers to fractions when values are displayed as fractions.
FractionCustomFormat ('FractionCustomFormat Property' in the on-line documentation)	Sets how values are displayed as fractions with custom formatting. To use the custom format, set the FractionDenominatorPrecision ('FractionDenominatorPrecision Property' in the on-line documentation) property to Custom .
FractionDenominatorDigits ('FractionDenominatorDigits Property' in the on-line documentation)	Sets the number of digits when values are displayed as fractions.
FractionDenominatorPrecision ('FractionDenominatorPrecision Property' in the on-line documentation)	Sets the precision when values are displayed as fractions.
FractionRenderOnly ('FractionRenderOnly Property' in the on-line documentation)	Sets whether to allow fractions in edit mode when values are displayed as fractions.

Another way to set the fraction display is to set a value for the fraction custom format (using the **FractionCustomFormat** (**'FractionCustomFormat Property' in the on-line documentation**) property). The default value is "# ???/???" which formats the number as an integer (#) followed by a three-digit fraction (???/???). The question marks after the slash determine the number of digits of denominator precision of which there can be from one to fifteen (because 15-digit precision is the maximum). With the custom format, you can also specify the denominator, such as "# ???/100" or "# ??/64". If **FractionConvertWholeNumbers** (**'FractionConvertWholeNumbers Property' in the on-line documentation**) is set to true, then there is no integer to display and the entire number is displayed as a fraction.

The alignment of the display is determined by the alignment properties that are set for the cell. The number is not aligned based on the fraction display. (In the example below, the numbers are right aligned regardless of whether there is a fractional part or not.)

A complete list of fraction properties can be found in the **NumberCellType** (**'NumberCellType Class' in the on-line documentation**) class. You can use code, the Properties Window, or the Spread Designer to set these properties.

Using Spin Buttons

By default, no spin buttons are shown, but you can display spin buttons on the side of the cell when the cell is in edit mode. You can set various spin functions using the properties of the **NumberCellType** ('**NumberCellType Class**' in the on-line documentation) class that begin with the word Spin. Refer to **Displaying Spin Buttons**.

Using the Pop-Up Calculator

By default, in a number cell, if you double-click on the cell in edit mode at run-time, a pop-up calculator appears. You can specify the text that displays in the **OK** and **Cancel** buttons. For more information, refer to **Customizing the Pop-Up Calculator Control**. To prohibit the popping up of the calculator, cancel the FpSpread **SubEditorOpening** ('**SubEditorOpening Event**' in the on-line documentation) event. Handle this event and set the *Cancel* argument of the **SubEditorOpeningEventArgs** ('**SubEditorOpeningEventArgs Class**' in the on-line documentation) to True.

For more information on the properties and methods of the number cell type, refer to the **NumberCellType** ('**NumberCellType Class**' in the on-line documentation) class.

For more information on the currency cell type, refer to the **Setting a Currency Cell**.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet in which the cells appear.
5. In the property list, select the **Cells** property and then click the button to display the **Cell, Column, and Row Editor**.
6. Select the cells for which you want to set the cell type.
7. In the property list, select the **CellType** property and choose the **Number** cell type.
8. Expand the list of properties under the **CellType** property. Select and set these specific properties as needed.
9. Click **OK** to close the **Cell, Column, and Row Editor**.
10. Click **OK** to close the **SheetView Collection Editor**.

Using Code for Formatting Numbers

1. Define the number cell by creating an instance of the **NumberCellType** ('**NumberCellType Class**' in the on-line documentation) class.
2. Set properties for the class.
3. Assign the number cell type to a cell or range of cells by setting the **CellType** property for a cell, column, row, or style to the **NumberCellType** ('**NumberCellType Class**' in the on-line documentation) object.

Example

This example sets a cell to be a numeric cell with certain formatting by assigning the **NumberCellType** ('**NumberCellType Class**' in the on-line documentation) object with defined formatting properties.

C#

```
FarPoint.Win.Spread.CellType.NumberCellType nmbrcell = new
FarPoint.Win.Spread.CellType.NumberCellType();
nmbrcell.DecimalSeparator = ",";
nmbrcell.DecimalPlaces = 5;
nmbrcell.LeadingZero = FarPoint.Win.Spread.CellType.LeadingZero.UseRegional;
nmbrcell.MaximumValue = 500.000;
nmbrcell.MinimumValue = -10.000;
```

```
fpSpread1.ActiveSheet.Cells[1, 1].CellType = nmbrcell;
```

VB

```
Dim nmbrcell As New FarPoint.Win.Spread.CellType.NumberCellType()  
nmbrcell.DecimalSeparator = ","  
nmbrcell.DecimalPlaces = 5  
nmbrcell.LeadingZero = FarPoint.Win.Spread.CellType.LeadingZero.UseRegional  
nmbrcell.MaximumValue = 500.000  
nmbrcell.MinimumValue = -10.000  
fpSpread1.ActiveSheet.Cells(1, 1).CellType = nmbrcell
```

Using Code for Formatting Fractions

1. Define the number cell by creating an instance of the **NumberCellType** ('**NumberCellType Class**' in the **on-line documentation**) class.
2. Set the **FractionMode** property to true and other fraction properties as needed.
3. Assign the number cell type to a cell or range of cells by setting the **CellType** property for a cell, column, row, or style to the **NumberCellType** ('**NumberCellType Class**' in the **on-line documentation**) object.

Example

This example sets a cell to display numbers as fractions.

C#

```
fpSpread1.ActiveSheet.Columns[0, 9].Width = 120;  
FarPoint.Win.Spread.CellType.NumberCellType frac = new  
FarPoint.Win.Spread.CellType.NumberCellType();  
frac.FractionMode = true;  
frac.FractionConvertWholeNumbers = false;  
frac.FractionDenominatorPrecision =  
FarPoint.Win.Spread.CellType.FractionDenominatorPrecision.Custom;  
frac.FractionCustomFormat = "## ???/???"  
frac.FractionDenominatorDigits = 3;  
fpSpread1.ActiveSheet.Columns[0].CellType = frac;  
fpSpread1.ActiveSheet.Columns[1].CellType = frac;  
fpSpread1.ActiveSheet.Cells[0, 0].Value = 5.00;  
fpSpread1.ActiveSheet.Cells[1, 0].Value = 5.01;  
fpSpread1.ActiveSheet.Cells[2, 0].Value = 5.02;  
fpSpread1.ActiveSheet.Cells[3, 0].Value = 5.03;  
fpSpread1.ActiveSheet.Cells[4, 0].Value = 5.04;  
fpSpread1.ActiveSheet.Cells[5, 0].Value = 5.05;  
fpSpread1.ActiveSheet.Cells[6, 0].Value = 5.06;  
fpSpread1.ActiveSheet.Cells[7, 0].Value = 5.07;  
fpSpread1.ActiveSheet.Cells[8, 0].Value = 5.08;  
fpSpread1.ActiveSheet.Cells[9, 0].Value = 5.09;  
fpSpread1.ActiveSheet.Cells[0, 1].Value = 25.000;  
fpSpread1.ActiveSheet.Cells[1, 1].Value = 25.011;  
fpSpread1.ActiveSheet.Cells[2, 1].Value = 25.021;  
fpSpread1.ActiveSheet.Cells[3, 1].Value = 25.031;  
fpSpread1.ActiveSheet.Cells[4, 1].Value = 25.041;  
fpSpread1.ActiveSheet.Cells[5, 1].Value = 25.051;  
fpSpread1.ActiveSheet.Cells[6, 1].Value = 25.061;  
fpSpread1.ActiveSheet.Cells[7, 1].Value = 25.071;  
fpSpread1.ActiveSheet.Cells[8, 1].Value = 25.081;
```

```
fpSpread1.ActiveSheet.Cells[9, 1].Value = 25.091;
```

VB

```
FpSpread1.ActiveSheet.Columns(0, 9).Width = 120
Dim frac As New FarPoint.Win.Spread.CellType.NumberCellType
frac.FractionMode = True
frac.FractionConvertWholeNumbers = False
frac.FractionDenominatorPrecision =
FarPoint.Win.Spread.CellType.FractionDenominatorPrecision.Custom
frac.FractionCustomFormat = "# ???/???"
frac.FractionDenominatorDigits = 3
FpSpread1.ActiveSheet.Columns(0).CellType = frac
FpSpread1.ActiveSheet.Columns(1).CellType = frac
FpSpread1.ActiveSheet.Cells(0, 0).CellType = frac
FpSpread1.ActiveSheet.Cells(0, 0).Value = 5.00
FpSpread1.ActiveSheet.Cells(1, 0).Value = 5.01
FpSpread1.ActiveSheet.Cells(2, 0).Value = 5.02
FpSpread1.ActiveSheet.Cells(3, 0).Value = 5.03
FpSpread1.ActiveSheet.Cells(4, 0).Value = 5.04
FpSpread1.ActiveSheet.Cells(5, 0).Value = 5.05
FpSpread1.ActiveSheet.Cells(6, 0).Value = 5.06
FpSpread1.ActiveSheet.Cells(7, 0).Value = 5.07
FpSpread1.ActiveSheet.Cells(8, 0).Value = 5.08
FpSpread1.ActiveSheet.Cells(9, 0).Value = 5.09
FpSpread1.ActiveSheet.Cells(0, 1).Value = 25.000
FpSpread1.ActiveSheet.Cells(1, 1).Value = 25.011
FpSpread1.ActiveSheet.Cells(2, 1).Value = 25.021
FpSpread1.ActiveSheet.Cells(3, 1).Value = 25.031
FpSpread1.ActiveSheet.Cells(4, 1).Value = 25.041
FpSpread1.ActiveSheet.Cells(5, 1).Value = 25.051
FpSpread1.ActiveSheet.Cells(6, 1).Value = 25.061
FpSpread1.ActiveSheet.Cells(7, 1).Value = 25.071
FpSpread1.ActiveSheet.Cells(8, 1).Value = 25.081
FpSpread1.ActiveSheet.Cells(9, 1).Value = 25.091
```

This is what the result looks like in Spread.

	A	B
1	5	25
2	5 1/100	25 1/91
3	5 1/50	25 13/619
4	5 3/100	25 27/871
5	5 1/25	25 18/439
6	5 1/20	25 23/451
7	5 3/50	25 33/541
8	5 7/100	25 12/169
9	5 2/25	25 26/321
10	5 9/100	25 90/989
11		

Using the Spread Designer

1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the **Number** cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Select and set those properties as needed. The fraction properties are under the fraction tab. Or right-click on the cell or cells and select **Cell Type**. From the list, select **Number**. In the **CellType** editor, set the properties you need. Click **Apply**.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting a Currency Cell

You can set a cell to display currency values using the currency cell. A currency cell displays the numeric currency values with formatting that you can customize including a currency symbol, a separator character, and other formatting.

CAS15,664.00
(CAS334.00)
CAS1,247.89

You use the **CurrencyCellType** ('**CurrencyCellType Class**' in the on-line documentation) class to set the currency cell and its properties.

By default, Spread uses the regional Windows settings (or options) of the machine on which it runs for the formatting of currency. You can customize any of these currency formatting properties:

- currency symbol (and whether to display it)
- separator character (and whether to display it)
- decimal symbol
- whether to display a leading zero
- positive value indicator (and whether to display it)
- negative value indicator (and whether to display it)

By default, in a currency cell, if you double-click on the cell in edit mode at run-time, a pop-up calculator appears. You can determine whether to allow this, and you can specify the text that displays on the **OK** and **Cancel** buttons. For more information, refer to **Customizing the Pop-Up Calculator Control**.

You can also set the minimum and maximum values that can be entered to provide validation of the user entry. To define the limits for values, refer to **Limiting Values for a Numeric Cell**.

Using Spin Buttons

By default, no spin buttons are shown, but you can display spin buttons on the side of the cell when the cell is in edit mode. You can set various spin functions using the properties of the **CurrencyCellType** ('**CurrencyCellType Class**' in the on-line documentation) that begin with the word Spin. For more information, refer to **Displaying Spin Buttons**.

For more information on the properties and methods of this cell type, refer to the **CurrencyCellType** ('**CurrencyCellType Class**' in the on-line documentation) class.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet in which the cells appear.
5. In the property list, select the **Cells** property and then click the button to display the **Cell, Column, and Row**

Editor.

6. Select the cells for which you want to set the cell type.
7. In the property list, select the **CellType** property and choose the **Currency** cell type.
8. Expand the list of properties under the **CellType** property. Select and set these specific properties as needed.
9. Click **OK** to close the **Cell, Column, and Row Editor**.
10. Click **OK** to close the **SheetView Collection Editor**.

Using Code

1. Define a currency cell by creating an instance of the **CurrencyCellType** ('**CurrencyCellType Class**' in the **on-line documentation**) class.
2. Specify the formatting of a currency cell by setting the **CurrencySymbol** ('**CurrencySymbol Property**' in the **on-line documentation**) and other properties for the **CurrencyCellType** ('**CurrencyCellType Class**' in the **on-line documentation**) object.
3. Assign the currency cell type to a cell or range of cells by setting the **CellType** property for a cell, column, row, or style to the **CurrencyCellType** ('**CurrencyCellType Class**' in the **on-line documentation**) object.

Example

This example creates a currency cell.

C#

```
FarPoint.Win.Spread.CellType.CurrencyCellType currcell = new
FarPoint.Win.Spread.CellType.CurrencyCellType();
currcell.CurrencySymbol = "US$";
currcell.DecimalSeparator = ":";
currcell.DecimalPlaces = 8;
fpSpread1.ActiveSheet.Cells[1,1].CellType = currcell;
```

VB

```
Dim currcell As New FarPoint.Win.Spread.CellType.CurrencyCellType()
currcell.CurrencySymbol = "US$"
currcell.DecimalSeparator = ":"
currcell.DecimalPlaces = 8
fpSpread1.ActiveSheet.Cells(1,1).CellType = currcell
```

Using the Spread Designer

1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the **Currency** cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Select and set those properties as needed.
Or right-click on the cell or cells and select **Cell Type**. From the list, select **Currency**. In the **CellType** editor, set the properties you need. Click **Apply**.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting a Mask Cell

You can use a mask cell for masking characters to limit user entry. You specify which subsets of characters are allowed for each item in the mask. You can define how the mask appears, with literals displayed exactly as typed and placeholders showing the places for user entry. To create a mask, set the **Mask** ('**Mask Property**' in the **on-line**

documentation) property to a string of mask characters. Each mask character represents a position in which the user can type a character.



You use the **MaskCellType** ('**MaskCellType Class**' in the on-line documentation) class to set the mask cell and its properties.

For a detailed list of the mask characters, refer to the **Mask** ('**Mask Property**' in the on-line documentation) property in the **MaskCellType** ('**MaskCellType Class**' in the on-line documentation) class. For a description of how to set the placeholder character, refer to the **MaskChar** ('**MaskChar Property**' in the on-line documentation) property.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet in which the cells appear.
5. In the property list, select the **Cells** property and then click the button to display the **Cell, Column, and Row Editor**.
6. Select the cells for which you want to set the cell type.
7. In the property list, select the **CellType** property and choose the **Mask** cell type.
8. Expand the list of properties under the **CellType** property. Select and set these specific properties as needed.
9. Click **OK** to close the **Cell, Column, and Row Editor**.
10. Click **OK** to close the **SheetView Collection Editor**.

Using Code

1. Define the mask cell by creating an instance of the **MaskCellType** ('**MaskCellType Class**' in the on-line documentation) class.
2. Define the mask, including literals and placeholders by setting the **Mask** ('**Mask Property**' in the on-line documentation) property.
3. Assign the mask cell type to a cell or range of cells by setting the **CellType** property for a cell, column, row, or style to the **MaskCellType** ('**MaskCellType Class**' in the on-line documentation) object.

Example

This example sets a cell to be a mask cell and restricts the user to entering two alphabetic names and prompts them with X's. The display looks like this:

```
-> XXXXXXXX : XXXXXXXX <-
```

C#

```
FarPoint.Win.Spread.CellType.MaskCellType maskcell = new
FarPoint.Win.Spread.CellType.MaskCellType();
maskcell.Mask = "-> ULLLLLLL : ULLLLLLL <-";
maskcell.MaskChar = Convert.ToChar("X");
fpSpread1.ActiveSheet.Cells[1, 1].CellType = maskcell;
```

VB

```
Dim maskcell As New FarPoint.Win.Spread.CellType.MaskCellType()
```

```
maskcell.Mask = "-> ULLLLLLL : ULLLLLLL <-"  
maskcell.MaskChar = "X"  
fpSpread1.ActiveSheet.Cells(1, 1).CellType = maskcell
```

Using the Spread Designer

1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the **Mask** cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Select and set those properties as needed.
Or right-click on the cell or cells and select **Cell Type**. From the list, select **Mask**. In the **CellType** editor, set the properties you need. Click **Apply**.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting a Percent Cell

You can use a percent cell for displaying values as percentages and restricting inputs to percentage numeric values.



You use the **PercentCellType** (**'PercentCellType Class' in the on-line documentation**) class to set the percent cell and its properties.

Using Spin Buttons

By default, no spin buttons are shown, but you can display spin buttons on the side of the cell when the cell is in edit mode. You can set various spin functions using the properties of the **PercentCellType** (**'PercentCellType Class' in the on-line documentation**) class that begin with the word Spin. Refer to **Displaying Spin Buttons**.

Using the Calculator

By default, in a percent cell, if you double-click on the cell in edit mode at run-time, a pop-up calculator appears. You can determine whether to allow this, and you can specify the text that displays in the **OK** and **Cancel** buttons. For more information, refer to **Customizing the Pop-Up Calculator Control**. To prohibit the popping up of the calculator, cancel the FpSpread **SubEditorOpening** (**'SubEditorOpening Event' in the on-line documentation**) event. Handle this event and set the *Cancel* argument of the **SubEditorOpeningEventArgs** (**'SubEditorOpeningEventArgs Class' in the on-line documentation**) to True.

For more information on the properties and methods of this cell type, refer to the **PercentCellType** (**'PercentCellType Class' in the on-line documentation**) class.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet in which the cells appear.
5. In the property list, select the **Cells** property and then click the button to display the **Cell, Column, and Row Editor**.
6. Select the cells for which you want to set the cell type.
7. In the property list, select the **CellType** property and choose the **Percent** cell type.
8. Expand the list of properties under the **CellType** property. Select and set these specific properties as needed.
9. Click **OK** to close the **Cell, Column, and Row Editor**.

10. Click **OK** to close the **SheetView Collection Editor**.

Using Code

1. Define the percent cell by creating an instance of the **PercentCellType** (**'PercentCellType Class' in the on-line documentation**) class.
2. Set properties for the class.
3. Assign the percent cell type to a cell or range of cells by setting the **CellType** property for a cell, column, row, or style to the **PercentCellType** (**'PercentCellType Class' in the on-line documentation**) object.

Example

This example sets a cell to be a percent cell and displays an abbreviation (PRCNT) instead of the percent sign (%).

C#

```
FarPoint.Win.Spread.CellType.PercentCellType prctcell = new
FarPoint.Win.Spread.CellType.PercentCellType();
prctcell.PercentSign = "PRCNT";
prctcell.PositiveFormat =
FarPoint.Win.Spread.CellType.PercentPositiveFormat.PercentBefore;
fpSpread1.ActiveSheet.Cells[1, 1].CellType = prctcell;
```

VB

```
Dim prctcell As New FarPoint.Win.Spread.CellType.PercentCellType()
prctcell.PercentSign = "PRCNT"
prctcell.PositiveFormat =
FarPoint.Win.Spread.CellType.PercentPositiveFormat.PercentBefore
fpSpread1.ActiveSheet.Cells(1, 1).CellType = prctcell
```

Using the Spread Designer

1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the **Percent** cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Select and set those properties as needed.
Or right-click on the cell or cells and select **Cell Type**. From the list, select **Percent**. In the **CellType** editor, set the properties you need. Click **Apply**.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting a Regular Expression Cell

You can create a regular expression cell that restricts the data entered in the cell to valid entries defined in a regular expression. The data is evaluated when exiting the cell. Invalid data is removed and the **EditError** (**'EditError Event' in the on-line documentation**) event is raised.

You use the **RegularExpressionCellType** (**'RegularExpressionCellType Class' in the on-line documentation**) class to set the regular expression cell and its properties.

For a summary of regular expression syntax, refer to the Regular Expression Syntax topic in the Microsoft .NET Framework Reference. For an introduction to regular expressions, refer to the Introduction to Regular Expressions topic in the Microsoft .NET Framework Reference.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet in which the cells appear.
5. In the property list, select the **Cells** property and then click the button to display the **Cell, Column, and Row Editor**.
6. Select the cells for which you want to set the cell type.
7. In the property list, select the **CellType** property and choose the **RegularExpression** cell type.
8. Expand the list of properties under the **CellType** property. Select and set these specific properties as needed.
9. Click **OK** to close the **Cell, Column, and Row Editor**.
10. Click **OK** to close the **SheetView Collection Editor**.

Using Code

1. Define the regular expression cell by creating an instance of the **RegularExpressionCellType** (**'RegularExpressionCellType Class' in the on-line documentation**) class.
2. Create a regular expression.
3. Create a message to display to the user when the expression is not valid.
4. Assign the regular expression cell type to a cell or range of cells by setting the **CellType** property for a cell, column, row, or style to the **RegularExpressionCellType** (**'RegularExpressionCellType Class' in the on-line documentation**) object.

Example

This example creates a regular expression cell.

C#

```
FarPoint.Win.Spread.CellType.RegularExpressionCellType regexcell = new
FarPoint.Win.Spread.CellType.RegularExpressionCellType ()
regexcell.RegularExpression = "[0-9]{3}-[0-9]{2}-[0-9]{4}";
fpSpread1.ActiveSheet.Cells[0, 0].CellType = regexcell;
```

VB

```
Dim regexcell As New FarPoint.Win.Spread.CellType.RegularExpressionCellType ()
regexcell.RegularExpression = "[0-9]{3}-[0-9]{2}-[0-9]{4}"
fpSpread1.ActiveSheet.Cells(0, 0).CellType = regexcell
```

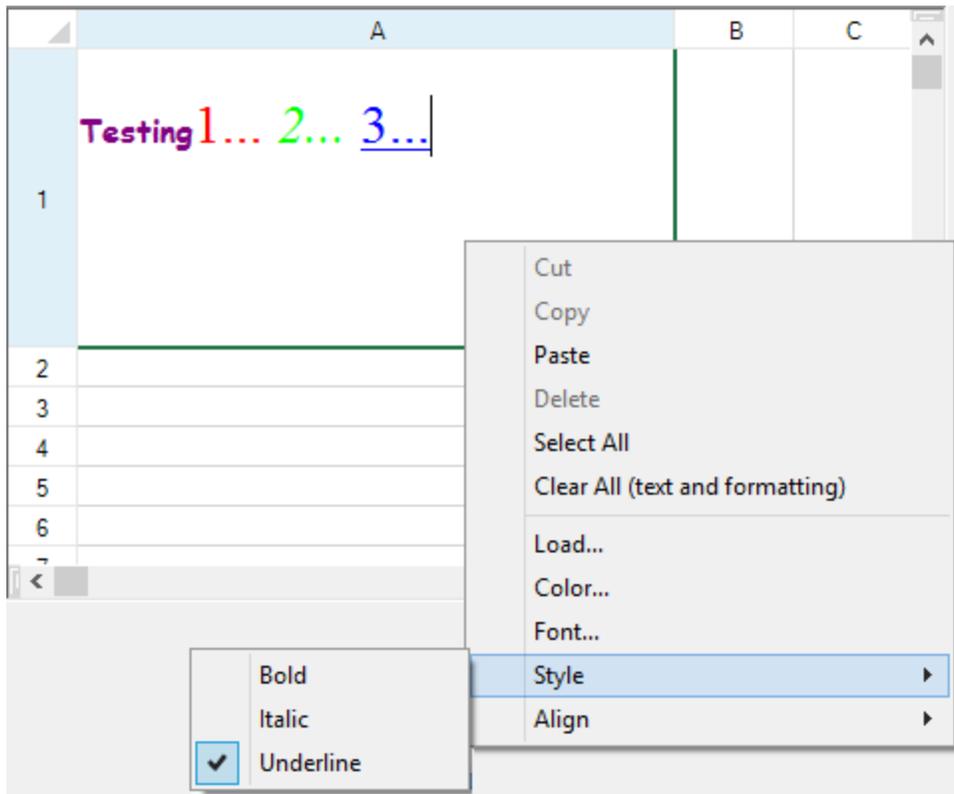
Using the Spread Designer

1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the **RegularExpression** cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Select and set those properties as needed.
Or right-click on the cell or cells and select **Cell Type**. From the list, select **RegularExpression**. In the **CellType** editor, set the properties you need. Click **Apply**.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting a Rich Text Cell

You can create a rich text cell that has text with multiple colors and fonts in the cell.

The rich text cell has a run-time menu for formatting the text in a cell and for loading a rich text formatted file (RTF). The menu also has basic edit operations such as cut, copy, paste, delete, and select all. To bring up the menu at run time the cell must be in Edit mode. Then you right-click in the cell to bring up the menu for handling the contents of that cell. Highlight the text first if you want to set the color, font, or style of the text in the cell. An example of the use of the menu is shown in this figure.



Understanding Context Menu Choices

The choices in the context menu are described in this table.

Menu Item	Description
Cut Copy Paste	Standard Clipboard operations that cut, copy, and paste to and from the Clipboard the contents (text and formatting) of the cell
Delete	Removes the selected contents (text and formatting) of the cell
Select All	Selects the entire contents (text and formatting) of the cell
Clear All	Clears the entire contents (text and formatting) of the cell
Load	Opens the File Open dialog to allow you to select an RTF file to load
Color	Opens a color palette to allow you to select a color that is either applied to selected text or is used as the color of subsequent text
Font	Opens a font picker to allow you to select a font face and size that is either applied to selected text or is used as the font of subsequent text
Style	Opens a menu to allow you to select a font style (bold, italics, or underline) that is either applied to

	selected text or is used as the font of subsequent text
Align	Opens a menu to allow you to set the text alignment

Using the Rich-Text Cell

When the rich-text cell is first edited or text is loaded, the RTF string of data and formatting information is set in the cell. Subsequent changes to the cell, column, or row appearance settings (such as font, text color, alignment) do not affect the existing text because the formatting information for that text is already set. For the newer settings to be applied to the original string, you must completely clear the cell of both data and formatting.

You can use the **Value ('Value Property' in the on-line documentation)** property in code to load the contents of an RTF file as a string of RTF commands, or you can set any set of RTF commands directly.

Setting a property such as alignment may not apply after the cell has been modified in edit mode. Many formatting properties (**HorizontalAlignment ('HorizontalAlignment Property' in the on-line documentation)**, **Font ('Font Property' in the on-line documentation)**, **ForeColor ('ForeColor Property' in the on-line documentation)**, and so on) can be set in the hidden RTF code associated with the value of the cell. For example, setting the **HorizontalAlignment** property prior to any editing may correctly set the alignment, but setting this property after editing might not have the desired effect. RTF has hidden formatting embedded in the RTF string which can be parsed or viewed through the **Value ('Value Property' in the on-line documentation)** property of the cell.

The cell width may be off several pixels if you automatically size the rich text cell and you are using multiple fonts.

Deleting the text from a rich-text cell puts a non-null string in the cell. This may cause unexpected results with the ISBLANK and COUNTBLANK formula functions since they treat the non-null string as non-blank.

If the **WordWrap ('WordWrap Property' in the on-line documentation)** property is true when using the **GetPreferredRowHeight ('GetPreferredRowHeight Method' in the on-line documentation)** method, the height may be slightly larger if the last character on a wrapped line is on the border.

Important Notes

The rich-text cell uses the Microsoft .NET **RichTextBox** class for editing. Text using a font with extended character sets, like MSPMincho, cannot be set to a font that does not support extended character sets, like Tahoma. The result is that **RichTextBox** does not allow you to change the font of a Far East script-based font to a Western font.

This font issue is a limitation of the **RichTextBox** for the Microsoft Windows 2000 operating system. If you try to set a font that does not support extended character sets, like Tahoma, on text with extended characters, a message box is displayed.

For more information on the properties and methods of this cell type, refer to the **RichTextCellType ('RichTextCellType Class' in the on-line documentation)** class.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet in which the cells appear.
5. In the property list, select the **Cells** property and then click the button to display the **Cell, Column, and Row Editor**.
6. Select the cells for which you want to set the cell type.
7. In the property list, select the **CellType** property and choose the **RichText** cell type.
8. Expand the list of properties under the **CellType** property. Select and set these specific properties as needed.
9. Click **OK** to close the **Cell, Column, and Row Editor**.
10. Click **OK** to close the **SheetView Collection Editor**.

Using Code

1. Define a rich text cell by creating an instance of the **RichTextCellType ('RichTextCellType Class' in the on-line documentation)** class.
2. Specify the properties of the **RichTextCellType ('RichTextCellType Class' in the on-line documentation)** class.
3. If you want to load RTF data, set the **Value** property of the **RichTextCellType ('RichTextCellType Class' in the on-line documentation)** class to load the data
4. Assign the rich text cell type to a cell or range of cells by setting the **CellType ('CellType Property' in the on-line documentation)** property for a cell, column, row, or style to the **RichTextCellType ('RichTextCellType Class' in the on-line documentation)** object.

Example

This example creates a rich text cell and loads data.

C#

```
FarPoint.Win.Spread.CellType.RichTextCellType rtf = new
FarPoint.Win.Spread.CellType.RichTextCellType();
rtf.WordWrap = true;
rtf.Multiline = true;
fpSpread1.ActiveSheet.Cells[0, 0].CellType = rtf;
fpSpread1.ActiveSheet.Columns[0].Width = 300;
fpSpread1.ActiveSheet.Rows[0].Height = 150;
fpSpread1.ActiveSheet.Cells[0, 0].Value = @"{\rtf1\ansi\ansicpg1252\deff0\deflang1033
{\fonttbl{\f0\fscript\fprq2\fcharset0 Comic Sans MS;}
{\f1\froman\fprq2\fcharset0 Times New Roman;}
{\f2\fswiss\fcharset0 Arial;}}{\colortbl ;\red128
\green0\blue128;\red0\green255\blue255;\red255\green0\
blue0;\red0\green255\blue0;\red0\green0\blue255;}
\viewkind4\uc1\pard\cf1\b\f0\fs24 Testing\cf2\b0\fs28 \cf3
\f1\fs40 1... \cf4\i 2... \cf5\ul\i0 3...\cf0\f2\fs20\par}";
```

VB

```
Dim rtf As New FarPoint.Win.Spread.CellType.RichTextCellType()
rtf.WordWrap = True
rtf.Multiline = True
fpSpread1.ActiveSheet.Cells(0, 0).CellType = rtf
fpSpread1.ActiveSheet.Columns(0).Width = 300
fpSpread1.ActiveSheet.Rows(0).Height = 150
fpSpread1.ActiveSheet.Cells(0, 0).Value = "{\rtf1\ansi\ansicpg1252\deff0\deflang1033" +
_
"\fonttbl{\f0\fscript\fprq2\fcharset0 Comic Sans MS;}" + _
_
"{\f1\froman\fprq2\fcharset0 Times New Roman;}" + _
_
"{\f2\fswiss\fcharset0 Arial;}}}" + _
_
"{\colortbl ;\red128\green0\blue128;\red0\green255\blue255;" + _
_
"\red255\green0\blue0;" + _
_
"\red0\green255\blue0;\red0\green0\blue255;}" + _
_
"\viewkind4\uc1\pard\cf1\b\f0\fs24 Testing\cf2\b0\fs28" + _
_
" \cf3\f1\fs40 1... \cf4\i 2... \cf5\ul\i0 3...\cf0\f2\fs20\par" + _
_
"}"
```

Example

This example loads a rich text file.

C#

```
System.IO.TextReader f = System.IO.File.OpenText("your_file.rtf");
string bits;
bits = f.ReadToEnd();
f.Close();
fpSpread1.ActiveSheet.Cells[0, 0].CellType = new
FarPoint.Win.Spread.CellType.RichTextCellType();
fpSpread1.ActiveSheet.Cells[0, 0].Value = bits;
```

VB

```
Dim f as System.IO.TextReader = System.IO.File.OpenText("your_file.rtf")
Dim bits As String
bits = f.ReadToEnd()
f.Close()
fpSpread1.ActiveSheet.Cells(0, 0).CellType = New
FarPoint.Win.Spread.CellType.RichTextCellType()
fpSpread1.ActiveSheet.Cells(0, 0).Value = bits
```

Using the Spread Designer

1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the **RichTextCellType** cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Select and set those properties as needed.
Or right-click on the cell or cells and select **Cell Type**. From the list, select **RichText**. In the **CellType** editor, set the properties you need. Click **Apply**.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Working with Graphical Cell Types

You can work with the graphical cell types as described in the following topics:

- **Setting a Button Cell**
- **Setting a Check Box Cell**
- **Setting a Combo Box Cell**
- **Setting a Hyperlink Cell**
- **Setting an Image Cell**
- **Setting a List Box Cell**
- **Setting a Multiple-Column Combo Box Cell**
- **Setting a Multiple Option Cell**
- **Setting a Progress Indicator Cell**
- **Setting a Slider Cell**

The graphical cell types use a graphic or a control or form. They are based on the **BaseCellType ('BaseCellType Class' in the on-line documentation)** class and you can define a subeditor for them.

For other cell types, refer to **Working with Editable Cell Types**.

Note that some graphical elements in certain cell types are affected by XP themes (visual styles). Setting the **VisualStyles** property of the Spread component to "off" can allow visual customizations of those graphical cell types to work as expected. For more information, refer to **Using XP Themes with the Component**.

Setting a Barcode Cell

You can display a barcode graphic in a barcode cell. Various barcode types are available such as barcodes that are used in retail, for shipping, and so on. You can set height and width properties for the barcode display.



You use the **BarcodeCellType** (**'BarcodeCellType Class' in the on-line documentation**) class to set the barcode cell and its properties.

Customizing the Appearance

You can customize the barcode cell by using these properties:

Property	Description
AcceptsCheckDigit ('AcceptsCheckDigit Property' in the on-line documentation)	Sets whether to accept the check digit in the input.
AdjustSize ('AdjustSize Property' in the on-line documentation)	Sets whether the barcode adjusts its size based on the barcode size.
AutoStretch ('AutoStretch Property' in the on-line documentation)	Sets whether the size is based on the cell size.
BarAdjust ('BarAdjust Property' in the on-line documentation)	Sets whether to fine tune the width of the barcode.
BarcodePadding ('BarcodePadding Property' in the on-line documentation)	Sets the left side and right side padding of the barcode.
BarSize ('BarSize Property' in the on-line documentation)	Sets the height and width of the barcode.
DisplayCheckDigit ('DisplayCheckDigit Property' in the on-line documentation)	Sets whether the check digit is available for the barcode.
DisplayMode ('DisplayMode Property' in the on-line documentation)	Sets whether the barcode draws a barcode image.
FixedLength ('FixedLength Property' in the on-line documentation)	Sets the number of the fixed digits of the value of the barcode.
IsFormulaValue ('IsFormulaValue Property' in the on-line documentation)	Determines whether the editor contains a formula.
Message ('Message Property' in the on-line documentation)	Sets whether to display the custom message string below the barcode image (if the barcode type allows it).
MessagePosition ('MessagePosition Property' in the on-line documentation)	Sets the alignment of the custom message below the barcode image.
MessageValue ('MessageValue Property' in the on-line documentation)	Sets a custom message to display below the barcode image.

MinimumHeight ('MinimumHeight Property' in the on-line documentation)	Sets the minimum height of entire barcode.
ModuleSize ('ModuleSize Property' in the on-line documentation)	Sets the size of the barcode module.
Resolution ('Resolution Property' in the on-line documentation)	Sets the resolution of the barcode.
Rotation ('Rotation Property' in the on-line documentation)	Sets the rotation angle of the barcode.
Type ('Type Property' in the on-line documentation)	Sets the bar type of the barcode.
Unit ('Unit Property' in the on-line documentation)	Sets the unit of measure of the barcode.

The **FixedLength ('FixedLength Property' in the on-line documentation)** property only works with PostNet, ITF, or Code39 barcode types. The PostNet barcode option allows one less digit for the value than the setting for the **FixedLength** property.

Only the bar code types that have a line at the bottom to display the value can display a message (as set by the **Message ('Message Property' in the on-line documentation)**, **MessagePosition ('MessagePosition Property' in the on-line documentation)**, and **MessageValue ('MessageValue Property' in the on-line documentation)** properties).

Barcode Types

Here is a sample image of each of the supported barcode types that can be set with the **Type ('Type Property' in the on-line documentation)** property. The Jan8 type is similar to the EAN8 type. For more information about barcode types, refer to this article: <https://www.gs1.org/standards/barcodes>.

Barcode Type

Sample Barcode Image

Code128



Code39



Code49



Code93



EAN128



ITF



Jan13



Jan8



JapanesePostal



NW7



PDF417



PostNet



QRCode



UPC



Customizing the Message

You can display a customizable text message at the bottom of the barcode cell for several barcode types. This is supported for those barcode types that display the number value below the barcode image. (For example, PostNet, PDF417, JapanesePostal, and QRCode do not display a message.) In the example here, the value has been replaced with a text message. The code that generated this barcode image is shown below. The message is set using these properties: **Message** ('Message Property' in the on-line documentation), **MessagePosition** ('MessagePosition Property' in the on-line documentation), and **MessageValue** ('MessageValue Property' in the on-line documentation).

	A	B
1		
2		

C#

```
FarPoint.Win.Spread.CellType.BarCodeCellType barc = new
FarPoint.Win.Spread.CellType.BarCodeCellType();
barc.DisplayMode = FarPoint.Win.Spread.CellType.BarCodeDisplayMode.Image;
barc.Message = true;
barc.MessagePosition = FarPoint.Win.Spread.CellType.BarCode.MessagePosition.Left;
barc.MessageValue = "Display This Instead of Value";
barc.Type = new FarPoint.Win.Spread.CellType.BarCode.UPC();
fpSpread1.ActiveSheet.Columns[0].Width = 220;
fpSpread1.ActiveSheet.Rows[0].Height = 100;
fpSpread1.ActiveSheet.Cells[0, 0].CellType = barc;
fpSpread1.ActiveSheet.Cells[0, 0].Value = 36000280753;
```

VB

```
Dim barc As New FarPoint.Win.Spread.CellType.BarCodeCellType
```

```
barc.DisplayMode = FarPoint.Win.Spread.CellType.BarCodeDisplayMode.Image
barc.Message = True
barc.MessagePosition = FarPoint.Win.Spread.CellType.BarCode.MessagePosition.Left
barc.MessageValue = "Display This Instead of Value"
barc.Type = New FarPoint.Win.Spread.CellType.BarCode.UPC
fpSpread1.ActiveSheet.Columns(0).Width = 220
fpSpread1.ActiveSheet.Rows(0).Height = 100
fpSpread1.ActiveSheet.Cells(0, 0).CellType = barc
fpSpread1.ActiveSheet.Cells(0, 0).Value = 36000280753
```

Note that some graphical elements in certain cell types are affected by XP themes (visual styles). Setting the **VisualStyles** property of the Spread component to "off" can allow visual customizations of those graphical cell types to work as expected. For more information, refer to **Using XP Themes with the Component**.

For more information on the properties and methods of this cell type, refer to the **BarCodeCellType** (**'BarCodeCellType Class' in the on-line documentation**) class.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet in which the cells appear.
5. In the property list, select the **Cells** property and then click the button to display the **Cell, Column, and Row Editor**.
6. Select the cells for which you want to set the cell type.
7. In the property list, select the **CellType** property and choose the **BarCode** cell type.
8. Expand the list of properties under the **CellType** property. Select and set these specific properties as needed.
9. Click **OK** to close the **Cell, Column, and Row Editor**.
10. Click **OK** to close the **SheetView Collection Editor**.

Using Code

1. Define the barcode cell by creating an instance of the **BarCodeCellType** (**'BarCodeCellType Class' in the on-line documentation**) class.
2. Specify the properties of the barcode cell.
3. Assign the barcode cell to a cell or range of cells by setting the **CellType** property for a cell, column, row, or style to the **BarCodeCellType** (**'BarCodeCellType Class' in the on-line documentation**) object.

Example

This example creates a bar code cell.

C#

```
FarPoint.Win.Spread.CellType.BarCodeCellType brcdcell = new
FarPoint.Win.Spread.CellType.BarCodeCellType();
FpSpread1.Sheets[0].Cells[0, 0].CellType = brcdcell;
FpSpread1.Sheets[0].Cells[0, 0].Value = "12345";
```

VB

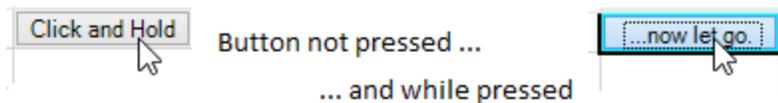
```
Dim brcdcell As New FarPoint.Win.Spread.CellType.BarCodeCellType
FpSpread1.Sheets(0).Cells(0, 0).CellType = brcdcell
FpSpread1.Sheets(0).Cells(0, 0).Value = "12345"
```

Using the Spread Designer

1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the BarCode cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Select and set those properties as needed.
Or right-click on the cell or cells and select **Cell Type**. From the list, select **Barcode**. In the **CellType** editor, set the properties you need. Click **Apply**.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting a Button Cell

You can display a button in a cell using the button cell. A button cell, by default displays a rectangular button with a default color; you can customize the text, color, and an image for that button as well as specify certain aspects of its behavior when clicked.



To create a cell that acts like a button, use the **ButtonCellType** ('**ButtonCellType Class**' in the **on-line documentation**) class and the settings that are summarized here. You can create a button cell using the examples shown in this topic or in the topics for the individual members of the **ButtonCellType** ('**ButtonCellType Class**' in the **on-line documentation**) class.

Customizing the Button Appearance

Button cells can display text, pictures, or both. If they display pictures, you can choose that a different picture is displayed when the button is pressed. You can customize the colors in button cells, including the color of the border, text, and background. In addition, button cells can display a three-dimensional appearance, and you can customize the colors of the highlight and shadow in the appearance. The following properties relate to the overall appearance of the button cell.

Property	Description
BackgroundStyle (' BackgroundStyle Property ' in the on-line documentation)	Sets how the background is rendered.
ButtonColor (' ButtonColor Property ' in the on-line documentation)	Sets the color of the button.
ButtonColor2 (' ButtonColor2 Property ' in the on-line documentation)	Sets the secondary color used when drawing a gradient button.
DarkColor (' DarkColor Property ' in the on-line documentation)	Sets the color at the bottom and right edges of the button (that give it the three-dimensional appearance along with the light color).
GradientMode (' GradientMode Property ' in the on-line documentation)	Sets the drawing style of a gradient button.

LightColor ('LightColor Property' in the on-line documentation)

Sets the color at the top and left edges of the button (that give it the three-dimensional appearance along with the dark color).

Picture ('Picture Property' in the on-line documentation)

Sets an image that fills the surface of the button. Any GDI+ bitmap can be used, such as a BMP, GIF, or JPG file. If you are using a two-state button, this serves as the unpressed state.

PictureDown ('PictureDown Property' in the on-line documentation)

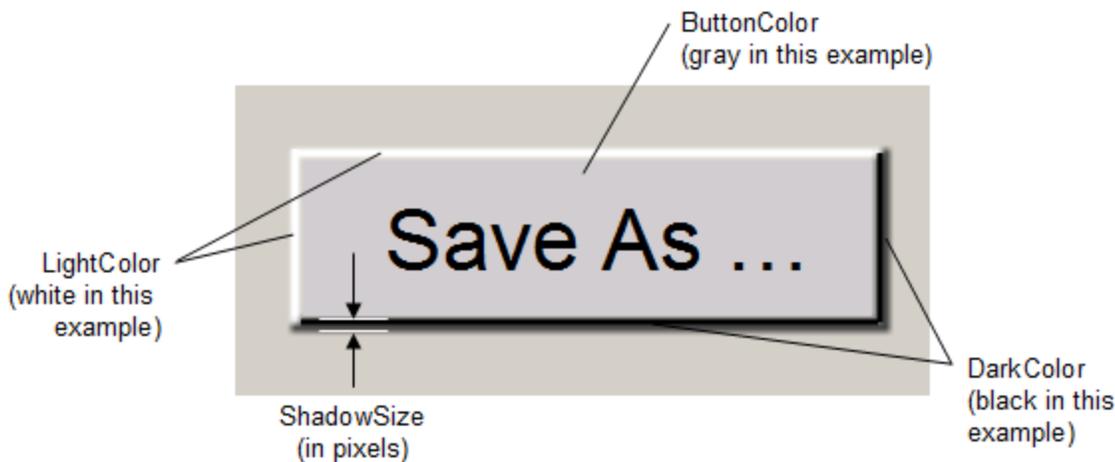
Sets an image for the pressed state of the button.

ShadowSize ('ShadowSize Property' in the on-line documentation)

Sets the thickness of the shadow, and the dark and light colors (that give it the three-dimensional appearance).

TwoState ('TwoState Property' in the on-line documentation)

Sets whether the button functions as a toggle switch with two states. Each time you click the button, the button changes state.



By default, the button has a single state, and changes its appearance only as long as you have the pointer over it with the mouse button pressed. For this setting, button cells behave like push buttons, which you can press by pressing your left mouse button, and which do not stay pressed when you release your mouse button. Alternatively, you can set the button to be a two-state button and then the button toggles between those two states when clicked. The button is clicked when the user clicks anywhere in that cell. The button stays pressed when you click it using your left mouse button. Buttons are False when they are not pressed, and True when they are pressed.

Customizing the Text Appearance

You can specify the text that is displayed in the button cell and you can specify the appearance of that text. You can specify the alignment of text alongside pictures in button cells as well as whether to wrap text to multiple lines.

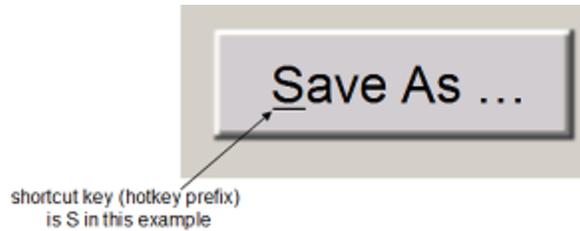
The following properties relate to the text that is displayed in the button cell.

Property

Description

HotkeyPrefix ('HotkeyPrefix Property' in the on-line documentation)

Sets whether to display the underline that indicates the access key (keyboard shortcut or hot key).



Text ('Text Property' in the on-line documentation)

Sets the text that appears in the button.

TextAlign ('TextAlign Property' in the on-line documentation)

Sets the alignment of the text with respect to a picture

TextColor ('TextColor Property' in the on-line documentation)

Sets the color of the text in the button.

TextDown ('TextDown Property' in the on-line documentation)

Sets the text of the button when it is pressed, if it is a two-state button.

TextOrientation ('TextOrientation Property' in the on-line documentation)

Sets the orientation of the text in the button. See the following table that shows examples of the various orientations.

WordWrap ('WordWrap Property' in the on-line documentation)

Sets whether to wrap the text to multiple lines.

Here are the results of different settings of the **TextOrientation ('TextOrientation Property' in the on-line documentation)** property.

Text Orientation

Example Button

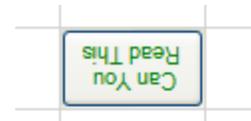
Text Orientation

Example Button

Text Horizontal



Text Horizontal Flipped



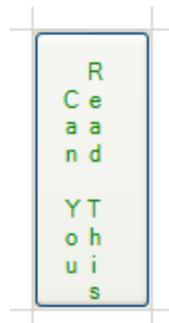
Text Vertical



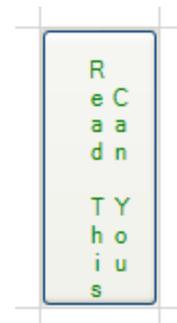
Text Vertical Flipped



Text TopDown



Text TopDown RTL



Text RotateCustom



Beyond the properties of the button cell itself, you can also set a property in the **FpSpread** (**'FpSpread Class' in the on-line documentation**) class that affects how buttons behave. The **FpSpread** (**'FpSpread Class' in the on-line documentation**) class has a **ButtonDrawMode** (**'ButtonDrawMode Property' in the on-line documentation**) property for button cells and combo box cells. This property allows you to always show a button, or show buttons in the current column, row, or cell.

Note that some graphical elements in certain cell types are affected by XP themes (visual styles). Setting the **VisualStyles** (**'VisualStyles Property' in the on-line documentation**) property of the Spread component to "off" can allow visual customizations of those graphical cell types to work as expected. For more information, refer to **Using XP Themes with the Component**.

For more information on the properties and methods of this cell type, refer to the **ButtonCellType** (**'ButtonCellType Class' in the on-line documentation**) class.

For more information on the corresponding event when a user clicks on the button, refer to the **FpSpread.ButtonClicked** (**'ButtonClicked Event' in the on-line documentation**) event.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet in which the cells appear.
5. In the property list, select the **Cells** property and then click the button to display the **Cell, Column, and Row Editor**.
6. Select the cells for which you want to set the cell type.
7. In the property list, select the **CellType** property and choose the **Button** cell type.
8. Expand the list of properties under the **CellType** property. Select and set these specific properties as needed.
9. Click **OK** to close the **Cell, Column, and Row Editor**.
10. Click **OK** to close the **SheetView Collection Editor**.

Using Code

1. Define the button cell by creating an instance of the **ButtonCellType** (**'ButtonCellType Class' in the on-line documentation**) class.
2. Specify the properties of the button by setting the properties of that instance, such as **Text**, **TwoState**, and **ButtonColor**.
3. Assign the button cell type to a cell or range of cells by setting the **CellType** property for a cell, column, row, or style to the **ButtonCellType** (**'ButtonCellType Class' in the on-line documentation**) object.

Example

This example creates a button with text in a blue colored button. It defines the text to be different when the mouse pointer is held down. This example creates the button shown in the first part of this topic.

C#

```
FarPoint.Win.Spread.CellType.ButtonCellType bttnCell = new
```

```

FarPoint.Win.Spread.CellType.ButtonCellType();
bttnCell.ButtonColor = Color.Cyan;
bttnCell.DarkColor = Color.DarkCyan;
bttnCell.LightColor = Color.AliceBlue;
bttnCell.TwoState = false;
bttnCell.Text = "Click and Hold";
bttnCell.TextDown = "...now let go.";
bttnCell.ShadowSize = 3;
fpSpread1.Sheets[0].Cells[0,2].CellType = bttnCell;
fpSpread1.Sheets[0].SetColumnWidth(2,90);

```

VB

```

Dim bttnCell As New FarPoint.Win.Spread.CellType.ButtonCellType()
bttnCell.ButtonColor = Color.Cyan
bttnCell.DarkColor = Color.DarkCyan
bttnCell.LightColor = Color.AliceBlue
bttnCell.TwoState = False
bttnCell.Text = "Click and Hold"
bttnCell.TextDown = "...now let go."
bttnCell.ShadowSize = 3
fpSpread1.Sheets(0).Cells(0,2).CellType = bttnCell
fpSpread1.Sheets(0).SetColumnWidth(2,90)

```

Using the Spread Designer

1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the Button cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Select and set those properties as needed.
Or right-click on the cell or cells and select **Cell Type**. From the list, select **Button**. In the **CellType** editor, set the properties you need. Click **Apply**.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting a Check Box Cell

You can display a check box in a cell using the check box cell. A check box cell can display a small check box that can have one of three states (checked, unchecked, or grayed) or two states (checked or unchecked). You can customize the check box by setting the text, determining the operation of the check box, and setting pictures in place of the standard check box pictures.

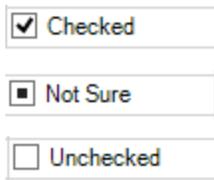
To create a cell that acts like a check box, use the **CheckBoxCellType** (**'CheckBoxCellType Class' in the on-line documentation**) class. Create a check box cell using the procedure and example shown below.

Customizing Text

You can customize the check box by specifying the image for each of the states. By default, the check box has only two states, checked or unchecked, so to use all three you must set the **ThreeState** (**'ThreeState Property' in the on-line documentation**) property. In the following table, default appearances are shown with text defined using the **TextTrue** (**'TextTrue Property' in the on-line documentation**), **TextFalse** (**'TextFalse Property' in the on-line documentation**), and **TextIndeterminate** (**'TextIndeterminate Property' in the on-line documentation**) properties. Clicking anywhere in the cell changes the check box state.

State Appearance

Description



True (checked)
 Indeterminate (grayed)
 False (unchecked)

You can customize the check box cell with these properties:

Property	Description
Caption ('Caption Property' in the on-line documentation)	Sets the text in the check box regardless of the state, overriding TextTrue ('TextTrue Property' in the on-line documentation) , TextFalse ('TextFalse Property' in the on-line documentation) , and TextIndeterminate ('TextIndeterminate Property' in the on-line documentation) text settings.
HotkeyPrefix ('HotkeyPrefix Property' in the on-line documentation)	Sets whether the ampersand character underlines text and creates an access key.
TextAlign ('TextAlign Property' in the on-line documentation)	Sets how the text is aligned in the cell with respect to the check box graphic.
TextFalse ('TextFalse Property' in the on-line documentation)	Sets the text for the false state of the check box.
TextIndeterminate ('TextIndeterminate Property' in the on-line documentation)	Sets the text for the indeterminate state of the check box.
TextTrue ('TextTrue Property' in the on-line documentation)	Sets the text for the true state of the check box.

Customizing Pictures

For each state, you can also set custom pictures for each state of the check box cell (making it appear more like a button). You can determine the appearance of the check box according to whether the cell has focus (normal), does not have focus (disabled), or is being clicked (pressed).

Property	Description
BackgroundImage ('BackgroundImage Property' in the on-line documentation)	Sets the background image for the cell.
Picture ('Picture Property' in the on-line documentation)	Sets the images to use for the states of the check box.
ThreeState ('ThreeState Property' in the on-line documentation)	Sets whether the check box has three states

Note that some graphical elements in certain cell types are affected by XP themes (visual styles). Setting the **VisualStyles ('VisualStyles Property' in the on-line documentation)** property of the Spread component to "off" can allow visual customizations of those graphical cell types to work as expected. For more information, refer to **Using XP Themes with the Component**.

For more information on the properties and methods of this cell type, refer to the **CheckBoxCellType**

('CheckBoxCellType Class' in the on-line documentation) class.

For more information on the corresponding event when a user clicks on the check box, refer to the `FpSpread.ButtonClicked` ('ButtonClicked Event' in the on-line documentation) event.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet in which the cells appear.
5. In the property list, select the **Cells** property and then click the button to display the **Cell, Column, and Row Editor**.
6. Select the cells for which you want to set the cell type.
7. In the property list, select the **CellType** property and choose the **CheckBox** cell type.
8. Expand the list of properties under the **CellType** property. Select and set these specific properties as needed.
9. Click **OK** to close the **Cell, Column, and Row Editor**.
10. Click **OK** to close the **SheetView Collection Editor**.

Using Code

1. Define the check box cell by creating an instance of the **CheckBoxCellType** ('CheckBoxCellType Class' in the on-line documentation) class.
2. Specify the properties of the check box cell, such as setting the check box to show three states using the **ThreeState** ('ThreeState Property' in the on-line documentation) property for that **CheckBoxCellType** ('CheckBoxCellType Class' in the on-line documentation) object.
3. Enter the text that goes along with each check box, using the **TextTrue** ('TextTrue Property' in the on-line documentation), **TextFalse** ('TextFalse Property' in the on-line documentation), and **TextIndeterminate** ('TextIndeterminate Property' in the on-line documentation) properties.
4. Specify the location of the images for the checked and unchecked boxes if you do not want to use the defaults, using the **Picture** ('Picture Property' in the on-line documentation) property. (This is not done in the following example.)
5. Assign the check box cell type to a cell or range of cells by setting the **CellType** ('CellType Property' in the on-line documentation) property for a cell, column, row, or style to the **CheckBoxCellType** ('CheckBoxCellType Class' in the on-line documentation) object.

Example

This example creates a check box cell.

C#

```
FarPoint.Win.Spread.CellType.CheckBoxCellType chkboxcell = new
FarPoint.Win.Spread.CellType.CheckBoxCellType();
chkboxcell.ThreeState = true;
chkboxcell.TextTrue = "Checked!";
chkboxcell.TextFalse = "Check";
chkboxcell.TextIndeterminate = "Not Sure";
fpSpread1.ActiveSheet.Cells[0, 0].CellType = chkboxcell;
```

VB

```
Dim chkboxcell As New FarPoint.Win.Spread.CellType.CheckBoxCellType()
chkboxcell.ThreeState = true
chkboxcell.TextTrue = "Checked!"
```

```
checkboxcell.TextFalse = "Check"  
checkboxcell.TextIndeterminate = "Not Sure"  
fpSpread1.ActiveSheet.Cells(0, 0).CellType = checkboxcell
```

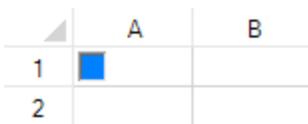
Using the Spread Designer

1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the **CheckBox** cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Select and set those properties as needed.
Or right-click on the cell or cells and select **Cell Type**. From the list, select **CheckBox**. In the **CellType** editor, set the properties you need. Click **Apply**.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting a Color Picker Cell

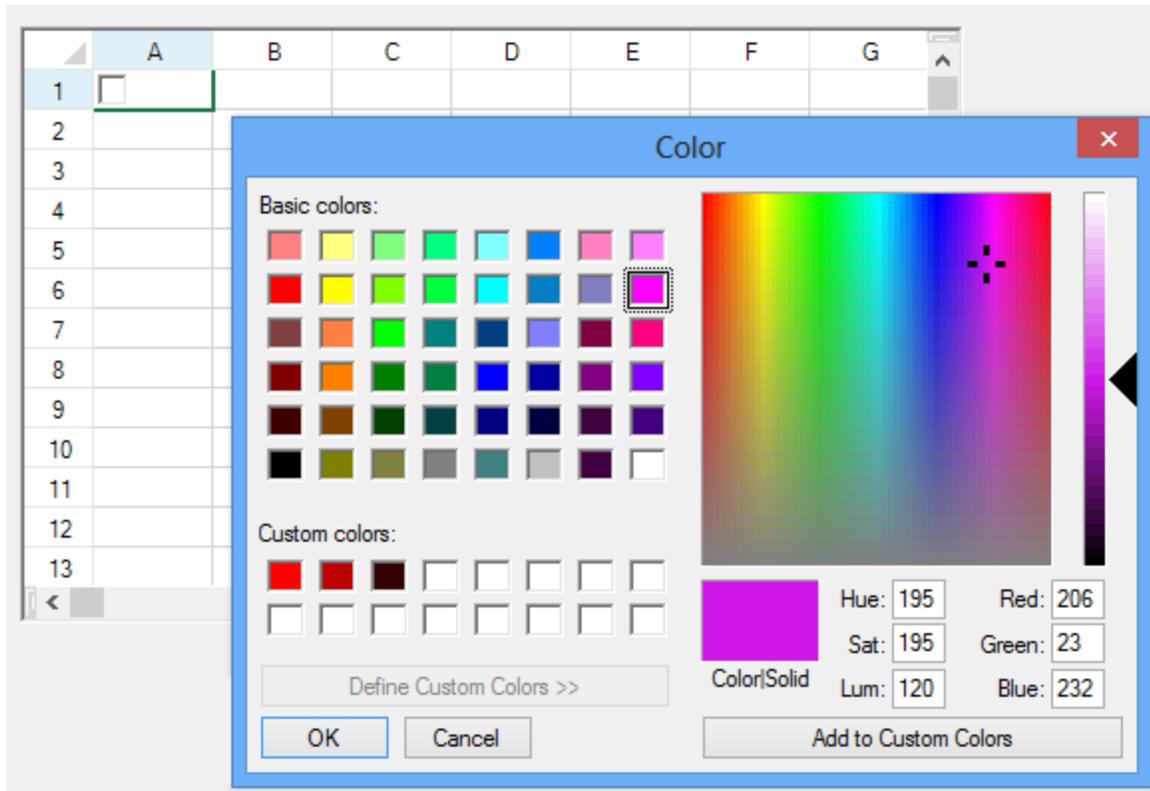
You can allow your end user to select a color from a color picker using the color picker cell. A color picker cell displays a dialog for selecting a color. There are several options for the color dialog.

When a color picker cell is selected it displays a single color, which can appear either in a box, as shown in the following image, or filling the entire area of the cell. Optionally text can be displayed.

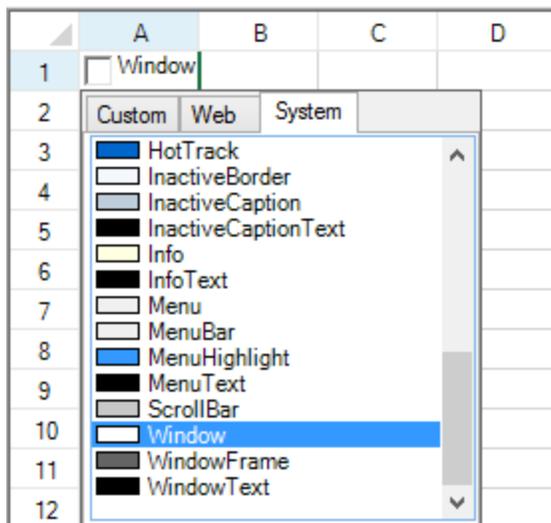


	A	B
1		
2		

When the cell is double-clicked, either the drop-down color picker is displayed or the pop-up color dialog is displayed. There are several options for the display of the color dialog. The following figure shows the pop-up color dialog:



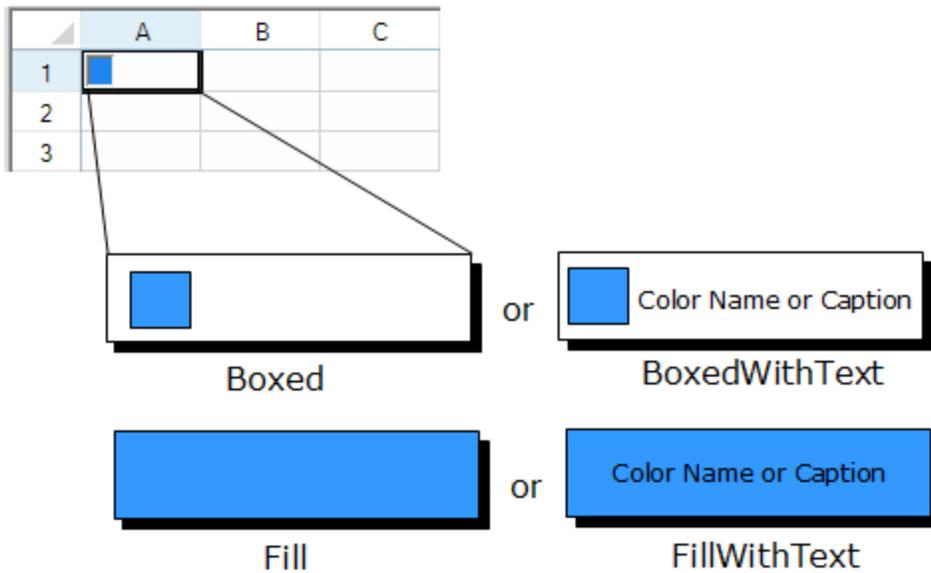
The following figure shows the drop-down color picker:



To create a color picker cell, use the **ColorPickerCellType** ('BarcodeCellType Class' in the on-line **documentation**) class. Create a color picker cell using the procedure and example shown below.

Customizing the Color Cell

The options for the color picker cell are in the **ColorPickerStyle** ('ColorPickerStyle Enumeration' in the on-line **documentation**) enumeration.



Customizing the Color Dialog

The color picker cell allows these customizations of the color dialog.

Property

AllowFullOpen ('AllowFullOpen Property' in the on-line documentation)

AnyColor ('AnyColor Property' in the on-line documentation)

Caption ('Caption Property' in the on-line documentation)

CustomColors ('CustomColors Property' in the on-line documentation)

DialogShowing ('DialogShowing Property' in the on-line documentation)

DropDown ('DropDown Property' in the on-line documentation)

FullOpen ('FullOpen Property' in the on-line documentation)

SolidColorOnly ('SolidColorOnly Property' in the on-line documentation)

Style ('Style Property' in the on-line documentation)

UnknownText ('UnknownText Property' in the on-line documentation)

UnknownTextStyle ('UnknownTextStyle Property' in the on-line documentation)

Description

Sets whether to allow the color dialog to open fully to show the custom color selector.

Sets whether the color dialog displays all available colors in the set of basic colors.

Sets the text that appears in the cell (if any).

Sets the custom colors shown in the color dialog.

Sets whether to display the color dialog automatically.

Sets whether to use the drop-down color picker (not the pop-up color dialog).

Sets whether the color dialog opens fully to show controls used to create custom colors.

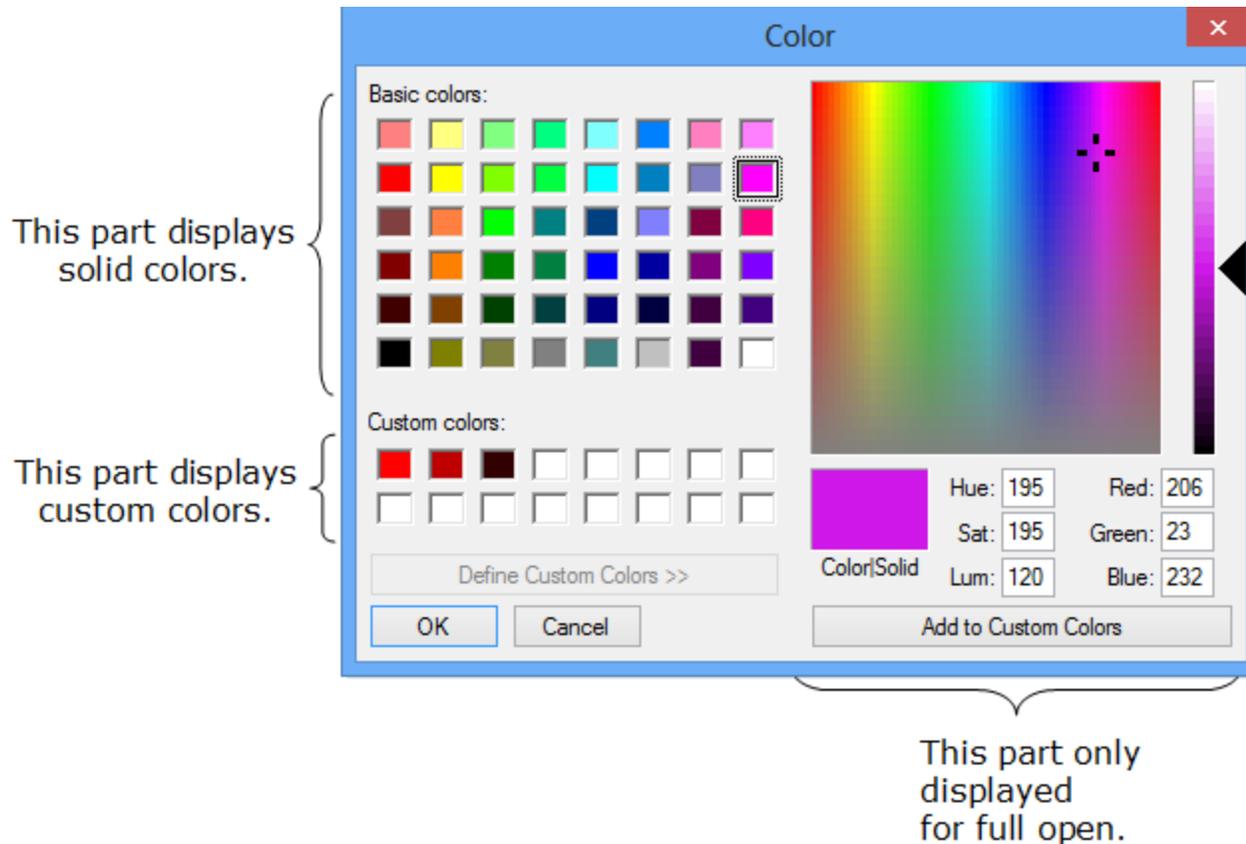
Sets whether the color dialog restricts users to selecting solid colors only.

Sets the style of the color dialog.

Sets the text for an unknown color.

Sets the style of the text for an unknown color.

The following figure illustrates the color dialog when it is set to fully open.



Note that some graphical elements in certain cell types are affected by XP themes (visual styles). Setting the **VisualStyles** ('**VisualStyles Property**' in the on-line documentation) property of the Spread component to "off" can allow visual customizations of those graphical cell types to work as expected. For more information, refer to **Using XP Themes with the Component**.

For more information on the properties and methods of this cell type, refer to the **ColorPickerCellType** ('**BarcodeCellType Class**' in the on-line documentation) class.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet in which the cells appear.
5. In the property list, select the **Cells** property and then click the button to display the **Cell, Column, and Row Editor**.
6. Select the cells for which you want to set the cell type.
7. In the property list, select the **CellType** property and choose the **ColorPicker** cell type.
8. Expand the list of properties under the **CellType** property. Select and set these specific properties as needed.
9. Click **OK** to close the **Cell, Column, and Row Editor**.
10. Click **OK** to close the **SheetView Collection Editor**.

Using Code

1. Define the color picker cell by creating an instance of the **ColorPickerCellType** ('**BarcodeCellType Class**' in the on-line documentation) class.

2. Specify the properties of the color picker cell.
3. Assign the color picker cell type to a cell or range of cells by setting the **CellType ('CellType Property' in the on-line documentation)** property for a cell, column, row, or style to the **ColorPickerCellType ('BarcodeCellType Class' in the on-line documentation)** object.

Example

This example creates a color picker cell.

C#

```
FarPoint.Win.Spread.CellType.ColorPickerCellType cp = new
FarPoint.Win.Spread.CellType.ColorPickerCellType();
cp.AllowFullOpen = true;
cp.AnyColor = false;
cp.CustomColors = new int[] {255, 190, 50};
cp.FullOpen = true;
cp.Style = FarPoint.Win.Spread.CellType.ColorPickerStyle.BoxedWithText;
FarPoint.Win.Spread.CellType.ColorPickerCellType c = new
FarPoint.Win.Spread.CellType.ColorPickerCellType(cp);
fpSpread1.ActiveSheet.Cells[0, 0].CellType = c;
```

VB

```
Dim cp As New FarPoint.Win.Spread.CellType.ColorPickerCellType
cp.AllowFullOpen = True
cp.AnyColor = False
cp.CustomColors = New Integer() {255, 190, 50}
cp.FullOpen = True
cp.Style = FarPoint.Win.Spread.CellType.ColorPickerStyle.BoxedWithText
Dim c As New FarPoint.Win.Spread.CellType.ColorPickerCellType(cp)
fpSpread1.ActiveSheet.Cells(0, 0).CellType = c
```

Using the Spread Designer

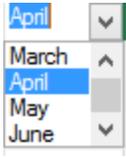
1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the **ColorPicker** cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Select and set those properties as needed.
Or right-click on the cell or cells and select **Cell Type**. From the list, select **ColorPicker**. In the **CellType** editor, set the properties you need. Click **Apply**.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting a Combo Box Cell

You can use a combo box cell to display an editable drop-down list, allowing the user to type in values as well as choosing from a displayed list. You can specify the list of items, whether to include icons to appear along with text, the number of items that are displayed at any time, and whether the cell is editable by the user.

Text only

Text and icon



To create a cell that acts like a combo box, use the **ComboBoxCellType** ('**ComboBoxCellType Class**' in the **on-line documentation**) class. Create a combo box cell using the following procedure.

Customizing the List Appearance

Use the following appearance properties to customize the combo box.

Property	Description
BackgroundImage (' BackgroundImage Property ' in the on-line documentation)	Sets an image to paint in the background of the edit portion of the combo box.
ButtonAlign (' ButtonAlign Property ' in the on-line documentation)	Sets where buttons are displayed.
ImageList (' ImageList Property ' in the on-line documentation)	Sets an image list for displaying icons along with text in the drop-down list in the combo box.
ItemData (' ItemData Property ' in the on-line documentation)	Sets item data, which is different from the items that are displayed, for the drop-down list in the combo box.
Items (' Items Property ' in the on-line documentation)	Sets items for the drop-down list in the combo box.
ListAlignment (' ListAlignment Property ' in the on-line documentation)	Sets the side of the cell on which the list aligns.
ListOffset (' ListOffset Property ' in the on-line documentation)	Sets how many pixels to offset the list from the aligned edge of the cell.
ListWidth (' ListWidth Property ' in the on-line documentation)	Sets the width (in pixels) of the drop-down list.
MaxDrop (' MaxDrop Property ' in the on-line documentation)	Sets the number of items to display at one time in the list portion. If there are more items than are displayed, a vertical scroll bar is displayed.
MaxLength (' MaxLength Property ' in the on-line documentation)	Sets the maximum number of characters allowed in the combo box cell.

Customizing the List Operation

Use the following operation properties to customize the combo box.

Property	Description
AcceptsArrowKeys (' AcceptsArrowKeys Property ' in the on-line documentation)	Sets how arrow keys are processed by the combo box control.
AutoSearch (' AutoSearch Property ' in the on-line documentation)	Sets how a list of items in a combo box is searched based on input of a character key.
CharacterCasing (' CharacterCasing Property ' in the on-line documentation)	Sets the case of characters in the text cell.

the on-line documentation)

CharacterSet ('**CharacterSet Property**' in the on-line documentation)

Sets what characters to allow for the text cell.

Editable ('**Editable Property**' in the on-line documentation)

Sets whether you can type into the edit portion of the combo box.

EditorValue ('**EditorValue Property**' in the on-line documentation)

Sets what value is written to the underlying data model.

ListControl ('**ListControl Property**' in the on-line documentation)

Sets the control to use for the list portion if you do not want to use the built-in list control in Spread.

The Spread control has a **ButtonDrawMode** ('**ButtonDrawMode Property**' in the on-line documentation) property for button cells and combo box cells. This property allows you to always show a button, or show buttons in the current column, row, or cell.

Customizing Automatic Completion

Use the following properties to customize the automatic completion feature in combo box cells when using the 2005 build of the component.

Property

AutoCompleteCustomSource ('**AutoCompleteCustomSource Property**' in the on-line documentation)

Description

Set the custom source (strings) for automatic completion of entries in the combo box.

AutoCompleteMode ('**AutoCompleteMode Property**' in the on-line documentation)

Set the mode for automatic completion of entries in the combo box.

AutoCompleteSource ('**AutoCompleteSource Property**' in the on-line documentation)

Set the source for automatic completion of entries in the combo box.

Note that some graphical elements in certain cell types are affected by XP themes (visual styles). Setting the **VisualStyles** ('**VisualStyles Property**' in the on-line documentation) property of the Spread component to "off" can allow visual customizations of those graphical cell types to work as expected. For more information, refer to **Using XP Themes with the Component**.

To display a text tip over a combo box cell, see the note in **Displaying Text Tips in a Cell**.

For more information on the properties and methods of this cell type, refer to the **ComboBoxCellType** ('**ComboBoxCellType Class**' in the on-line documentation) class. For information on the multiple-column combo box, refer to **Setting a Multiple-Column Combo Box Cell**.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet in which the cells appear.
5. In the property list, select the **Cells** property and then click the button to display the **Cell, Column, and Row Editor**.
6. Select the cells for which you want to set the cell type.
7. In the property list, select the **CellType** property and choose the **ComboBox** cell type.
8. Expand the list of properties under the **CellType** property. Select and set these specific properties as needed.
9. Click **OK** to close the **Cell, Column, and Row Editor**.
10. Click **OK** to close the **SheetView Collection Editor**.

Using Code

1. Define a combo box cell by creating an instance of the **ComboBoxCellType ('ComboBoxCellType Class' in the on-line documentation)** class.
2. Specify the items in the list that appear as part of the combo box. You can either use the **Items** property of the **ComboBoxCellType ('ComboBoxCellType Class' in the on-line documentation)** class or define a string and pass that in when creating the instance of the class.
3. Specify how the list of items appears. For example, set the **MaxDrop ('MaxDrop Property' in the on-line documentation)** property to set the maximum number of items to display at a time. If there are more items, a scroll bar appears. You can also set the horizontal alignment of the check box with respect to the cell.
4. Assign the combo box cell type to a cell or range of cells by setting the **CellType ('CellType Property' in the on-line documentation)** property for a cell, column, row, or style to the **ComboBoxCellType ('ComboBoxCellType Class' in the on-line documentation)** object.

Example

This example creates a combo cell.

C#

```
FarPoint.Win.Spread.CellType.ComboBoxCellType cmbocell = new
FarPoint.Win.Spread.CellType.ComboBoxCellType();
cmbocell.Items = (new String[] {"January", "February", "March", "April", "May",
"June"});
cmbocell.AutoSearch = FarPoint.Win.AutoSearch.SingleCharacter;
cmbocell.Editable = true;
cmbocell.MaxDrop = 4;
fpSpread1.ActiveSheet.Cells[0, 0].CellType = cmbocell;
```

VB

```
Dim cbstr As String( )
cbstr = New String( ) {"Jan", "Feb", "Mar", "Apr", "May", "Jun"}
Dim cmbocell As New FarPoint.Win.Spread.CellType.ComboBoxCellType()
cmbocell.Items = cbstr
cmbocell.AutoSearch = FarPoint.Win.AutoSearch.SingleCharacter
cmbocell.Editable = True
cmbocell.MaxDrop = 4
fpSpread1.ActiveSheet.Cells(0, 0).CellType = cmbocell
```

Using the Spread Designer

1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the **ComboBox** cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Select and set those properties as needed.
Or right-click on the cell or cells and select **Cell Type**. From the list, select **ComboBox**. In the **CellType** editor, set the properties you need. Click **Apply**.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Allowing a Combo Box Cell to Handle a Double Click

By default, a combo box cell (**ComboBoxCellType ('ComboBoxCellType Class' in the on-line**

documentation) cannot receive a double-click with the left mouse button. The cell goes into edit mode on the first click, so the next click goes to the FpCombo control that is the subeditor. To handle double-clicking on a combo box cell, use code based on the example shown here.

For more information on the properties and methods of this cell type, refer to the **ComboBoxCellType ('ComboBoxCellType Class' in the on-line documentation)** class.

For information on the graphical cell types, refer to **Working with Graphical Cell Types**.

For information on other features of cell types, refer to **Understanding Additional Features of Cell Types**.

Using Code

1. Define a combo box cell by creating an instance of the **ComboBoxCellType ('ComboBoxCellType Class' in the on-line documentation)** class.
2. Define the items in the combo box list.
3. Handle the double-click event.

Example

This example provides an event when double-clicking on a combo cell.

C#

```
private void Form1_Load(object sender, System.EventArgs e)
{
    FarPoint.Win.Spread.CellType.ComboBoxCellType c = new
FarPoint.Win.Spread.CellType.ComboBoxCellType();
    c.Items = new String[] {"a", "b", "c"};
    fpSpread1.Sheets[0].Rows[0].CellType = c;
}

private void HeaderDoubleClick(object sender, System.EventArgs e)
{
    //Add event code
}

private void fpSpread1_EditModeOn(object sender, System.EventArgs e)
{
    FarPoint.Win.FpCombo c;
    if (fpSpread1.Sheets[0].ActiveRowIndex == 0)
    {
        c = ((FarPoint.Win.FpCombo) (fpSpread1.EditingControl));
        c.Click += new System.EventHandler(HeaderDoubleClick);
    }
}
```

VB

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
MyBase.Load
    Dim c As New FarPoint.Win.Spread.CellType.ComboBoxCellType
    c.Items = New String() {"a", "b", "c"}
    fpSpread1.Sheets(0).Rows(0).CellType = c
End Sub

Private Sub HeaderDoubleClick(ByVal sender As Object, ByVal e As System.EventArgs)
'Add event code
End Sub
```

```

Private Sub fpSpread1_EditModeOn(ByVal sender As Object, ByVal e As System.EventArgs)
Handles fpSpread1.EditModeOn
    Dim c As FarPoint.Win.FpCombo
    If fpSpread1.Sheets(0).ActiveRowIndex = 0 Then
        c = CType(fpSpread1.EditingControl, FarPoint.Win.FpCombo)
        AddHandler c.Click, AddressOf HeaderDoubleClick
    End If
End Sub

```

Setting a Hyperlink Cell

You can use a hyperlink cell to contain text that functions as a single hyperlink or multiple hyperlinks. The destination of the hyperlink can be any universal resource locator (URL). For example:

- <https://developer.mescius.com/>
- <mailto:us.sales@mescius.com?Subject=Spread>

Customizing Links

You can specify how much of the text functions as a hyperlink and the rest displays as ordinary text. You can specify the appearance of the hyperlinked text, customize the color of the link that has been followed (visited or clicked) and whether to use the text for the hyperlink from the Data Model.

Property

BackgroundImage ('BackgroundImage Property' in the on-line documentation)

Link ('Link Property' in the on-line documentation)

LinkArea ('LinkArea Property' in the on-line documentation)

LinkAreas ('LinkAreas Property' in the on-line documentation)

LinkColor ('LinkColor Property' in the on-line documentation)

Links ('Links Property' in the on-line documentation)

Text ('Text Property' in the on-line documentation)

VisitedLinkColor ('VisitedLinkColor Property' in the on-line documentation)

UseModelValueAsText ('UseModelValueAsText Property' in the on-line documentation)

Customization

Sets the background graphic image.

Sets the destination URL.

Sets the area of the text that is the hyperlink.

Sets the area of the text that is the hyperlink.

Sets the color of links (before they are followed).

Sets the hyperlinks.

Sets the label of the hyperlink, that is, what appears in the cell.

Sets the color of followed links.

Gets a value indicating whether to use the text for the hyperlink from the Data Model.

Making Links in Text

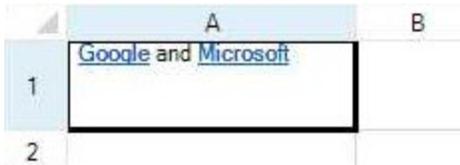
To create a cell that acts like a hyperlink, use the **HyperLinkCellType** ('HyperLinkCellType Class' in the on-line documentation) class. Create a hyperlink cell using the following procedure. The results of the procedure, both normal and followed links, are shown in the following figure.

Hyperlink Text Before Clicking Link

Hyperlink Text After Following Link



The following figure displays a hyperlink cell with multiple links.



For more information on the properties and methods of this cell type, refer to the **HyperLinkCellType ('HyperLinkCellType Class' in the on-line documentation)** class.

For more information on the corresponding event when a user clicks on a hyperlink, refer to the **FpSpread.ButtonClicked ('ButtonClicked Event' in the on-line documentation)** event.

Sorting and Filtering Hyperlinks by Text

After creating hyperlinks, you can also perform sorting and filtering operations on them by text values.

To do this, you can use the **UseModelValueAsText ('UseModelValueAsText Property' in the on-line documentation)** property. By default, the value of this property is a boolean false. When this property is set to true, it fetches the text value from the Data Model and sets this value in the cell of HyperLinkCellType. The text values (fetched from the Data Model) entered in the cells can later be sorted and filtered as and when desired, like other celltypes.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet in which the cells appear.
5. In the property list, select the **Cells** property and then click the button to display the **Cell, Column, and Row Editor**.
6. Select the cells for which you want to set the cell type.
7. In the property list, select the **CellType** property and choose the **HyperLink** cell type.
8. Expand the list of properties under the **CellType** property. Select and set these specific properties as needed.
9. Click **OK** to close the **Cell, Column, and Row Editor**.
10. Click **OK** to close the **SheetView Collection Editor**.

Using Code

1. Define the hyperlink cell by creating an instance of the **HyperLinkCellType ('HyperLinkCellType Class' in the on-line documentation)** class.
2. Be sure to set the size of the cell so that all the text including the hyperlink are visible and display properly.
3. Specify the text that appears in the cell by specifying the **Text ('Text Property' in the on-line documentation)** property for the **HyperLinkCellType ('HyperLinkCellType Class' in the on-line documentation)** object. Specify how much of the text is hyperlink using the **LinkArea ('LinkArea Property' in the on-line documentation)** property for that object.
4. Specify the appearance of the hyperlink by setting properties, such as **LinkColor ('LinkColor Property' in the on-line documentation)**.
5. Assign the hyperlink cell type to a cell or range of cells by setting the **CellType ('CellType Property' in the**

on-line documentation) property for a cell, column, row, or style to the **HyperLinkCellType** (**'HyperLinkCellType Class' in the on-line documentation**) object.

6. If you want to perform sort/filter operations on the hyperlink cell, set the **UseModelValueAsText** (**'UseModelValueAsText Property' in the on-line documentation**) property to true in order to indicate that the text for the hyperlink must be used from the data model.

Example

This example code sets the size of the cell (by column and row), creates a hyperlink button, and specifies the destination URL.

Also, it shows how to use the **UseModelValueAsText** property to indicate that the text value for the hyperlink must be fetched from the Data Model.

C#

```
fpSpread1.ActiveSheet.Columns[1].Width = 145;
fpSpread1.ActiveSheet.Rows[1].Height = 45;
FarPoint.Win.Spread.CellType.HyperLinkCellType hlnkcell = new
FarPoint.Win.Spread.CellType.HyperLinkCellType();
hlnkcell.Text = "Click to See Our Web Site";
hlnkcell.Link = "http://developer.mescius.com";
hlnkcell.LinkArea = new LinkArea(9,16);
hlnkcell.LinkColor = Color.DarkGreen;
hlnkcell.VisitedLinkColor = Color.Chartreuse;
fpSpread1.ActiveSheet.Cells[1, 1].CellType = hlnkcell;
// Set value in the Data Model
fpSpread1.ActiveSheet.SetValue(1,1,"Click to see our website");
hlnkcell.UseModelValueAsText = true; // Indicates that text must be used from data
model
```

VB

```
fpSpread1.ActiveSheet.Columns(1).Width = 145
fpSpread1.ActiveSheet.Rows(1).Height = 45
Dim hlnkcell As New FarPoint.Win.Spread.CellType.HyperLinkCellType()
hlnkcell.Text = "Click to See Our Web Site"
hlnkcell.Link = "http://developer.mescius.com"
hlnkcell.LinkArea = New LinkArea(9,16)
hlnkcell.LinkColor = Color.DarkGreen
hlnkcell.VisitedLinkColor = Color.Chartreuse
fpSpread1.ActiveSheet.Cells(1, 1).CellType = hlnkcell
' Set value in the Data Model
fpSpread1.ActiveSheet.SetValue(1, 1, "Click to see our website")
' Indicates that text must be used from data model
hlnkcell.UseModelValueAsText = True
```

Using Code

1. Define the hyperlink cell by creating an instance of the **HyperLinkCellType** (**'HyperLinkCellType Class' in the on-line documentation**) class.
2. Be sure to set the size of the cell so that all the text including the hyperlink are visible and display properly.
3. Specify the text that appears in the cell by specifying the **Text** (**'Text Property' in the on-line documentation**) property for the **HyperLinkCellType** (**'HyperLinkCellType Class' in the on-line documentation**) object. Specify how much of the text is hyperlink using the **LinkAreas** (**'LinkAreas Property' in the on-line documentation**) property for that object.

- Specify the appearance of the hyperlink by setting properties, such as **LinkColor** ('**LinkColor Property**' in the on-line documentation).
- Assign the hyperlink cell type to a cell or range of cells by setting the **CellType** ('**CellType Property**' in the on-line documentation) property for a cell, column, row, or style to the **HyperLinkCellType** ('**HyperLinkCellType Class**' in the on-line documentation) object.

Example

This example creates a hyperlink cell with multiple links.

C#

```
fpSpread1.ActiveSheet.Columns[0].Width = 145;
fpSpread1.ActiveSheet.Rows[0].Height = 45;
FarPoint.Win.Spread.CellType.HyperLinkCellType mhp = new
FarPoint.Win.Spread.CellType.HyperLinkCellType();
mhp.Text = "Google and Microsoft";
string[] s = new string[]{"www.google.com", "www.microsoft.com"};
mhp.Links = s;
mhp.VisitedLinkColor = Color.Maroon;
LinkArea[] la = new LinkArea[]{new LinkArea(0, 8), new LinkArea(13, 9)};
mhp.LinkAreas = la;
fpSpread1.ActiveSheet.Cells[0, 0].CellType = mhp;
```

VB

```
fpSpread1.ActiveSheet.Columns(0).Width = 145
fpSpread1.ActiveSheet.Rows(0).Height = 45
Dim mhp As New FarPoint.Win.Spread.CellType.HyperLinkCellType
mhp.Text = "Google and Microsoft"
Dim s() As String = New String() {"www.google.com", "www.microsoft.com"}
mhp.Links = s
mhp.VisitedLinkColor = Color.Maroon
Dim la() As LinkArea = New LinkArea() {New LinkArea(0, 8), New LinkArea(13, 9)}
mhp.LinkAreas = la
fpSpread1.ActiveSheet.Cells(0, 0).CellType = mhp
```

Using the Spread Designer

- Select the cell or cells in the work area.
- In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the **HyperLink** cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Select and set those properties as needed.
Or right-click on the cell or cells and select **Cell Type**. From the list, select **HyperLink**. In the **CellType** editor, set the properties you need. Click **Apply**. If you create multiple hyper links then make sure the text is set as well. Note that if you were to create the previous code sample in the designer then the text would need to contain 23 characters since the second link starts at 13 and continues for 9 characters. The text is zero based.
- From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting an Image Cell

You can display a graphic image in a cell using the image cell type. An image cell shows an image as data. If the data type for a bound column is a bit array then the default cell type for that bound column would be an image cell type.

An image object can be assigned to the **Value** ('**Value Property**' in the on-line documentation) property of a cell.

The image or serialized image object must be in the data model.

To create a cell that contains an image, use the **ImageCellType** ('**ImageCellType Class**' in the **on-line documentation**) class. Create an image cell using the following procedure. The result is shown in this figure.

	A	B	C
1			
2		SPREAD.NET	
3			

Notice that both cells have a magenta background, but the one with the image only shows the magenta through the transparent areas specified by the transparency color property and the transparency tolerance property. The image is mostly white so blocks out most of the magenta. The lettering that is more blue is beyond the tolerance and so is not transparent. When you set a transparency color, the background behind the picture shows through in the area that originally had the color you specify. Setting the transparency tolerance to 255 will cause everything to be transparent so you may wish to use a value less than 255.

For more information on the properties and methods of this cell type, refer to the **ImageCellType** ('**ImageCellType Class**' in the **on-line documentation**) class.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet in which the cells appear.
5. In the property list, select the **Cells** property and then click the button to display the **Cell, Column, and Row Editor**.
6. Select the cells for which you want to set the cell type.
7. In the property list, select the **CellType** property and choose the **Image** cell type.
8. Expand the list of properties under the **CellType** property. Select and set these specific properties as needed.
9. Click **OK** to close the **Cell, Column, and Row Editor**.
10. Click **OK** to close the **SheetView Collection Editor**.

Using Code

1. Define the image cell by creating an instance of the **ImageCellType** ('**ImageCellType Class**' in the **on-line documentation**) class.
2. Create the image.
3. Specify what appears in the cell by setting the **Value** ('**Value Property**' in the **on-line documentation**) property for the **ImageCellType** ('**ImageCellType Class**' in the **on-line documentation**) object.
4. Specify the appearance of the image by setting properties such as **Style**.
5. Assign the image cell type to a cell or range of cells by setting the **CellType** property for a cell, column, row, or style to the **ImageCellType** ('**ImageCellType Class**' in the **on-line documentation**) object.

Example

This example sets the properties of an image cell type then loads a logo image into a cell using the **Value** ('**Value Property**' in the **on-line documentation**) property. With background color set, the use of transparency can be seen.

C#

```

FarPoint.Win.Spread.CellType.ImageCellType imgct = new
FarPoint.Win.Spread.CellType.ImageCellType();
System.Drawing.Image image = System.Drawing.Image.FromFile("D:\\Logos\\logo.jpg");
imgct.Style = FarPoint.Win.RenderStyle.Stretch;
imgct.TransparencyColor = Color.Black;
imgct.TransparencyTolerance = 20;

fpSpread1.Sheets[0].Cells[1,1,1,2].BackColor = Color.Magenta;
fpSpread1.Sheets[0].Columns[1,2].Width = 100;
fpSpread1.Sheets[0].Rows[1,1].Height = 50;
fpSpread1.Sheets[0].Cells[1,1,2,2].CellType = imgct;
fpSpread1.Sheets[0].Cells[1,1].Value = image;

```

VB

```

Dim imgct As New FarPoint.Win.Spread.CellType.ImageCellType()
Dim image As System.Drawing.Image = System.Drawing.Image.FromFile("D:\Logos\logo.jpg")
imgct.Style = FarPoint.Win.RenderStyle.Stretch
imgct.TransparencyColor = Color.Black
imgct.TransparencyTolerance = 20

fpSpread1.Sheets(0).Cells(1,1,1,2).BackColor = Color.Magenta
fpSpread1.Sheets(0).Columns(1,2).Width =100
fpSpread1.Sheets(0).Rows(1,1).Height = 50
fpSpread1.Sheets(0).Cells(1,1,2,2).CellType = imgct
fpSpread1.Sheets(0).Cells(1, 1).Value = image

```

When loading multiple images in image cell types, you can directly assign the image file paths in cell values. This improves the memory management and Spread's performance. The below image shows multiple images added into the image cell types.

	A	B	C	D	E
1					
2					
3					
4					
5					
6					

This example loads multiple images into image cell types by assigning the image file paths in the **Value ('Value Property' in the on-line documentation)** property.

C#

```

FarPoint.Win.Spread.CellType.ImageCellType imgct = new
FarPoint.Win.Spread.CellType.ImageCellType();
imgct.Style = FarPoint.Win.RenderStyle.Stretch;
imgct.TransparencyColor = System.Drawing.Color.Black;
imgct.TransparencyTolerance = 20;

```

```

fpSpread1.Sheets[0].Columns[1, 3].Width = 100;
fpSpread1.Sheets[0].Rows[1, 2].Height = 70;
fpSpread1.Sheets[0].Cells[1, 1, 3, 3].CellType = imgct;
fpSpread1.Sheets[0].Cells[1, 1].Value = "D:\\Logos\\Spread.png";
fpSpread1.Sheets[0].Cells[1, 2].Value = "D:\\Logos\\Mescius.png";
fpSpread1.Sheets[0].Cells[1, 3].Value = "D:\\Logos\\ActiveReports.png";
fpSpread1.Sheets[0].Cells[2, 1].Value = "D:\\Logos\\ComponentOne.png";
fpSpread1.Sheets[0].Cells[2, 2].Value = "D:\\Logos\\DocSol.png";
fpSpread1.Sheets[0].Cells[2, 3].Value = "D:\\Logos\\Wijmo.png";

```

VB

```

Dim imgct As FarPoint.Win.Spread.CellType.ImageCellType = New
FarPoint.Win.Spread.CellType.ImageCellType()

```

```

imgct.Style = FarPoint.Win.RenderStyle.Stretch
imgct.TransparencyColor = System.Drawing.Color.Black
imgct.TransparencyTolerance = 20
fpSpread1.Sheets(0).Columns(1, 3).Width = 100
fpSpread1.Sheets(0).Rows(1, 2).Height = 70
fpSpread1.Sheets(0).Cells(1, 1, 3, 3).CellType = imgct

```

```

fpSpread1.Sheets(0).Cells(1, 1).Value = "D:\Logos\Spread.png"
fpSpread1.Sheets(0).Cells(1, 2).Value = "D:\Logos\Mescius.png"
fpSpread1.Sheets(0).Cells(1, 3).Value = "D:\Logos\ActiveReports.png"
fpSpread1.Sheets(0).Cells(2, 1).Value = "D:\Logos\ComponentOne.png"
fpSpread1.Sheets(0).Cells(2, 2).Value = "D:\Logos\DocSol.png"
fpSpread1.Sheets(0).Cells(2, 3).Value = "D:\Logos\Wijmo.png"

```

Using the Spread Designer

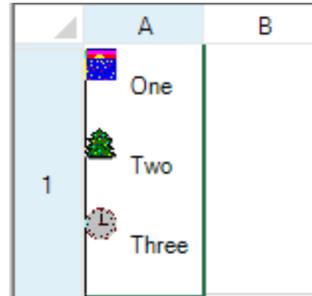
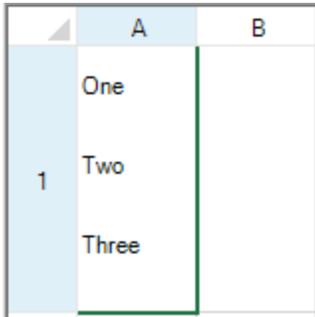
1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the **Image** cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Select and set those properties as needed.
Or right-click on the cell or cells and select **Cell Type**. From the list, select **Image**. In the **CellType** editor, set the properties you need. Click **Apply**.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting a List Box Cell

You can use a list box cell to display a list, which allows the user to select from the displayed list. You can specify the list of items, whether to include icons to appear along with text, the number of items that are displayed at any time, and other aspects of the display.

Text only

Text and icon



To create a cell that acts like a list box, use the **ListBoxCellType** ('**ListBoxCellType Class**' in the **on-line documentation**) class. Create a list box cell using the following procedure.

Customizing the List Appearance

Here is a summary of the appearance properties that you can use to customize the list box.

Property	Description
EditorValue (' EditorValue Property ' in the on-line documentation)	Sets what value is written to the underlying data model.
ImageList (' ImageList Property ' in the on-line documentation)	Sets an image list for displaying icons along with text in the list.
ItemHeight (' ItemHeight Property ' in the on-line documentation)	Sets the height for each item in the list.
ItemData (' ItemData Property ' in the on-line documentation)	Sets item data, which is different from the items that are displayed, to use for the list.
Items (' Items Property ' in the on-line documentation)	Sets items to use for the list.

For a complete list of the properties and methods of this cell type, refer to the **ListBoxCellType** ('**ListBoxCellType Class**' in the **on-line documentation**) class. For information on the combo box (which includes both a list box and an editable area), refer to **Setting a Combo Box Cell**.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet in which the cells appear.
5. In the property list, select the **Cells** property and then click the button to display the **Cell, Column, and Row Editor**.
6. Select the cells for which you want to set the cell type.
7. In the property list, select the **CellType** property and choose the **ListBox** cell type.
8. Expand the list of properties under the **CellType** property. Select and set these specific properties as needed.
9. Click **OK** to close the **Cell, Column, and Row Editor**.
10. Click **OK** to close the **SheetView Collection Editor**.

Using Code

1. Define a list box cell by creating an instance of the **ListBoxCellType** ('**ListBoxCellType Class**' in the **on-**

line documentation) class.

2. Specify the items in the list that appear as part of the list box. You can either use the **Items ('Items Property' in the on-line documentation)** property of the **ListBoxCellType ('ListBoxCellType Class' in the on-line documentation)** class or define a string and pass that in when creating the instance of the class.
3. Assign the list box cell type to a cell or range of cells by setting the **CellType ('CellType Property' in the on-line documentation)** property for a cell, column, row, or style to the **ListBoxCellType ('ListBoxCellType Class' in the on-line documentation)** object.

Example

This example creates a list box cell and uses images from an image list control.

C#

```
FarPoint.Win.Spread.CellType.ListBoxCellType listcell = new
FarPoint.Win.Spread.CellType.ListBoxCellType();
listcell.ImageList = ImageList1;
listcell.ItemData = new string[] { "One", "Two", "Three"};
listcell.Items = new string[] { "One", "Two", "Three"};
listcell.ItemHeight = 40;
fpSpread1.ActiveSheet.Cells[0, 0].CellType = listcell;
fpSpread1.ActiveSheet.Rows[0].Height = 120;
```

VB

```
Dim listcell As New FarPoint.Win.Spread.CellType.ListBoxCellType()
listcell.ImageList = ImageList1
listcell.ItemData = New String() {"One", "Two", "Three"}
listcell.Items = New String() {"One", "Two", "Three"}
listcell.ItemHeight = 40
fpSpread1.ActiveSheet.Cells(0, 0).CellType = listcell
fpSpread1.ActiveSheet.Rows(0).Height = 120
```

Using the Spread Designer

1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the **ListBox** cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Select and set those properties as needed.
Or right-click on the cell or cells and select **Cell Type**. From the list, select **ListBox**. In the **CellType** editor, set the properties you need. Click **Apply**.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting a Multiple-Column Combo Box Cell

You can create a combo box cell with multiple columns in the drop-down list. You can provide a drop-down list as well as an editable area allowing the user to type in values as well as choosing from a displayed list. You specify the list of items, the number that is displayed at any time, and whether the cell is editable by the user.

	A	B	C	D	E	F	G	H
1	Canyon							
2	LName	FName	HospNo	Street				
3	Canyon	Valerie	1111155	70 Hathaway Road				
4	Harper	Simon	1111344	19 Long Lane				
5	Norcott	Anthony	1212459	102 Rebels Lane				
6	Tyrie	Amanda	1335225	32 Lordship lane				
7	Clayton	Roberts	6754434	76 Princess Road				
8								

The Spread control has a **ButtonDrawMode** ('**ButtonDrawMode Property**' in the on-line documentation) property for button cells and combo box cells. This property allows you to always show a button, or show buttons in the current column, row, or cell.

To create a cell that acts like a multiple-column combo box, use the **MultiColumnComboBoxCellType** ('**MultiColumnComboBoxCellType Class**' in the on-line documentation) class. Create such a combo box cell using the following procedure.

Customizing the Display

You can customize the display of the multiple-column combo box cell by setting the following properties.

Property

BackgroundImage ('**BackgroundImage Property**' in the on-line documentation)

ButtonAlign ('**ButtonAlign Property**' in the on-line documentation)

ColumnEdit ('**ColumnEdit Property**' in the on-line documentation)

DataColumn ('**DataColumn Property**' in the on-line documentation)

DataSourceList ('**DataSourceList Property**' in the on-line documentation)

ListAlignment ('**ListAlignment Property**' in the on-line documentation)

ListOffset ('**ListOffset Property**' in the on-line documentation)

ListWidth ('**ListWidth Property**' in the on-line documentation)

MaxDrop ('**MaxDrop Property**' in the on-line documentation)

StringTrim ('**StringTrim Property**' in the on-line documentation)

SubEditor ('**SubEditor Property**' in the on-line documentation)

Description

Sets the background image in the cell.

Sets where the buttons are displayed.

Sets the column of the list to use for the edit portion.

Sets which list column to use as the data column.

Sets the data source for the list portion of the cell.

Sets which side of the editor the list aligns to.

Sets how much the list offsets from the editor.

Sets the width of the list.

Sets the maximum number of items to display in the list at one time.

Sets how to trim characters that do not fit in the cell.

Sets the subeditor.

Customizing the Operation

You can customize the operation of the multiple-column combo box cell by setting the following properties.

Property	Description
AcceptsArrowKeys ('AcceptsArrowKeys Property' in the on-line documentation)	Sets how arrow keys are processed by the cell.
AutoSearch ('AutoSearch Property' in the on-line documentation)	Sets how a list of items in a combo box cell is searched based on input of a character key.
DataColumn ('DataColumn Property' in the on-line documentation)	Sets which list column to use as the data column.
DataSourceList ('DataSourceList Property' in the on-line documentation)	Sets the data source for the list portion of the cell.
Editable ('Editable Property' in the on-line documentation)	Allows the user to type in the edit portion of the cell.
SubEditor ('SubEditor Property' in the on-line documentation)	Sets the subeditor.

Note that some graphical elements in certain cell types are affected by XP themes (visual styles). Setting the **VisualStyles** property of the Spread component to "off" can allow visual customizations of those graphical cell types to work as expected. For more information, refer to **Using XP Themes with the Component**.

For more information on the properties and methods of this cell type, refer to the **MultiColumnComboBoxCellType** ('MultiColumnComboBoxCellType Class' in the on-line documentation) class. For more information on a standard combo box (single column), refer to **Setting a Combo Box Cell**.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet in which the cells appear.
5. In the property list, select the **Cells** property and then click the button to display the **Cell, Column, and Row Editor**.
6. Select the cells for which you want to set the cell type.
7. In the property list, select the **CellType** property and choose the **MultiColumnComboBox** cell type.
8. Expand the list of properties under the **CellType** property. Select and set these specific properties as needed.
9. Click **OK** to close the **Cell, Column, and Row Editor**.
10. Click **OK** to close the **SheetView Collection Editor**.

Using Code

1. Define a combo box cell by creating an instance of the **MultiColumnComboBoxCellType** ('MultiColumnComboBoxCellType Class' in the on-line documentation) class.
2. Specify the items in the list that appear as part of the combo box. You can either use the **Items** property of the **MultiColumnComboBoxCellType** ('MultiColumnComboBoxCellType Class' in the on-line documentation) class or define a string and pass that in when creating the instance of the class.
3. Specify how the list of items appears. For example, set the **MaxDrop** property to set the maximum number of items to display at a time. If there are more items, a scroll bar appears. You can also set the horizontal alignment of the check box with respect to the cell.
4. Assign the combo box cell type to a cell or range of cells by setting the **CellType** ('CellType Property' in the on-line documentation) property for a cell, column, row, or style to the **MultiColumnComboBoxCellType** ('MultiColumnComboBoxCellType Class' in the on-line documentation) object.

Example

This example creates a multiple-column combo box cell and adds data from a data source.

C#

```
string conStr = "Provider=Microsoft.JET.OLEDB.4.0;data
source=C:\\SpreadStudio\\Common\\Patients2000.mdb";
string sqlStr = "SELECT * FROM Patients";
System.Data.OleDb.OleDbConnection conn = new System.Data.OleDb.OleDbConnection(conStr);
DataSet ds = new DataSet();
System.Data.OleDb.OleDbDataAdapter da = new System.Data.OleDb.OleDbDataAdapter(sqlStr,
conn);
da.Fill(ds);
FarPoint.Win.Spread.CellType.MultiColumnComboBoxCellType mcb = new
FarPoint.Win.Spread.CellType.MultiColumnComboBoxCellType();
mcb.DataSourceList = ds;
mcb.DataColumn = 2;
mcb.ColumnEdit = 2;
mcb.ButtonAlign = FarPoint.Win.ButtonAlign.Left;
mcb.ListAlignment = FarPoint.Win.ListAlignment.Right;
mcb.ListWidth = 500;
mcb.ListOffset = 5;
mcb.MaxDrop = 5;
fpSpread1.ActiveSheet.Cells[0, 0].CellType = mcb;
```

VB

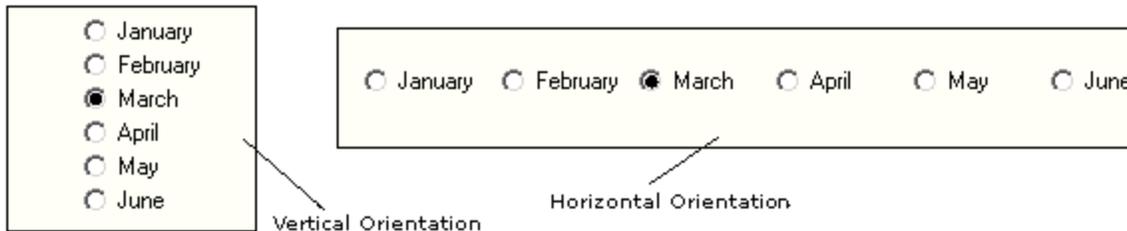
```
Dim conStr As String = "Provider=Microsoft.JET.OLEDB.4.0;data
source=C:\\SpreadStudio\\Common\\Patients2000.mdb"
Dim sqlStr As String = "SELECT * FROM Patients"
Dim conn As New System.Data.OleDb.OleDbConnection(conStr)
Dim ds As DataSet = New DataSet()
Dim da As New System.Data.OleDb.OleDbDataAdapter(sqlStr, conn)
da.Fill(ds)
Dim mcb As New FarPoint.Win.Spread.CellType.MultiColumnComboBoxCellType()
mcb.DataSourceList = ds
mcb.DataColumn = 1
mcb.ButtonAlign = FarPoint.Win.ButtonAlign.Left
mcb.ListWidth = 500
mcb.ListOffset = 5
mcb.MaxDrop = 5
fpSpread1.ActiveSheet.Cells(0, 0).CellType = mcb
```

Using the Spread Designer

1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the **ComboBox** cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Select and set those properties as needed.
Or right-click on the cell or cells and select **Cell Type**. From the list, select **ComboBox**. In the **CellType** editor, set the properties you need. Click **Apply**.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting a Multiple Option Cell

You can define multiple option buttons in a multiple option cell. This cell type offers several option buttons, either horizontally or vertically, for the user to select. Only one button can be selected at a time. The default is for none of the buttons to be selected.



To create a cell that acts like a list of multiple option buttons, use the **MultiOptionCellType ('MultiOptionCellType Class' in the on-line documentation)** class. Create a multiple option cell using the following procedure.

Customizing Display and Operation

You can customize the display and operation of the multiple options in the cell by setting the following properties.

Property

BackgroundImage ('BackgroundImage Property' in the on-line documentation)

EditorValue ('EditorValue Property' in the on-line documentation)

ItemData ('ItemData Property' in the on-line documentation)

Items ('Items Property' in the on-line documentation)

Orientation ('Orientation Property' in the on-line documentation)

Picture ('Picture Property' in the on-line documentation)

TextAlign ('TextAlign Property' in the on-line documentation)

UseMnemonic ('UseMnemonic Property' in the on-line documentation)

Description

Sets the background image for the cell.

Sets which value is written to the underlying data model.

Sets the ItemData to use for the list.

Creates the list to use for the option buttons.

Sets the orientation of the option buttons.

Customizes the option button images.

Sets how text aligns in the cell.

Sets whether access keys (hot keys or keyboard shortcuts) are used in the cell.

For more information on the properties and methods of this cell type, refer to the **MultiOptionCellType ('MultiOptionCellType Class' in the on-line documentation)** class.

For more information on the corresponding event when a user clicks on an option, refer to the **FpSpread.ButtonClicked ('ButtonClicked Event' in the on-line documentation)** event.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet in which the cells appear.

5. In the property list, select the **Cells** property and then click the button to display the **Cell, Column, and Row Editor**.
6. Select the cells for which you want to set the cell type.
7. In the property list, select the **CellType** property and choose the **MultiOption** cell type.
8. Expand the list of properties under the **CellType** property. Select and set these specific properties as needed.
9. Click **OK** to close the **Cell, Column, and Row Editor**.
10. Click **OK** to close the **SheetView Collection Editor**.

Using Code

1. Define the multiple option list cell by creating an instance of the **MultiOptionCellType** (**'MultiOptionCellType Class' in the on-line documentation**) class.
2. Specify the items in the list of options.
3. Assign the multiple option cell type to a cell or range of cells by setting the **CellType** (**'CellType Property' in the on-line documentation**) property for a cell, column, row, or style to the **MultiOptionCellType** (**'MultiOptionCellType Class' in the on-line documentation**) object.

Example

The following example displays a set of options for the user to choose from.

C#

```
FarPoint.Win.Spread.CellType.MultiOptionCellType multcell = new
FarPoint.Win.Spread.CellType.MultiOptionCellType();
multcell.Items = new String[] { "Carbon", "Oxygen", "Hydrogen" };
multcell.Orientation = FarPoint.Win.RadioOrientation.Horizontal;
fpSpread1.ActiveSheet.Cells[0, 0].CellType = multcell;
fpSpread1.ActiveSheet.Columns[0].Width = 220;
```

VB

```
Dim multcell As New FarPoint.Win.Spread.CellType.MultiOptionCellType()
multcell.Items = New String() { "Carbon", "Oxygen", "Hydrogen" }
multcell.Orientation = FarPoint.Win.RadioOrientation.Horizontal
fpSpread1.ActiveSheet.Cells(0, 0).CellType = multcell
fpSpread1.ActiveSheet.Columns(0).Width = 220
```

Using the Spread Designer

1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the **MultiOption** cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Select and set those properties as needed.
Or right-click on the cell or cells and select **Cell Type**. From the list, select **MultiOption**. In the **CellType** editor, set the properties you need. Click **Apply**.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting a Progress Indicator Cell

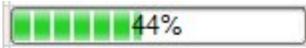
A progress indicator cell displays a progress indicator control across the entire cell. You can specify the color of the fill, the text to display, the color of the text and other properties.



To create a cell that acts like a progress indicator, use the **ProgressCellType** (**'ProgressCellType Class' in the on-line documentation**) class. Create a progress indicator cell using the procedure described here.

Customize the Indicator

You can fill in the indicator with a solid color, by default, or with individual bars, as shown in this figure.



You can customize the display and operation of the progress indicator in the cell by setting the following properties.

Property

BackgroundImage (**'BackgroundImage Property' in the on-line documentation**)

FillColor (**'FillColor Property' in the on-line documentation**)

FillColor2 (**'FillColor2 Property' in the on-line documentation**)

FillTextColor (**'FillTextColor Property' in the on-line documentation**)

GradientMode (**'GradientMode Property' in the on-line documentation**)

Maximum (**'Maximum Property' in the on-line documentation**)

Minimum (**'Minimum Property' in the on-line documentation**)

Orientation (**'Orientation Property' in the on-line documentation**)

Picture (**'Picture Property' in the on-line documentation**)

ShowText (**'ShowText Property' in the on-line documentation**)

Style (**'Style Property' in the on-line documentation**)

Text (**'Text Property' in the on-line documentation**)

TextStyle (**'TextStyle Property' in the on-line documentation**)

Description

Sets the background image for the cell.

Sets the color to use for the filled part of the progress indicator.

Sets the second fill color to use for the gradient part of the progress indicator.

Sets the color to use for the text in the filled part of the indicator.

Sets the gradient mode for a gradient style progress indicator.

Sets the maximum value for user entry.

Sets the minimum value for user entry.

Sets the orientation of the progress bar.

Sets the image to use for the progress bar when the style is set to Picture.

Sets whether the percent filled string is displayed.

Sets the style of the progress bar(s).

Sets the string to use when **TextStyle** is set to Custom.

Sets how the text portion of the progress bar is displayed.

With these properties, you can set the various aspects of the text, you can set a picture to display, and you can define the colors, even specifying two colors for a gradient from one color to another.

Note that some graphical elements in certain cell types are affected by XP themes (visual styles). Setting the **VisualStyles** (**'VisualStyles Property' in the on-line documentation**) property of the Spread component to "off" can allow visual customizations of those graphical cell types to work as expected. For more information, refer to **Using XP Themes with the Component**.

For more information on the properties and methods of this cell type, refer to the **ProgressCellType**

(**'ProgressCellType Class' in the on-line documentation**) class.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet in which the cells appear.
5. In the property list, select the **Cells** property and then click the button to display the **Cell, Column, and Row Editor**.
6. Select the cells for which you want to set the cell type.
7. In the property list, select the **CellType** property and choose the **Progress** cell type.
8. Expand the list of properties under the **CellType** property. Select and set these specific properties as needed.
9. Click **OK** to close the **Cell, Column, and Row Editor**.
10. Click **OK** to close the **SheetView Collection Editor**.

Using Code

1. Define the progress indicator cell by creating an instance of the **ProgressCellType ('ProgressCellType Class' in the on-line documentation)** class.
2. Format and specify the appearance of the progress indicator.
3. Assign the progress indicator cell type to a cell or range of cells by setting the **CellType ('CellType Property' in the on-line documentation)** property for a cell, column, row, or style to the **ProgressCellType ('ProgressCellType Class' in the on-line documentation)** object.

Example

This example creates a progress cell.

C#

```
FarPoint.Win.Spread.CellType.ProgressCellType progcell = new
FarPoint.Win.Spread.CellType.ProgressCellType();
progcell.FillColor = Color.Red;
fpSpread1.ActiveSheet.Cells[0, 0].CellType = progcell;
fpSpread1.ActiveSheet.Cells[0, 0].VisualStyles = FarPoint.Win.VisualStyles.Off;
fpSpread1.ActiveSheet.Cells[0, 0].Value = 50;
```

VB

```
Dim progcell As New FarPoint.Win.Spread.CellType.ProgressCellType()
progcell.FillColor = Color.Red
fpSpread1.ActiveSheet.Cells(0, 0).CellType = progcell
fpSpread1.ActiveSheet.Cells(0, 0).VisualStyles = FarPoint.Win.VisualStyles.Off
fpSpread1.ActiveSheet.Cells(0, 0).Value = 50\
```

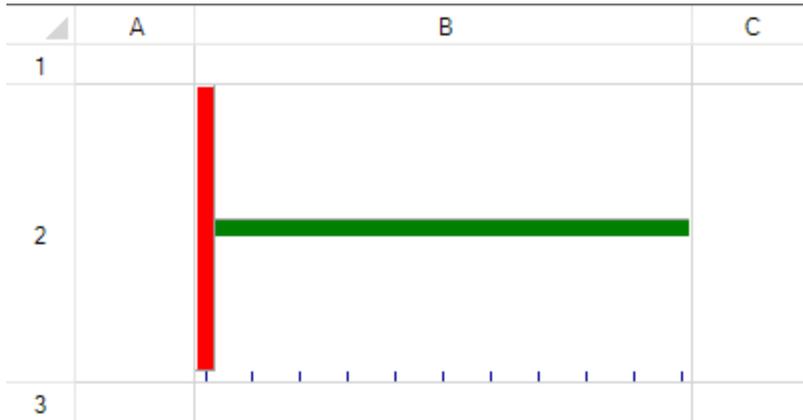
Using the Spread Designer

1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the **Progress** cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Select and set those properties as needed.
Or right-click on the cell or cells and select **Cell Type**. From the list, select **Progress**. In the **CellType** editor, set the properties you need. Click **Apply**.

- From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting a Slider Cell

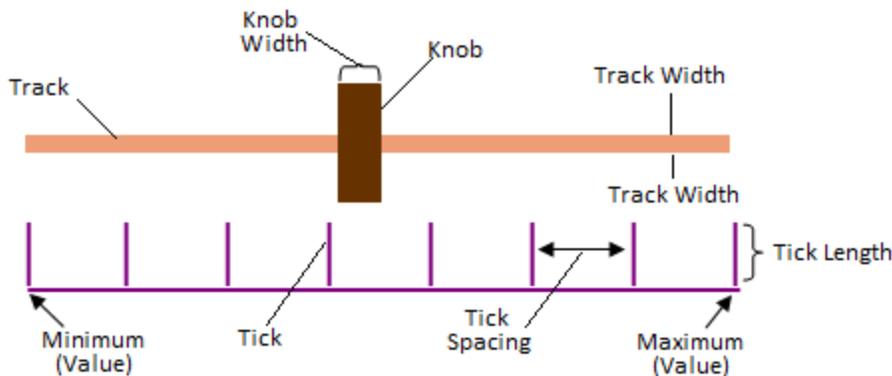
A slider cell displays a slider control in the cell.



To create a cell that acts like a slider, use the **SliderCellType ('SliderCellType Class' in the on-line documentation)** class and follow the procedure described in this topic.

Customizing the Slider

The parts of the slider (corresponding to their property names) are shown here. For this example, the orientation is horizontal and solid colors are used (as opposed to pictures).



You can customize the display and operation of the slider in the cell by setting the following properties.

Property

BackgroundImage ('BackgroundImage Property' in the on-line documentation)

ChangeOnFocus ('ChangeOnFocus Property' in the on-line documentation)

KnobColor ('KnobColor Property' in the on-line documentation)

KnobPicture ('KnobPicture Property' in the on-line documentation)

Customization

Sets the background image for the cell.

Sets whether the slider moves with the initial click.

Sets the color of the slider knob.

Customizes the slider knob image.

KnobWidth ('KnobWidth Property' in the on-line documentation)	Sets the width (in pixels) of the slider knob.
Maximum ('Maximum Property' in the on-line documentation)	Sets the maximum value for user entry.
Minimum ('Minimum Property' in the on-line documentation)	Sets the minimum value for user entry.
Orientation ('Orientation Property' in the on-line documentation)	Sets the orientation of the slider.
TickColor ('TickColor Property' in the on-line documentation)	Sets the color of the slider tick mark.
TickLength ('TickLength Property' in the on-line documentation)	Sets the size of the slider tick mark.
TickSpacing ('TickSpacing Property' in the on-line documentation)	Sets how frequently to space the tick marks.
TrackColor ('TrackColor Property' in the on-line documentation)	Sets the color of the slider track.
TrackPicture ('TrackPicture Property' in the on-line documentation)	Customizes the image for the slider track.
TrackWidth ('TrackWidth Property' in the on-line documentation)	Sets the width (in pixels) of the slider track.

You can programmatically change the location of the knob in the slider cell by setting the **Value** ('Value Property' in the on-line documentation) property to a value that is greater than or equal to the minimum or less than or equal to the maximum setting.

Note that some graphical elements in certain cell types are affected by XP themes (visual styles). Setting the **VisualStyles** ('VisualStyles Property' in the on-line documentation) property of the Spread component to "off" can allow visual customizations of those graphical cell types to work as expected. For more information, refer to **Using XP Themes with the Component**.

For more information on the properties and methods of this cell type, refer to the **SliderCellType** ('SliderCellType Class' in the on-line documentation) class.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet in which the cells appear.
5. In the property list, select the **Cells** property and then click the button to display the **Cell, Column, and Row Editor**.
6. Select the cells for which you want to set the cell type.
7. In the property list, select the **CellType** property and choose the **Slider** cell type.
8. Expand the list of properties under the **CellType** property. Select and set these specific properties as needed.
9. Click **OK** to close the **Cell, Column, and Row Editor**.
10. Click **OK** to close the **SheetView Collection Editor**.

Using Code

1. Define the slider cell by creating an instance of the **SliderCellType** ('SliderCellType Class' in the on-line documentation) class.
2. Format and specify the appearance of the slider.
3. Assign the slider cell type to a cell or range of cells by setting the **CellType** ('CellType Property' in the on-

line documentation) property for a cell, column, row, or style to the **SliderCellType** ('SliderCellType Class' in the on-line documentation) object.

Example

This example creates a slider cell.

C#

```
fpSpread1.ActiveSheet.Columns[1].Width = 250;
fpSpread1.ActiveSheet.Rows[1].Height = 150;
fpSpread1.VisualStyles = FarPoint.Win.VisualStyles.Off;

FarPoint.Win.Spread.CellType.SliderCellType slider = new
FarPoint.Win.Spread.CellType.SliderCellType();
slider.BackgroundImage = new
FarPoint.Win.Picture(Image.FromFile("C:\\images\\scene.jpg"));
slider.ChangeOnFocus = true;

slider.KnobColor = Color.Red;
// Or if you want to use an image instead of a solid color:
// slider.KnobPicture = new
FarPoint.Win.Picture(Image.FromFile("../images\\brush.gif"));
slider.KnobWidth = 10;
slider.Maximum = 200;
slider.Minimum = 0;
slider.Orientation = FarPoint.Win.SliderOrientation.Horizontal;
slider.TickColor = Color.DarkBlue;
slider.TickLength = 5;
slider.TickSpacing = 20;
slider.TrackColor = Color.Green;
// Or if you want to use an image instead of a solid color:
// slider.TrackPicture = new
FarPoint.Win.Picture(Image.FromFile("../images\\pattern.jpg"));
slider.TrackWidth = 10;

fpSpread1.ActiveSheet.Cells[1, 1].CellType = slider;
```

VB

```
fpSpread1.ActiveSheet.Columns(1).Width = 250
fpSpread1.ActiveSheet.Rows(1).Height = 150
fpSpread1.VisualStyles = FarPoint.Win.VisualStyles.Off

Dim slider As New FarPoint.Win.Spread.CellType.SliderCellType()
slider.BackgroundImage = New
FarPoint.Win.Picture(Image.FromFile("C:\\images\\scene.jpg"))
slider.ChangeOnFocus = True
slider.KnobColor = Color.Red
` Or if you want to use an image instead of a solid color:
` slider.KnobPicture = New
FarPoint.Win.Picture(Image.FromFile("../images\\brush.gif"))
slider.KnobWidth = 10
slider.Maximum = 200
slider.Minimum = 0
slider.Orientation = FarPoint.Win.SliderOrientation.Horizontal
slider.TickColor = Color.DarkBlue
```

```
slider.TickLength = 5
slider.TickSpacing = 20
slider.TrackColor = Color.Green
` Or if you want to use an image instead of a solid color:
` slider.TrackPicture = New
FarPoint.Win.Image.FromFile("../images/pattern.jpg")
slider.TrackWidth = 10
fpSpread1.ActiveSheet.Cells(1, 1).CellType = slider
```

Using the Spread Designer

1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the **Slider** cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Select and set those properties as needed.
Or right-click on the cell or cells and select **Cell Type**. From the list, select **Slider**. In the **CellType** editor, set the properties you need. Click **Apply**.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Understanding Additional Features of Cell Types

You can specify the cell type for individual cells, columns, rows, a range of cells, or an entire sheet. For any cell type there are properties of a cell that can be set. In general, working with cell types includes defining the cell type, setting the properties, and applying that cell type to cells.

The following topics provide additional information about customizing cells based on cell types:

- **Allowing the Display of Buttons in a Cell**
- **Displaying Spin Buttons**
- **Limiting Values for a Numeric Cell**
- **Customizing the Pop-Up Date-Time Control**
- **Customizing the Pop-Up Calculator Control**
- **Working with a SubEditor**
- **Creating a Custom Cell Type**

Allowing the Display of Buttons in a Cell

You can allow or restrict the display of buttons in cells that are the specific graphical cell types that allow buttons. Use the FpSpread **ButtonDrawMode** (**'ButtonDrawMode Property' in the on-line documentation**) property to set the limits on where buttons can be displayed. These refer to primary buttons, such as those found in these cell types:

- button cells (**ButtonCellType** (**'ButtonCellType Class' in the on-line documentation**))
- check box cells (**CheckBoxCellType** (**'CheckBoxCellType Class' in the on-line documentation**))
- hyperlink cells, (**HyperLinkCellType** (**'HyperLinkCellType Class' in the on-line documentation**))
- multiple-option button cells (**MultiOptionCellType** (**'MultiOptionCellType Class' in the on-line documentation**))

Secondary buttons refer to those buttons in the cell that are part of the control that allow you to change values in the cell, such as the spin buttons or the drop-down arrow in a combo box.

For more details on possible settings, refer to the **ButtonDrawModes** (**'ButtonDrawModes Enumeration' in the on-line documentation**) enumeration.

For details on setting the display of primary and secondary buttons as part of a cell's appearance, see the **Appearance**

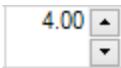
('Appearance Class' in the on-line documentation) class **DrawPrimaryButton** ('DrawPrimaryButton Property' in the on-line documentation) property and **DrawSecondaryButton** ('DrawSecondaryButton Property' in the on-line documentation) property.

You can also make use of the FpSpread **ButtonClicked** ('ButtonClicked Event' in the on-line documentation) event.

For more information about cell types that allow buttons to be drawn in a cell, refer to **Working with Graphical Cell Types**. For more information about cell types that can display spin buttons, see **Displaying Spin Buttons**.

Displaying Spin Buttons

In some cell types, you can display spin buttons, to let users change the value in a cell quickly. Spin buttons are a set of two arrow buttons that appear together, one for increasing the value and one for decreasing the value. Spin buttons appear when the cell is in edit mode. The following figure illustrates spin buttons on the right-hand side of a number cell.



You specify how much the value changes when the user clicks a spin button and you determine whether the value wraps when the minimum or maximum value is reached.

Spin buttons can be displayed in:

- **CurrencyCellType** ('CurrencyCellType Class' in the on-line documentation) (see **Setting a Currency Cell**)
- **DateTimeCellType** ('DateTimeCellType Class' in the on-line documentation) (see **Setting a Date-Time Cell**)
- **NumberCellType** ('NumberCellType Class' in the on-line documentation) (see **Setting a Number Cell**)
- **PercentCellType** ('PercentCellType Class' in the on-line documentation) (see **Setting a Percent Cell**)

For the numeric cell types, if the cursor is left of the decimal point, by default the spin button increments the value using the whole number. If the cursor is to the right, by default the spin button increments the value using the first, or tenths, decimal place.

For the date-time cell type, the day, month, year, etc. are incremented or decremented depending on which part of the date and time the cursor is in.

The spin buttons expand vertically to fill the entire height of the cell, so the taller the cell, the bigger the spin buttons. The properties of the currency, number, and percent cell types that relate to spin buttons are listed in the following table. The date-time cell type only provides the **SpinButton** property.

Property	Description
SpinButton	Sets whether a spin button is displayed when editing.
SpinDecimalIncrement	Sets the amount by which the value increments when using the spin buttons and the cursor is in the decimal portion.
SpinIntegerIncrement	Sets the amount by which the value increments when using the spin buttons and the cursor is in the integer portion.
SpinWrap	Sets whether the value wraps when the minimum or maximum is reached.

For more information refer to the property for each cell type:

- **CurrencyCellType.SpinButton** ('SpinButton Property' in the on-line documentation)
- **DateTimeCellType.SpinButton** ('SpinButton Property' in the on-line documentation)

- `NumberCellType.SpinButton` ('**SpinButton Property**' in the on-line documentation)
- `PercentCellType.SpinButton` ('**SpinButton Property**' in the on-line documentation)

For information on the editable cell types, refer to **Working with Editable Cell Types**.

For information on other features of cell types, refer to **Understanding Additional Features of Cell Types**.

Using Code

- Display the spin buttons by setting the **SpinButton** property to `True`.
- Specify the amount by which the value in the cell is incremented or decremented when the user clicks on the spin buttons by setting the **SpinIncrement** property to a nonzero value.
- Specify whether the value in the cell wraps when the cell reaches the minimum or maximum value by setting the **SpinWrap** property to `True` or `False`.

Example

This example specifies a currency cell and sets spin button properties.

C#

```
FarPoint.Win.Spread.CellType.CurrencyCellType crcycell = new
FarPoint.Win.Spread.CellType.CurrencyCellType();
crcycell.SpinButton = true;
crcycell.SpinDecimalIncrement = 0.5F;
crcycell.SpinIntegerIncrement = 5;
crcycell.SpinWrap = true;
crcycell.MaximumValue = 500;
crcycell.MinimumValue = -100;
fpSpread1.Sheets[0].Cells[6,2].CellType = crcycell;
fpSpread1.Sheets[0].Cells[6,2].Value = 443.3482;
```

VB

```
Dim curr As New FarPoint.Win.Spread.CellType.CurrencyCellType()
curr.SpinButton = True
curr.SpinDecimalIncrement = 0.5
curr.SpinIntegerIncrement = 5
curr.SpinWrap = True
curr.MaximumValue = 500
curr.MinimumValue = -100
fpSpread1.ActiveSheet.Cells(0, 0).CellType = curr
fpSpread1.ActiveSheet.Cells(0, 0).Value = 443.348
```

Using the Spread Designer

1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Set **SpinButton** to `True` to display the spin buttons. Select and set other properties as needed. Or right-click on the cell or cells and select **Cell Type**. From the list, select the cell type. In the **CellType** editor, set the properties you need. Click **Apply**.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Limiting Values for a Numeric Cell

You can set the minimum and maximum values that can be entered in a cell and notify the user with a message if the entry is smaller than the minimum or larger than the maximum.

The cell types that allow you to set a minimum and maximum value are:

- **CurrencyCellType** ('**CurrencyCellType Class**' in the on-line documentation)
- **NumberCellType** ('**NumberCellType Class**' in the on-line documentation)
- **PercentCellType** ('**PercentCellType Class**' in the on-line documentation)
- **ProgressCellType** ('**ProgressCellType Class**' in the on-line documentation)
- **SliderCellType** ('**SliderCellType Class**' in the on-line documentation)

For more information about these cell types, refer to the following topics:

- **Setting a Currency Cell**
- **Setting a Number Cell**
- **Setting a Percent Cell**
- **Setting a Progress Indicator Cell**
- **Setting a Slider Cell**

For information on other features of cell types, refer to **Understanding Additional Features of Cell Types**.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet in which the cells appear.
5. In the property list, select the **Cells** property and then click the button to display the **Cell, Column, and Row Editor**.
6. Select the cells for which you want to set the cell type.
7. In the property list, select the **CellType** property and choose the **Currency**, **Number**, **Percent**, **Progress**, or **Slider** cell type.
8. Expand the list of properties under the **CellType** property. Select and set the **MaximumValue** and **MinimumValue** properties to the highest and lowest allowable currency values.
9. Click **OK** to close the **Cell, Column, and Row Editor**.
10. Click **OK** to close the **SheetView Collection Editor**.

Using Code

1. Set the minimum or maximum value or both for a currency cell by setting the **MaximumValue** and **MinimumValue** properties for a **CurrencyCellType** ('**CurrencyCellType Class**' in the on-line documentation), **NumberCellType** ('**NumberCellType Class**' in the on-line documentation), **PercentCellType** ('**PercentCellType Class**' in the on-line documentation), **ProgressCellType** ('**ProgressCellType Class**' in the on-line documentation), or **SliderCellType** ('**SliderCellType Class**' in the on-line documentation) object.
2. Assign the cell type to a cell or range of cells by setting the **CellType** property for a cell, column, row, or style to the particular cell type object.

Example

This example creates a currency cell and sets the minimum and maximum values.

C#

```
FarPoint.Win.Spread.CellType.CurrencyCellType currcell = new
FarPoint.Win.Spread.CellType.CurrencyCellType();
currcell.MinimumValue = 1;
currcell.MaximumValue = 10;
fpSpread1.ActiveSheet.Cells[1,1].CellType = currcell;
fpSpread1.ActiveSheet.Cells[1,1].Note = "Pick a number between 1 and 10!";
```

VB

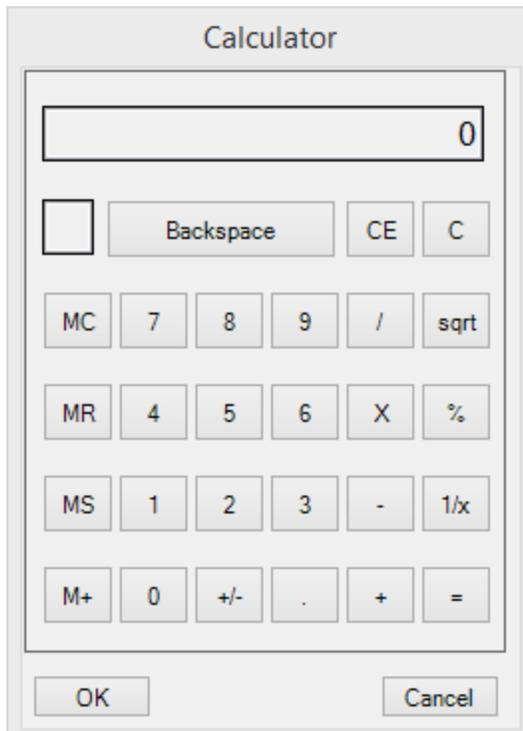
```
Dim currcell As New FarPoint.Win.Spread.CellType.CurrencyCellType()
currcell.MinimumValue = 1
currcell.MaximumValue = 10
fpSpread1.ActiveSheet.Cells(1,1).CellType = currcell
fpSpread1.ActiveSheet.Cells(1,1).Note = "Pick a number between 1 and 10!"
```

Using the Spread Designer

1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType** (or right-click on the cells and select the cell type). From the drop-down list, choose **Currency** or other numeric cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type.
3. Select and set the **MaximumValue** and **MinimumValue** properties to the highest and lowest allowable currency values.
4. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Customizing the Pop-Up Calculator Control

If the user presses F4 or double-clicks any of the several numeric cell types when the cell is in edit mode, a pop-up calculator appears, as shown in the following figure.



You can use the calculator to type a number or to perform a calculation. The result from the calculator is placed in the numeric cell. The cell types that allow a calculator pop-up are:

- **CurrencyCellType** ('**CurrencyCellType Class**' in the on-line documentation) (see **Setting a Currency Cell**)
- **NumberCellType** ('**NumberCellType Class**' in the on-line documentation) (see **Setting a Number Cell**)
- **PercentCellType** ('**PercentCellType Class**' in the on-line documentation) (see **Setting a Percent Cell**)

When using the controls, you must click the **OK** or **Cancel** button to close the control. You can change the text of the buttons at the bottom with the **SetCalculatorText** method for that cell type.

Note that the text appears centered on the button. If you set custom text for the buttons, try to limit your text to eight or nine characters in length. The button will display ten characters, but the first and last appear very close to the edges of the button.

For information on the editable cell types, refer to **Working with Editable Cell Types**.

For information on other features of cell types, refer to **Understanding Additional Features of Cell Types**.

Using Code

1. Define the cell by creating an instance of the **CurrencyCellType** ('**CurrencyCellType Class**' in the on-line documentation) class (or other numeric type cell).
2. Specify the text for the buttons.
3. Assign the cell type to a cell or range of cells by setting the **CellType** property for a cell, column, row, or style.

Example

This example sets the text of the buttons.

C#

```
FarPoint.Win.Spread.CellType.CurrencyCellType ctest = new
FarPoint.Win.Spread.CellType.CurrencyCellType();
ctest.SetCalculatorText("Accept", "Cancel");
fpSpread1.Sheets[0].Cells[0, 0].CellType = ctest;
```

VB

```
Dim ctest As New FarPoint.Win.Spread.CellType.CurrencyCellType()
ctest.SetCalculatorText("Accept", "Cancel")
fpSpread1.Sheets(0).Cells(0, 0).CellType = ctest
```

Customizing Automatic Completion (Type Ahead)

You can provide automatic completion (type ahead) of user input to a cell. You use the **IAutoCompleteSupport** ('**IAutoCompleteSupport Interface**' in the on-line documentation) interface and its properties to provide the automatic completion feature in the editable cell types.

	CompanyName	ContactName
1	Alfreds Futterkiste	Maria Anders
2	Ana Trujillo Emparedados y helados	Ana Trujillo
3	Antonio Moreno Taquería	Antonio Moreno
4	benjamin	Thomas Hardy
5	Benjamin	Christina Berglund
6	Berglunds snabbköp	Hanna Moos
7	Blauer See Delikatessen	Frédérique Citeaux
8	Blondesddsl père et fils	Martin Sommer
9	Bólido Comidas preparadas Bon app'	Laurence Lebihan
10	Bottom-Dollar Markets	Elizabeth Lincoln
11	Bottom-Dollar Markets	Victoria Ashworth
12	B's Beverages	Patricio Simpson
13	Cactus Comidas para llevar	Francisco Chang

Automatic completion modes

Append – Appends the remainder of the most likely candidate

Suggest – Displays a drop-down list of suggested candidates

Basically there are two properties that are set. First you set the mode of automatic completion, as shown in the figure above. The options include whether to suggest a list of possible completions or a drop-down list of possible completions or both (or none).

Second, you can set the source of the suggestions and drop-down list. The source is the list of items that are considered for completion. You can create a custom source and define your own list of items or you can set various system sources. There are two properties in the interface that provide settings for the custom source. The first one sets the list of possible candidates for a custom source. The second sets whether to fill the list with the list of values from other cells in the column. To use the values in the cells in the column, for example, you would set the source to custom and then turn on automatic fill. The automatic fill only adds items to the custom source if they are above or below the cell without a blank cell in between.

For an example of automatic completion, refer to the web site at <https://developer.mescius.com/spreadnet>.

For information on other features of cell types, refer to **Understanding Additional Features of Cell Types**.

Working with a SubEditor

For several editable cell types (the ones in **Working with Editable Cell Types**), when you click inside the cell, an editor is displayed. You can go beyond this simple line editor and provide a custom user interface (to provide options to make the task of user input easier). This other level of interface is controlled by the subeditor, or the editor within the cell editor. For example, when you select the date-time cell, you can provide a calendar for the user to select a date. This calendar control would be called by the subeditor.

Creating a Subeditor

You can create your own subeditor, which can be displayed by the following actions:

- by pressing F4 key
- by double-clicking cells in edit mode
- by pressing a drop-down button (when **DropDownButton** property is set to true)

The steps for creating your own subeditor are:

1. Create a new **Form** class for a subeditor.
2. Implement **ISubEditor** interface to the Form that you have just created.
3. Set the subeditor (**SubEditor** property) to the caller.

To see an example of a subeditor, refer to **Customizing the Pop-Up Date-Time Control** where the calendar subeditor is available for a date-time cell, and **Customizing the Pop-Up Calculator Control** where a calculator subeditor is available for several numeric cell types.

Canceling a Subeditor

For several editable cell types, when you click inside the cell, a subeditor is displayed by default. But sometimes you might want to disable these subeditors. For example, in date-time cells you might want to disable the pop-up calendar control; in the number cells, you might want to disable the pop-up calculator control.

Example

To cancel subeditors, you can set `e.Cancel` to `True` in the **SubEditorOpening** ('SubEditorOpening Event' in the **on-line documentation**) event, as shown in the following example.

C#

```
private void fpSpread1_SubEditorOpening(object sender,
FarPoint.Win.Spread.SubEditorOpeningEventArgs e)
{
    e.Cancel = true;
}
```

VB

```
Private Sub fpSpread1_SubEditorOpening(ByVal sender As Object, ByVal e As
FarPoint.Win.Spread.SubEditorOpeningEventArgs) Handles FpSpread1.SubEditorOpening
    e.Cancel = True
End Sub
```

For information on the editable cell types, refer to **Working with Editable Cell Types**.

For information on other features of cell types, refer to **Understanding Additional Features of Cell Types**.

Creating a Custom Cell Type

You can implement your own types of cells by creating a subclass that inherits an existing cell type (that is, overriding each method in that class). The custom cell type class should be marked as serializable if the Clipboard will be used with it or if the export to Excel methods are used.

or information on other features of cell types, refer to **Understanding Additional Features of Cell Types**.

Using Code

This example subclasses the check box cell type to illustrate the use of the methods.

C#

```
class myCkBox : FarPoint.Win.Spread.CellType.CheckBoxCellType
{
    CheckBox ckbx = new CheckBox();

    public myCkBox()
    {
    }

    new event EventHandler EditingCanceled;
    new event EventHandler EditingStopped;
```

```

public override void StartEditing(EventArgs e, bool selectAll, bool
autoClipboard)
{
    return;
}
public override void CancelEditing()
{
    EditingCanceled(ckbx, EventArgs.Empty);
    base.FireEditingCanceled();
}
public override bool StopEditing()
{
    if (EditingStopped != null)
    {
        EditingStopped(ckbx, EventArgs.Empty);
        base.FireEditingStopped();
        return true;
    }
    else
    {
        return false;
    }
}

public override bool IsReservedKey(KeyEventArgs e)
{
    return base.IsReservedKey(e);
}
public override object IsReservedLocation(Graphics g, int x, int y,
Rectangle r, FarPoint.Win.Spread.Appearance appr, object
value, float zoom)
{
    return base.IsReservedLocation(g, x, y, r, appr, value, zoom);
}
public override Size GetPreferredSize(Graphics g, Size size,
FarPoint.Win.Spread.Appearance appr, object value, float zoom)
{
    return base.GetPreferredSize(g, size, appr, value, zoom);
}
public override object Parse(string s)
{
    return base.Parse(s);
}
public override string Format(object o)
{
    return base.Format(o.ToString());
}
public override Control GetEditorControl(FarPoint.Win.Spread.Appearance
appearance, float zoomFactor)
{
    return ckbx;
}
public override object GetEditorValue()
{
    return ckbx.CheckState;
}

```

```

        public override void PaintCell(Graphics g, Rectangle r,
FarPoint.Win.Spread.Appearance appr, object value, bool issel, bool
        islocked, float zoom)
        {
            GetEditorValue();
            if (ckbx.CheckState == CheckState.Checked)
            {
                ControlPaint.DrawCheckBox(g, r.X, r.Y, r.Width - 40, r.Height -
6, ButtonState.Checked);
            }
            else if (ckbx.CheckState == CheckState.Unchecked)
            {
                ControlPaint.DrawCheckBox(g, r.X, r.Y, r.Width - 40, r.Height - 6,
ButtonState.Normal);
            }
        }
        public override void SetEditorValue(object value)
        {
            ckbx.CheckState = CheckState.Checked;
        }
        public override Cursor GetReservedCursor(object o)
        {
            return base.GetReservedCursor(o);
        }
    }

    private void menuItem4_Click(object sender, System.EventArgs e)
    {
        myCkBox ckbx = new myCkBox();
        fpSpread1.ActiveSheet.Cells[0, 0].CellType = ckbx;
    }

```

Visual Basic

```

Public Class myCkBox
    Inherits FarPoint.Win.Spread.CellType.CheckBoxCellType

    Dim ckbx As New CheckBox()

    Sub New()

    End Sub

    Public Shadows Event EditingStopped(ByVal sender As Object, ByVal e As
EventArgs)

    Public Shadows Event EditingCancelled(ByVal sender As Object, ByVal e
As EventArgs)

    Public Overrides Sub StartEditing(ByVal e As EventArgs, ByVal selectAll
As Boolean, ByVal autoClipboard As Boolean)
        MyBase.StartEditing(e, selectAll, autoClipboard)
    End Sub

    Public Overrides Sub CancelEditing()
        RaiseEvent EditingCancelled(ckbx, EventArgs.Empty)
        MyBase.FireEditingCanceled()
    End Sub

```

```

End Sub

Public Overrides Function StopEditing() As Boolean
    RaiseEvent EditingStopped(ckbx, EventArgs.Empty)
    MyBase.FireEditingStopped()
    Return True
End Function

Public Overrides Function IsReservedKey(ByVal e As KeyEventArgs) As
Boolean
    Return MyBase.IsReservedKey(e)
End Function

Public Overrides Function IsReservedLocation(ByVal g As Graphics, ByVal
x As Integer, ByVal y As Integer, ByVal r As Rectangle,
ByVal appr As FarPoint.Win.Spread.Appearance, ByVal value As Object,
ByVal zoom As Single) As Object
    Return MyBase.IsReservedLocation(g, x, y, r, appr, value, zoom)
End Function

Public Overrides Function GetPreferredSize(ByVal g As Graphics, ByVal s
As Size, ByVal appr As FarPoint.Win.Spread.Appearance,
ByVal value As Object, ByVal zoom As Single) As Size
    Return MyBase.GetPreferredSize(g, s, appr, value, zoom)
End Function

Public Overrides Function Parse(ByVal s As String) As Object
    Return MyBase.Parse(s)
End Function

Public Overrides Function Format(ByVal o As Object) As String
    Return MyBase.Format(o)
End Function

Public Overrides Function GetEditorControl(ByVal appr As
FarPoint.Win.Spread.Appearance, ByVal zoom As Single) As Control
    Return ckbx
End Function

Public Overrides Function GetEditorValue() As Object
    Return ckbx.CheckState
End Function

Public Overrides Sub PaintCell(ByVal g As Graphics, ByVal r As
Rectangle, ByVal appr As FarPoint.Win.Spread.Appearance, ByVal
Value As Object, ByVal issel As Boolean, ByVal islocked As Boolean,
ByVal zoom As Single)
    GetEditorValue()
    If ckbx.CheckState = CheckState.Checked Then
        ControlPaint.DrawCheckBox(g, r.X, r.Y, r.Width - 40, r.Height -
6, ButtonState.Checked)
    ElseIf ckbx.CheckState = CheckState.Unchecked Then
        ControlPaint.DrawCheckBox(g, r.X, r.Y, r.Width - 40, r.Height -
6, ButtonState.Normal)
    End If
End Sub

```

```
Public Overrides Sub SetEditorValue(ByVal value As Object)
    ckbx.CheckState = CheckState.Checked
End Sub

Public Overrides Function GetReservedCursor(ByVal o As Object) As
Cursor
    Return MyBase.GetReservedCursor(o)
End Function
End Class

Private Sub MenuItem1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MenuItem1.Click
    Dim ckbx As New myCkBox()
    fpSpread1.ActiveSheet.Cells(0, 0).CellType = ckbx
End Sub
```

Data Binding

You can bind the Spread component to a data set, such as data in a database, or to anything that the .NET framework allows, such as an `IList` object. You can work with data binding Spread with data or other controls by these tasks:

- **Binding to Data**
 - **Binding Spread to an External Data Set**
 - **Binding a Cell Range in Spread to an External Data Source**
 - **Binding a Cell Range in Spread as a Data Source to an External Control**
 - **Binding a Combo Box to a DataReader**
- **Customizing Data Binding**
 - **Adding an Unbound Column to a Bound Sheet**
 - **Customizing Column and Field Binding**
 - **Customizing Cell Types for Bound Sheets**
 - **Customizing Column Headers for Bound Sheets**
- **Adding to Bound Data**
 - **Adding a Row to a Bound Sheet**
 - **Adding an Unbound Row to a Bound Sheet**
- **Working with Hierarchical Data Display**
 - **Creating a Hierarchical Display Manually**
 - **Creating Custom Hierarchy Icons**

Binding to Data

You can bind the Spread component to a data set, such as data in a database, or to anything that the .NET framework allows, such as an `IList` object.

The tasks you can perform to bind Spread to data or other controls are:

- **Binding Spread to an External Data Set**
- **Binding a Cell Range in Spread to an External Data Source**
- **Binding a Cell Range in Spread as a Data Source to an External Control**
- **Binding a Combo Box to a DataReader**

Binding Spread to an External Data Set

You can bind the Spread component to a data set. When you bind the component using the default settings, data from the data set is read into the columns and rows of the sheet to which you bind the data. Columns are associated with fields, and rows represent each record in the data set.

You can also bind a sheet, column, or cell range.

The following instructions provide the code necessary to bind the Spread component to a data set. For a tutorial that can introduce you to the procedure for data binding, refer to the **Tutorial: Binding to a Corporate Database (Visual Studio 2013 or later)**.

For more information, refer to the **AddRowToDataSource** ('**AddRowToDataSource Method**' in the **on-line documentation**) and **BindDataColumn** ('**BindDataColumn Method**' in the **on-line documentation**) methods.

If you want to re-order the columns in a data set, move around the columns in the Spread after the bind. Use the **BindDataColumn** ('**BindDataColumn Method**' in the **on-line documentation**) method in the **SheetView** ('**SheetView Class**' in the **on-line documentation**) class to do this.

There are many alternative ways to set up data binding. To learn more about data binding in Visual Studio .NET, consult the Visual Studio .NET documentation.

Using the Properties Window

1. Create your data set.
2. In the **Properties** window select the Spread component.
 - To set the data source for the component, set the **DataSource** property.
 - To set the data source for the sheet,
 - a. Select the **Sheets** property for the Spread component.
 - b. Click the button to display the **SheetView Collection Editor**.
 - c. Select the sheet for which to set the data source.
 - d. Set the **DataSource** property for that sheet.

Using Code

1. Create your data set.
2. Set the FpSpread **DataSource** (**'DataSource Property' in the on-line documentation**) or SheetView **DataSource** (**'DataSource Property' in the on-line documentation**) property equal to the data set.

Example

This example code binds the Spread component to a data set named "dbDataSet".

C#

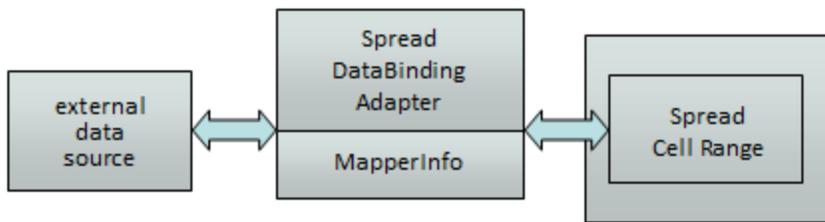
```
// Bind the component to the data set.
fpSpread1.Sheets[0].AutoGenerateColumns = false;
fpSpread1.Sheets[0].DataSource = dbDataSet;
fpSpread1.Sheets[0].ColumnCount = 2;
fpSpread1.Sheets[0].BindDataColumn(0, "ID");
fpSpread1.Sheets[0].BindDataColumn(1, "Description");
```

VB

```
' Bind the component to the data set.
fpSpread1.Sheets(0).AutoGenerateColumns = False
fpSpread1.Sheets(0).DataSource = dbDataSet
fpSpread1.Sheets(0).ColumnCount = 2
fpSpread1.Sheets(0).BindDataColumn(0, "ID")
fpSpread1.Sheets(0).BindDataColumn(1, "Description")
```

Binding a Cell Range in Spread to an External Data Source

You can bind a cell range in Spread to an external data source. To do this, use the **SpreadDataBindingAdpater** (**'SpreadDataBindingAdapter Class' in the on-line documentation**) class to create a connection between the Spread component and the data source and use the **MapperInfo** (**'MapperInfo Class' in the on-line documentation**) class to map the cell range to the range in the data source.



If you add or remove a column from the data source when bound to a cell range, the Spread component does not automatically update.

The data source and the cell range in the Spread are controlled by the **MapperInfo** ('**MapperInfo Class**' in the **on-line documentation**) class. They synchronize with each other via row synchronization. If the user adds or removes any row in the cell range, it will affect the data source and vice versa. If the user adds a new row right below the existing bound cell range, the cell range will expand one row and make the **MapperInfo** ('**MapperInfo Class**' in the **on-line documentation**) class and data source expand and vice versa. If the new row is added outside of the bound area, then it is not added to the bound range.

By default the Spread component tries to match the data type of the external data source to the cell type of Spread. You can set **DataAutoCellTypes** ('**DataAutoCellTypes Property**' in the **on-line documentation**) to False to prevent this. The following table shows the default cell type that is used based on the type of data.

Data Type

Boolean
 DateTime
 Double, Single, Decimal
 Int16, Int32, and so on
 String
 Other

Cell Type

Check box cell
 Date time cell
 Number cell
 Number cell
 Text cell
 General cell

For more information, refer to the **SpreadDataBindingAdapter** ('**SpreadDataBindingAdapter Class**' in the **on-line documentation**) class and the **MapperInfo** ('**MapperInfo Class**' in the **on-line documentation**) class in the API reference.

Using Code

1. Create your data set.
2. Create a new **SpreadDataBindingAdapter** object.
3. Set the **Spread** object to the adapter.
4. Set the sheet name.
5. Create the **MapperInfo** object and link it to the adapter.
6. Set the **FillSpreadDataByDataSource** method.

Example

This example code binds a single cell range to a data source. The example requires that you create a datasource object named "dt".

C#

```

FarPoint.Win.Spread.Data.SpreadDataBindingAdapter data = new
FarPoint.Win.Spread.Data.SpreadDataBindingAdapter();
// Assign the datasource to a data table
  
```

```

data.DataSource = dt;
data.Spread = fpSpread1;
data.SheetName = "Sheet1";
data.MapperInfo = new FarPoint.Win.Spread.Data.MapperInfo(3, 2, 1, 1);
data.FillSpreadDataByDataSource();

```

VB

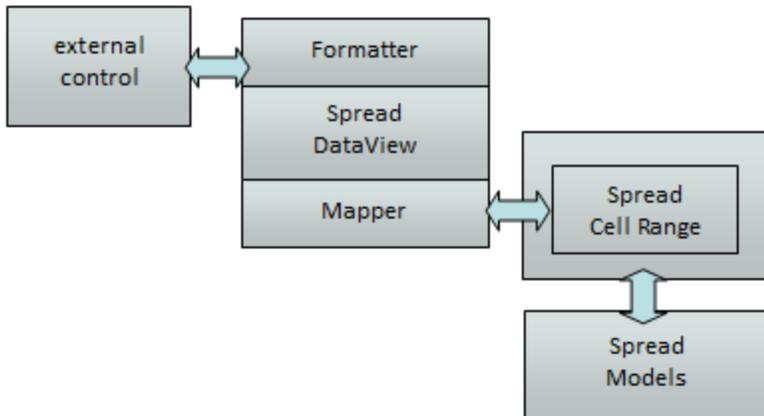
```

' Create an adapter.
Dim data As New FarPoint.Win.Spread.Data.SpreadDataBindingAdapter
' Assign the datasource to a data table
data.DataSource = dt
data.Spread = fpSpread1
data.SheetName = "Sheet1"
data.MapperInfo = New FarPoint.Win.Spread.Data.MapperInfo(3, 2, 1, 1)
data.FillSpreadDataByDataSource()

```

Binding a Cell Range in Spread as a Data Source to an External Control

You can bind a range of cells in Spread as a data source for an external control such as a **DataGrid** control. The following diagram shows the objects involved.



These are the objects involved:

- **SpreadDataView** ('SpreadDataView Class' in the on-line documentation)
- **SpreadDataRowView** ('SpreadDataRowView Class' in the on-line documentation)
- **ISpreadDataViewDataFormatter** ('ISpreadDataViewDataFormatter Interface' in the on-line documentation)
- **ISpreadDataViewMapper** ('ISpreadDataViewMapper Interface' in the on-line documentation)
- **DefaultSpreadDataViewDataFormatter** ('DefaultSpreadDataViewDataFormatter Class' in the on-line documentation)
- **DefaultSpreadDataViewMapper** ('DefaultSpreadDataViewMapper Class' in the on-line documentation)

Creating a Custom Formatter

You can create a custom formatter class by inheriting from **ISpreadDataViewDataFormatter** ('ISpreadDataViewDataFormatter Interface' in the on-line documentation).

Using Code

1. Create the class.
2. Assign the class to the data view.

Example

This example code creates a custom class.

C#

```
public class MySpreadDataViewDataFormatter : ISpreadDataViewDataFormatter
{
    private SpreadDataColumn column;
    private SheetView sheetView;
    public SheetView SheetView;
    {
    get { return sheetView; }
    set { sheetView = value; }
    }
    public MySpreadDataViewDataFormatter(SpreadDataColumn ownerColumn, SheetView sheetView)
    {
    if (ownerColumn == null)
    {
    throw new ArgumentNullException("ownerColumn");
    }
    column = ownerColumn;
    this.SheetView = sheetView;
    }
    public object GetCellValue(Cell cell)
    {
    object ret = null;
    try
    {
    ret = this.SheetView.GetValue(cell.Row.Index, cell.Column.Index);
    ret += ": Customized format";
    }
    catch
    {
    ret = " No value";
    }
    return ret;
    }
    public void SetCellValue(Cell cell, object value)
    {
    this.SheetView.SetValue(cell.Row.Index, cell.Column.Index, value + ": Customized
format");
    }
    }
    // Assign new formatter
    dataSet = BuildDataSet(5,5);
    this.spreadDataBindingAdapter1.Spread = this.fpSpread1;
    this.spreadDataBindingAdapter1.SheetName = this.fpSpread1.ActiveSheet.SheetName;
    this.spreadDataBindingAdapter1.DataSource = dataSet.Tables[0];
    spreadDataBindingAdapter1.MapperInfo = new MapperInfo(1, 2, 3, 4);
    MySpreadDataViewDataFormatter testFormatter = new MySpreadDataViewDataFormatter
    (this.spreadDataBindingAdapter1.SpreadDataView.Columns[2], fpSpread1.ActiveSheet);
    this.spreadDataBindingAdapter1.SpreadDataView.Columns[2].Formatter = testFormatter;
```

```
this.spreadDataBindingAdapter1.FillSpreadDataByDataSource();
```

Creating a Custom Mapper

You can create a custom mapper class by inheriting from **ISpreadDataViewMapper** (**'ISpreadDataViewMapper Interface' in the on-line documentation**).

Using Code

1. Create the class.
2. Assign the class to the data view.

Example

This example code creates a custom class.

C#

```
public class MySpreadDataViewMapper : ISpreadDataViewMapper
{
    ...
}
//Assign customized Mapper for SpreadDataView
dataSet = BuildDataSet(5,5);
this.spreadDataBindingAdapter1.Spread = this.fpSpread1;
this.spreadDataBindingAdapter1.SheetName = this.fpSpread1.ActiveSheet.SheetName;
this.spreadDataBindingAdapter1.DataSource = dataSet.Tables[0];
MySpreadDataViewMapper testMapper = new MySpreadDataViewMapper ();
this.spreadDataBindingAdapter1.SpreadDataView.Mapper = testMapper;
spreadDataBindingAdapter1.MapperInfo = new MapperInfo(1, 2, 3, 4);
this.spreadDataBindingAdapter1.FillSpreadDataByDataSource();
```

VB

```
Public Class MySpreadDataViewMapper
Implements FarPoint.Win.Spread.Data.ISpreadDataViewMapper
...
End Class
dataSet = BuildDataSet(5, 5)
Me.spreadDataBindingAdapter1.Spread = Me.fpSpread1
Me.spreadDataBindingAdapter1.SheetName = Me.fpSpread1.ActiveSheet.SheetName
Me.spreadDataBindingAdapter1.DataSource = dataSet.Tables(0)
Dim testMapper As New MySpreadDataViewMapper()
Me.spreadDataBindingAdapter1.SpreadDataView.Mapper = testMapper
spreadDataBindingAdapter1.MapperInfo = New MapperInfo(1, 2, 3, 4)
Me.spreadDataBindingAdapter1.FillSpreadDataByDataSource()
```

Binding a Combo Box to a DataReader

You can bind a combo box to a **DataReader**. A **DataReader** lets you access data in a read-only, forward-only way from a data source. Using a **DataReader** is often a quick way of returning results, as illustrated in the following example for populating results in a combo box.

Example

This example adds data to a combo cell from a data source.

C#

```

/// Set up a connection to the database. The database name is an example.
string dbpath = "c:\\reader.mdb";
System.Data.OleDb.OleDbConnection dbConn = new
System.Data.OleDb.OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;DATA SOURCE=" +
dbpath);
/// Open a connection and a SELECT command.
dbConn.Open();
System.Data.OleDb.OleDbCommand dbCommand = new System.Data.OleDb.OleDbCommand("SELECT *
FROM Table1", dbConn);
/// Open a DataReader on that connection and command.
System.Data.OleDb.OleDbDataReader dr =
dbCommand.ExecuteReader(System.Data.CommandBehavior.CloseConnection);
/// Loop through the rows returned by the reader.
ArrayList al = new ArrayList();
while (dr.Read()) {
    al.Add(dr("Data"));
}
/// Populate combo box with data converted to strings.
string[] s;
s = al.ToArray(typeof(string));
FarPoint.Win.Spread.ComboBoxCellType cb = new FarPoint.Win.Spread.ComboBoxCellType();
cb.Items = s;
fpSpread1.ActiveSheetView.Cells(0, 0).CellType = cb;
/// Dispose connection.
dbConn.Dispose();

```

VB

```

' Set up a connection to the database. The database name is an example.
Dim dbpath As String = "c:\reader.mdb"
Dim dbConn As New System.Data.OleDb.OleDbConnection( _
"Provider=Microsoft.Jet.OLEDB.4.0;DATA SOURCE=" & dbpath)
' Open a connection and a SELECT command.
dbConn.Open()
Dim dbCommand As New System.Data.OleDb.OleDbCommand( _
"SELECT * FROM Table1", dbConn)
' Open a DataReader on that connection and command.
Dim dr As System.Data.OleDb.OleDbDataReader =
dbCommand.ExecuteReader(System.Data.CommandBehavior.CloseConnection)
' Loop through the rows returned by the reader.
Dim al As New ArrayList()
While dr.Read()
    al.Add(dr("Data"))
End While
' Populate combo box with data converted to strings.
Dim s As String()
s = al.ToArray(GetType(String))
Dim cb As New FarPoint.Win.Spread.ComboBoxCellType
cb.Items = s
fpSpread1.ActiveSheetView.Cells(0, 0).CellType = cb
' Dispose connection.
dbConn.Dispose()

```

Customizing Data Binding

Adding an Unbound Column to a Bound Sheet

Once you bind a sheet to a data set you might want to add an unbound column to contain additional data.

You can add an unbound column to the Spread control by increasing the **ColumnCount** (**'ColumnCount Property' in the on-line documentation**) property after the control is bound. Another option is to use the **DataField** (**'DataField Property' in the on-line documentation**) property to bind specific columns and leave the **DataField** (**'DataField Property' in the on-line documentation**) property unset for the columns you do not want to bind.

Using Code

1. Create your data set.
2. Set the **FpSpread** object's or **Sheet** object's **DataSource** (**'DataSource Property' in the on-line documentation**) property equal to the data set.
3. Set the **ColumnCount** (**'ColumnCount Property' in the on-line documentation**) property for the sheet.

Example

This example code binds the Spread component to a data set and then sets the column count.

C#

```
// Bind the component to the data set.
fpSpread1.DataSource = dbDataSet;
// If this datasource has 19 columns, set the count to 20.
fpSpread1.Sheets[0].ColumnCount=20;
```

VB

```
' Bind the component to the data set.
fpSpread1.DataSource = dbDataSet
' If this datasource has 19 columns, set the count to 20.
fpSpread1.Sheets(0).ColumnCount=20
```

Customizing Column and Field Binding

When a sheet is bound to a data set, columns are assigned to data set fields sequentially. That is, the first data field is assigned to column A, the second to column B, and so on. You can change the assignments to assign any field to any column.

For bound spreadsheets, by default the spreadsheet inherits the width of the columns from the database. To determine your own custom widths, you would either need to set your width after binding the Spread or set **DataAutoSizeColumns** (**'DataAutoSizeColumns Property' in the on-line documentation**) to False and set your width.

If you have more than one Spread bound to a single data set, you may want to set the **AutoGenerateColumns** (**'AutoGenerateColumns Property' in the on-line documentation**) property of the sheet in each Spread to False, so Spread does not bind all the columns. Then you can set the **DataField** (**'DataField Property' in the on-line documentation**) property of the columns in each of the Spread controls to the field name from the data set. Then, only that column of the data set is bound to the Spread.

Using the Properties Window

1. Create your data set.
2. In the **Properties** window, select the Spread component.

3. Select the **DataSource** property (for the Spread component) or select the **Sheets** property and set the **DataSource** property, and set it equal to the data set.
4. If you have not already done so, select the **Sheets** property for the Spread component.
5. Click the button to display the **SheetView Collection Editor**.
6. Click the sheet for which you want to change the column fields.
7. Set the sheet's **AutoGenerateColumns** property to **False**, because you want to override the auto-generated column and field mappings.
8. In the property list, select the **Cells** property and click the button to display the **Cell, Column, and Row Editor**.
9. Select the column for which you want to change the column fields.
10. Set the **DataField** property to select one of the available fields, or leave it blank to make the column an unbound column.
11. Click **OK** to close the **Cell, Column, and Row Editor**.
12. Click **OK** to close the **SheetView Collection Editor**.

Using a Shortcut

1. Create your data set.
2. Set the Sheet **AutoGenerateColumn** property to false, because you want to override the auto-generated column and field mappings.
3. Set the **FpSpread** object's or **Sheet** object's **DataSource** property equal to the data set.
4. For each column in the sheet for which you want to map a field, set the **Column** object's **DataField** property to the field name in the data set.

Example

This example code binds the Spread component to an existing data set, then sets the fields to use in the first four columns.

C#

```
// Turn off automatic column and field mapping.
fpSpread1.Sheets[0].AutoGenerateColumns = false;
// Bind the component to the data set.
fpSpread1.DataSource = dataSet1;
// Set the fields for the columns.
fpSpread1.Sheets[0].Columns[0].DataField = "Description";
fpSpread1.Sheets[0].Columns[1].DataField = "ID";
fpSpread1.Sheets[0].Columns[2].DataField = "LeadTime";
fpSpread1.Sheets[0].Columns[3].DataField = "Price";
```

VB

```
' Turn off automatic column and field mapping.
fpSpread1.Sheets(0).AutoGenerateColumns = False
' Bind the component to the data set.
fpSpread1.DataSource = DataSet1
' Set the fields for the columns.
fpSpread1.Sheets(0).Columns(0).DataField = "Description"
fpSpread1.Sheets(0).Columns(1).DataField = "ID"
fpSpread1.Sheets(0).Columns(2).DataField = "LeadTime"
fpSpread1.Sheets(0).Columns(3).DataField = "Price"
```

Using Code

1. Create your data set.
2. Create a new **SheetView** object.
3. Set the **SheetView** object's **AutoGenerateColumn** property to False, because you want to override the auto-generated column and field mappings.
4. Set the **SheetView** object's **DataSource** property equal to the data set.
5. For each column for which you want to map the fields, set the **SheetView** object's **Columns** object **DataField** property to specify the field.
6. Assign the **SheetView** object to a sheet in the component.

Example

This example code creates a bound **SheetView** object and maps four fields to four columns, then assigns it to a sheet in a Spread component.

C#

```
// Create a new SheetView object.
FarPoint.Win.Spread.SheetView newsheet = new FarPoint.Win.Spread.SheetView();
// Turn off automatic column and field mapping.
newsheet.AutoGenerateColumns = false;
// Bind the SheetView object to the data set.
newsheet.DataSource = dataSet1;
// Set the fields for the columns.
newsheet.Columns[0].DataField = "Description";
newsheet.Columns[1].DataField = "ID";
newsheet.Columns[2].DataField = "LeadTime";
newsheet.Columns[3].DataField = "Price";
// Assign the SheetView object to the first sheet.
fpSpread1.Sheets[0] = newsheet;
```

VB

```
' Create a new SheetView object.
Dim newsheet As New FarPoint.Win.Spread.SheetView()
' Turn off automatic column and field mapping.
newsheet.AutoGenerateColumns = False
' Bind the SheetView object to the data set.
newsheet.DataSource = DataSet1
' Set the fields for the columns.
newsheet.Columns(0).DataField = "Description"
newsheet.Columns(1).DataField = "ID"
newsheet.Columns(2).DataField = "LeadTime"
newsheet.Columns(3).DataField = "Price"
' Assign the SheetView object to the first sheet.
fpSpread1.Sheets(0) = newsheet
```

Using the Spread Designer

1. Create your data set.
2. Set the **FpSpread** object's or **Sheet** object's **DataSource** property equal to the data set.
3. Open the Spread Designer for the Spread component.
4. Select the sheet tab for the sheet for which you want to set the column fields.
5. Set the **AutoGenerateColumn** property to False, because you want to override the auto-generated column and field mappings.

6. Select the column for which you want to set the data field.
7. Set the **DataField** property to the field you want to display in the selected column.
8. Continue to select columns and set their **DataField** properties until you have set all the columns you want. You can leave some of the columns unbound if you want to do so.
9. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Customizing Cell Types for Bound Sheets

By default, when the Spread component or sheet is bound to a data set, it sets the cell types for the bound rows based on the data in the data set. You can turn off this automatic cell type assignment and assign cell types yourself.

Using the Properties Window

1. Create your data set.
2. In the **Properties** window, select the Spread component.
3. Select the **DataSource** property (for the Spread component) or select the **Sheets** property and set the **DataSource** property, and set it equal to the data set.
4. If you have not already done so, select the **Sheets** property for the Spread component.
5. Click the button to display the **SheetView Collection Editor**.
6. Click the sheet for which you want to change the cell types.
7. Set the **DataAutoCellTypes** property to **False** so the sheet does not automatically assign the cell types for the columns.
8. In the property list, select the **Columns** property and click the button to display the **Cell, Column, and Row Editor**.
9. Select the column for which you want to change the cell type.
10. Set the **CellType** property to the cell type you want to use for the column.
11. Repeat steps 8 and 9 for each column for which you want to set the cell type.
12. Click **OK** to close the **Cell, Column, and Row Editor**.
13. Click **OK** to close the **SheetView Collection Editor**.

Using a Shortcut

1. Create your data set.
2. Set the **FpSpread** object's or **Sheet** object's **DataSource** (**'DataSource Property' in the on-line documentation**) property equal to the data set.
3. Set the **Sheet** object's **DataAutoCellTypes** (**'DataAutoCellTypes Property' in the on-line documentation**) property to **False** so the sheet does not automatically assign the cell types for the columns.
4. Set the **Column** object's **CellType** (**'CellType Property' in the on-line documentation**) property to specify the cell type for each column.

Example

This example code binds the Spread component to a data set then assigns the cell types for its three columns.

C#

```
// Bind the component to the data set.
fpSpread1.DataSource = dbDataSet;
// Turn off automatic cell type assignment.
fpSpread1.Sheets[0].DataAutoCellTypes = false;
// Set the first column as general cell type.
FarPoint.Win.Spread.CellType.GeneralCellType generalct = new
```

```

FarPoint.Win.Spread.CellType.GeneralCellType();
fpSpread1.Sheets[0].Columns[0].CellType = generalct;
// Set the second column as number cell type.
FarPoint.Win.Spread.CellType.NumberCellType numberct = new
FarPoint.Win.Spread.CellType.NumberCellType();
fpSpread1.Sheets[0].Columns[1].CellType = numberct;
// Set the third column as currency cell type.
FarPoint.Win.Spread.CellType.CurrencyCellType currct = new
FarPoint.Win.Spread.CellType.CurrencyCellType();
fpSpread1.Sheets[0].Columns[2].CellType = currct;

```

VB

```

' Bind the component to the data set.
fpSpread1.DataSource = dbDataSet
' Turn off automatic cell type assignment.
fpSpread1.Sheets(0).DataAutoCellTypes = False
' Set the first column as general cell type.
Dim generalct As New FarPoint.Win.Spread.CellType.GeneralCellType()
fpSpread1.Sheets(0).Columns(0).CellType = generalct
' Set the second column as number cell type.
Dim numberct As New FarPoint.Win.Spread.CellType.NumberCellType()
fpSpread1.Sheets(0).Columns(1).CellType = numberct
' Set the third column as currency cell type.
Dim currct As New FarPoint.Win.Spread.CellType.CurrencyCellType()
fpSpread1.Sheets(0).Columns(2).CellType = currct

```

Using Code

1. Create your data set.
2. Create a new **SheetView** object.
3. Set the **SheetView** object's **DataSource** property equal to the data set.
4. If you are providing custom text for every column header, set the **SheetView** object's **DataAutoHeadings** property to **False** so the sheet does not display the field names from the data set as the column header text.
5. Set the **Text** property of the ColumnHeader cells to specify the custom text.
6. Assign the **SheetView** object to a sheet in the component.

Example

This example code creates a bound **SheetView** object and customizes the text in the first column header cell, then assigns it to a sheet in a Spread component.

C#

```

// Create a new SheetView object.
FarPoint.Win.Spread.SheetView newsheet = new FarPoint.Win.Spread.SheetView();
// Bind the SheetView object to the data set.
newsheet.DataSource = dataSet1;
// Change the column header text in the first header cell.
newsheet.ColumnHeader.Cells[0, 0].Text = "Student ID";
// Assign the SheetView object to the first sheet.
fpSpread1.Sheets[0] = newsheet;

```

VB

```

' Create a new SheetView object.

```

```
Dim newsheet As New FarPoint.Win.Spread.SheetView()  
' Bind the SheetView object to the data set.  
newsheet.DataSource = DataSet1  
' Change the column header text in the first header cell.  
newsheet.ColumnHeader.Cells(0, 0).Text = "Student ID"  
' Assign the SheetView object to the first sheet.  
fpSpread1.Sheets(0) = newsheet
```

Using the Spread Designer

1. Create your data set.
2. Set the **FpSpread** object's or **Sheet** object's **DataSource** property equal to the data set.
3. Open the Spread Designer for the Spread component.
4. Select the sheet tab for the sheet for which you want to set the cell types.
5. Set the **DataAutoCellTypes** property to **False** so the sheet does not automatically assign the cell types for the columns.
6. Select the column for which you want to set the cell type.
7. In the property list, set the **CellType** property to the cell type you want to use for the column.
8. Repeat steps 6 and 7 for each column for which you want to set the cell type.
9. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Customizing Column Headers for Bound Sheets

By default, sheets display the field names in the column headers when bound to a data set. If you prefer, you can change the text to display custom column names.

Using the Properties Window

1. Create your data set.
2. In the **Properties** window, select the Spread component.
3. Select the **DataSource** property (for the Spread component) or select the **Sheets** property and set the **DataSource** property, and set it equal to the data set.
4. If you have not already done so, select the **Sheets** property for the Spread component.
5. Click the button to display the **SheetView Collection Editor**.
6. Click the sheet for which you want to change the column headings.
7. If you are providing custom text for every column header, set the **DataAutoHeadings** property to **False** so the sheet does not display the field names from the data set as the column header text.
8. In the property list, select the **Cells** property and click the button to display the **Cell, Column, and Row Editor**.
9. Select the column for which you want to change the column headings.
10. Set the **Label** property to the new text you want in the heading.
11. Repeat steps 8 and 9 for each column for which you want to set the column heading.
12. Click **OK** to close the **Cell, Column, and Row Editor**.
13. Click **OK** to close the **SheetView Collection Editor**.

Using a Shortcut

1. Create your data set.
2. Set the **FpSpread** object's or **Sheet** object's **DataSource** property equal to the data set.
3. If you are providing custom text for every column header, set the **Sheet** object's **DataAutoHeadings** property

- to **False** so the sheet does not display the field names from the data set as the column header text.
- Set the **Text** property of the ColumnHeader cells to specify the custom text.

Example

This example code binds the Spread component to a data set then customizes the first column header.

C#

```
// Bind the component to the data set.
fpSpread1.DataSource = dbDataSet;
// Set custom text in the first column header.
fpSpread1.Sheets[0].ColumnHeader.Cells[0, 0].Text = "Student ID";
```

VB

```
' Bind the component to the data set.
fpSpread1.DataSource = dbDataSet
' Set custom text in the first column header.
fpSpread1.Sheets(0).ColumnHeader.Cells(0, 0).Text = "Student ID"
```

Using Code

- Create your data set.
- Create a new **SheetView** object.
- Set the **SheetView** object's **DataSource** property equal to the data set.
- If you are providing custom text for every column header, set the **SheetView** object's **DataAutoHeadings** property to **False** so the sheet does not display the field names from the data set as the column header text.
- Set the **Text** property for ColumnHeader cells to specify the custom text.
- Assign the **SheetView** object to a sheet in the component.

Example

This example code creates a bound **SheetView** object and customizes the text in the first column header cell, then assigns it to a sheet in a Spread component.

C#

```
// Create a new SheetView object.
FarPoint.Win.Spread.SheetView newsheet = new FarPoint.Win.Spread.SheetView();
// Bind the SheetView object to the data set.
newsheet.DataSource = dataSet1;
// Change the column header text in the first header cell.
newsheet.ColumnHeader.Cells[0, 0].Text = "Student ID";
// Assign the SheetView object to the first sheet.
fpSpread1.Sheets[0] = newsheet;
```

VB

```
' Create a new SheetView object.
Dim newsheet As New FarPoint.Win.Spread.SheetView()
' Bind the SheetView object to the data set.
newsheet.DataSource = DataSet1
' Change the column header text in the first header cell.
newsheet.ColumnHeader.Cells(0, 0).Text = "Student ID"
' Assign the SheetView object to the first sheet.
```

```
FpSpread1.Sheets(0) = newsheet
```

Using the Spread Designer

1. Create your data set.
2. Set the **FpSpread** object's or **Sheet** object's **DataSource** property equal to the data set.
3. Open the Spread Designer for the Spread component.
4. Select the sheet tab for the sheet for which you want to set the custom header text.
5. If you are providing custom text for every column header, set the **DataAutoHeadings** property to **False** so the sheet does not display the field names from the data set as the column header text.
6. Select the column for which you want to set custom header text, then right-click and choose **Headers**.
7. In the **Header Editor**, double-click the header for which you want to display custom text.
8. Edit the text to be the custom text you want, and then press **Enter** to stop editing.
9. When you have edited all the header text you want to edit, click **OK** to close the **Header Editor**, then click **Yes** to apply your changes to the selection.
10. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Adding to Bound Data

SpreadWinForms allows you to set an image in the background of the cells in the data area of the sheet.

Adding a Row to a Bound Sheet

Once you bind a sheet to a data set you might want to add an unbound row to contain additional data. The row could then be bound using the **AddRowToDataSource ('AddRowToDataSource Method' in the on-line documentation)** method.

The following figure shows a sheet in a Spread component that contains data from a data set and an unbound row at the bottom that calculates the averages.

	LastName	GPA (Single)	GPA (Double)
1	Shorter	4.12	4.12
2	Williams	2.00	2.00
3	Zacheius	3.62	3.62
4	Average	3.25	3.25

Using Code

1. Create your data set.
2. Set the **FpSpread** object's **DataSource** ('DataSource Property' in the on-line documentation) property equal to the data set.
3. Call the **SheetView** object's **AddUnboundRows** ('AddUnboundRows Method' in the on-line documentation) method to specify where to add the unbound row.
4. Set properties for the unbound row.
5. Set the **SheetView** object's **AddRowToDataSource** ('AddRowToDataSource Method' in the on-line documentation) method if you want to add the row to the data source.

Example

This example code creates a bound **FpSpread** object and adds an unbound row to it.

C#

```
DataSet ds = new DataSet();
DataTable emp = new DataTable("Name");
emp.Columns.Add("LastName");
emp.Columns.Add("GPA (Single)", typeof(decimal));
emp.Columns.Add("GPA (Double)", typeof(decimal));
emp.Rows.Add(new Object[] { "Shorter", "4.12", "4.12" });
emp.Rows.Add(new Object[] { "Williams", "2.00", "2.00" });
emp.Rows.Add(new Object[] { "Zacheius", "3.62", "3.62" });
ds.Tables.Add(emp);
fpSpread1.DataSource = ds;
fpSpread1.ActiveSheet.AddUnboundRows(3, 1);
fpSpread1.ActiveSheet.Cells[3, 0].Text = "Average";
fpSpread1.ActiveSheet.Cells[3, 1].Formula = "AVERAGE(B1:B3)";
fpSpread1.ActiveSheet.Cells[3, 2].Formula = "AVERAGE(C1:C3)";
//fpSpread1.ActiveSheet.AddRowToDataSource(3, true);
```

VB

```
Dim ds = New DataSet()
Dim emp As New DataTable("Name")
emp.Columns.Add("LastName")
emp.Columns.Add("GPA (Single)", GetType(Decimal))
emp.Columns.Add("GPA (Double)", GetType(Decimal))
emp.Rows.Add(New Object() {"Shorter", "4.12", "4.12"})
emp.Rows.Add(New Object() {"Williams", "2.00", "2.00"})
emp.Rows.Add(New Object() {"Zacheius", "3.62", "3.62"})
ds.Tables.Add(emp)
fpSpread1.DataSource = ds
fpSpread1.ActiveSheet.AddUnboundRows(3, 1)
fpSpread1.ActiveSheet.Cells(3, 0).Text = "Average"
fpSpread1.ActiveSheet.Cells(3, 1).Formula = "AVERAGE(B1:B3)"
fpSpread1.ActiveSheet.Cells(3, 2).Formula = "AVERAGE(C1:C3)"
'fpSpread1.ActiveSheet.AddRowToDataSource(3, True)
```

Adding an Unbound Row to a Bound Sheet

Once you bind a sheet to a data set you might want to add an unbound row to contain additional data.

The following figure shows a sheet in a Spread component that contains data from a data set and an unbound row at the bottom that calculates the averages.



	Name	GPA (Single)	GPA (Double)
1	Shorter	4.12	4.12
2	Williams	2.00	2.00
3	Zacheius	3.62	3.62
4	Average	3.25	3.25

Using a Shortcut

1. Create your data set.
2. Set the **FpSpread** object's or **Sheet** object's **DataSource** ('DataSource Property' in the on-line **documentation**) property equal to the data set.
3. Call the **Sheet** object's **AddUnboundRows** ('AddUnboundRows Method' in the on-line **documentation**) method to specify where to add the unbound row.
4. Set properties for the unbound row.

Example

This example code binds the Spread component to a data set then adds an unbound row.

C#

```
// Bind the component to the data set.
fpSpread1.DataSource = dbDataSet;
// Add an unbound row.
fpSpread1.Sheets[0].AddUnboundRows(20, 1);
```

VB

```
' Bind the component to the data set.
fpSpread1.DataSource = dbDataSet
' Add an unbound row.
fpSpread1.Sheets(0).AddUnboundRows(20, 1)
```

Using Code

1. Create your data set.
2. Create a new **SheetView** object.
3. Set the **SheetView** object's **DataSource** ('DataSource Property' in the on-line **documentation**) property equal to the data set.
4. Call the **SheetView** object's **AddUnboundRows** ('AddUnboundRows Method' in the on-line **documentation**) method to specify where to add the unbound row.
5. Set properties for the unbound row.
6. Assign the **SheetView** object to a sheet in the component.

Example

This example code creates a bound **SheetView** object and adds an unbound row to it, then assigns it to a sheet in a Spread component.

C#

```
// Create a new SheetView object.
FarPoint.Win.Spread.SheetView newsheet = new FarPoint.Win.Spread.SheetView();
// Bind the SheetView object to the data set.
newsheet.DataSource = dataSet1;
// Add an unbound row.
newsheet.AddUnboundRows(20, 1);
// Assign the SheetView object to the first sheet.
fpSpread1.Sheets[0] = newsheet;
```

VB

```
' Create a new SheetView object.
```

```
Dim newsheet As New FarPoint.Win.Spread.SheetView()
' Bind the SheetView object to the data set.
newsheet.DataSource = DataSet1
' Add an unbound row.
newsheet.AddUnboundRows(20, 1)
' Assign the SheetView object to the first sheet.
fpSpread1.Sheets(0) = newsheet
```

Working with Hierarchical Data Display

Sheets can display relational data, such as from a relational database, in hierarchical views. The following figure shows an example of such a hierarchical view, which uses the database provided for the tutorials. The user can expand or collapse the levels of the hierarchy by clicking on the expand and collapse hierarchy icons (plus and minus signs).

CategoryName					
1	+	dog food			
2	-	cat food			
		ProductName	ProductDescription	Unit	
1	+	Kitty Delight Organic 20 lb.	Organic food for adult cats		
2	+	Kitty Delight Organic 8 lb.	Organic food for adult cats		
3	+	Nature's Best Food for Cats 10 lb.	Premium food for cats		
4	+	Nature's Best Food for Cats 5 lb.	Premium food for cats		
3	+	dog treats			
4	+	cat treats			
5	+	toys			
6	+	health and care			

To set up hierarchical data display, you first create a data set to hold the relational data, then define the relations between the data, and finally, set the Spread component to display the data as you want. You can customize the cell type, the colors, the headers, and other aspects of the appearance of the child view.

You can bind to a hierarchical collection.

If you set a skin for a sheet, you must apply that skin to the parent sheet and all the child sheets. For more information about skins, refer to **Applying a Skin to a Sheet**.

You can set the display of the hierarchy, which Spread treats as child views of the overall parent sheet. You can get information about child views using these properties of the **SheetView** (**'SheetView Class' in the on-line documentation**) class.

- **ChildRelationCount** (**'ChildRelationCount Property' in the on-line documentation**)
- **GetChildDataModel** (**'GetChildDataModel Method' in the on-line documentation**)
- **GetChildRelation** (**'GetChildRelation Method' in the on-line documentation**)
- **GetChildSheets** (**'GetChildSheets Method' in the on-line documentation**)
- **GetChildView** (**'GetChildView Method' in the on-line documentation**)
- **GetChildVisible** (**'GetChildVisible Method' in the on-line documentation**)
- **ParentRelationName** (**'ParentRelationName Property' in the on-line documentation**)

You can catch when the end user is expanding or collapsing the child view. For more information, see the **Expand** (**'Expand Event' in the on-line documentation**) event and the **ChildViewCreated** (**'ChildViewCreated**

Event' in the on-line documentation) event. You can determine if the row is expandable using the **GetRowExpandable ('GetRowExpandable Method' in the on-line documentation)** and **SetRowExpandable ('SetRowExpandable Method' in the on-line documentation)** methods, and if the row is expanded using the **IsRowExpanded ('IsRowExpanded Method' in the on-line documentation)** method.

If you need to set properties on the child **SpreadView**, the best place to put code to do that is in the **ChildWorkbookCreated** event. That event fires when a child **SpreadView** has been created. The **ChildViewCreated** event fires after the child **SheetView** has been created, but the child **SpreadView** does not get created until afterward, and it does not get created unless the child sheet is visible in the component (so that the layout calculations are faster).

Rather than deleting child sheets from a parent sheet, when working with bound data, you would delete the relation in your data source to delete that child sheet from Spread.

The following sample code assumes you want to remove the first child sheet returned, in this example, `alist(0)`:

Code

```
Dim alist As ArrayList = fpSpread1.Sheets(0).GetChildSheets()
Dim sv As FarPoint.Win.Spread.SheetView = alist(0)
Dim ds As DataSet = CType(fpSpread1.DataSource, DataSet)
ds.Relations.Remove(sv.ParentRelationName)
```

For more information about printing a hierarchical sheet, refer to **Printing a Child View of a Hierarchical Display**.

Using a Shortcut

1. Create your data set.
2. Set up the data relations between the data coming from the data set, for example, between tables coming from a relational database.
3. Set the **DataSource** property of the **FpSpread** or the **SheetView** object equal to the data set.
4. Provide code in the **ChildViewCreated** event of the Spread component for displaying the parent and child views of the data.

Example

This example code binds the Spread component to a data set that contains multiple related tables from a database, and sets up the component to display hierarchical views. This code example uses the database provided for the tutorials (`databind.mdb`). If you performed the default installation, the database file is in `\Program Files\Mescius\Spread.NET 12\Docs\Windows Forms\TutorialFiles`.

VB

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load ' Call subroutines to set up data and format the Spread component
InitData() FormatSpread() End Sub Private Sub InitData() Dim con As New
OleDb.OleDbConnection() Dim cmd As New OleDb.OleDbCommand() Dim da As New
OleDb.OleDbDataAdapter() Dim ds As New DataSet() Dim dt As DataTable
con.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Program Files
(x86)\ComponentOne\Spread.NET 12\Docs\Windows Forms\TutorialFiles\databind.mdb"
con.Open() With cmd .Connection = con .CommandType = CommandType.TableDirect
.CommandText = "Categories" End With da.SelectCommand = cmd da.Fill(ds, "Categories")
cmd.CommandText = "Products" da.SelectCommand = cmd da.Fill(ds, "Products")
cmd.CommandText = "Inventory Transactions" da.SelectCommand = cmd da.Fill(ds,
"Inventory Transactions") ds.Relations.Add("Root",
ds.Tables("Categories").Columns("CategoryID"),
ds.Tables("Products").Columns("CategoryID")) ds.Relations.Add("Secondary",
```

```

ds.Tables("Products").Columns("ProductID"), ds.Tables("Inventory
Transactions").Columns("TransactionID")) fpSpread1.DataSource = ds End Sub Private Sub
FormatSpread() With fpSpread1.Sheets(0) .ColumnHeader.Rows(0).Height = 30
.Columns(0).Visible = False End With End Sub Private Sub
fpSpread1_ChildViewCreated(ByVal sender As Object, ByVal e As
FarPoint.Win.Spread.ChildViewCreatedEventArgs) Handles fpSpread1.ChildViewCreated Dim
dateType As New FarPoint.Win.Spread.CellType.DateTimeCellType() If
e.SheetView.ParentRelationName = "Root" Then With e.SheetView .DataAutoCellTypes =
False .DataAutoSizeColumns = False .ColumnHeader.Rows(0).Height = 30
.Columns(0).Visible = False .Columns(3).Visible = False .Columns(4).Visible = False
.Columns(1).Width = 200 .Columns(2).Width = 185 .Columns(6).Width = 85
.Columns(7).Width = 80 .Columns(8).Width = 80 .Columns(5).CellType = New
FarPoint.Win.Spread.CellType.CurrencyCellType() .Columns(7).CellType = New
FarPoint.Win.Spread.CellType.CheckBoxCellType() End With Else With e.SheetView
.DataAutoCellTypes = False .DataAutoSizeColumns = False .ColumnHeader.Rows(0).Height =
30 .Columns(0).Visible = False .Columns(2).Visible = False .Columns(3).Visible = False
.Columns(4).Visible = False .Columns(7).Visible = False .Columns(8).Visible = False
.Columns(9).Visible = False .Columns(1).Width = 100 .Columns(6).Width = 80
.Columns(5).CellType = New FarPoint.Win.Spread.CellType.CurrencyCellType()
dateType.DateTimeFormat = FarPoint.Win.Spread.CellType.DateTimeFormat.ShortDate
.Columns(1).CellType = dateType 'Add a total column .ColumnCount = .ColumnCount + 1
.ColumnHeader.Cells(0, .ColumnCount - 1).Value = "Total" .Columns(.ColumnCount -
1).CellType = New FarPoint.Win.Spread.CellType.CurrencyCellType() .Columns(.ColumnCount
- 1).Formula = "F1*G1" End With End If End Sub

```

Using a Shortcut

1. Create your data set.
2. Set up the data relations between the data coming from the data set, for example, between tables coming from a relational database.
3. Set the **FpSpread** object or the **Sheet** object **DataSource** property equal to the data set.
4. Provide code in the FpSpread **ChildViewCreated** event for displaying the parent and child views of the data.

Example

This example code binds the Spread component to a hierarchical collection.

C#

```

public class Score
{
private string classname;
private string grade;
public string ClassName
{
get { return classname; }
set { classname = value; }
}
public string Grade
{
get { return grade; }
set { grade = value; }
}
}

public class Student

```

```
{
private string name;
private string id;
private ArrayList score = new ArrayList();
public string Name
{
get { return name; }
set { name = value; }
}
public string Id
{
get { return id; }
set { id = value; }
}
public ArrayList Score
{
get { return score; }
}
}

private void Form1_Load(object sender, System.EventArgs e)
{
ArrayList list = new ArrayList();
Student s = new Student();
s.Name = "John Smith";
s.Id = "100001";
Score sc = new Score();
sc.ClassName = "math";
sc.Grade = "A";
s.Score.Add(sc);
sc = new Score();
sc.ClassName = "English";
sc.Grade = "A";
s.Score.Add(sc);
list.Add(s);
s = new Student();
s.Name = "David Black";
s.Id = "100002";
sc = new Score();
sc.ClassName = "math";
sc.Grade = "B";
s.Score.Add(sc);
sc = new Score();
sc.ClassName = "English";
sc.Grade = "A";
s.Score.Add(sc);
list.Add(s);
fpSpread1_Sheet1.DataSource = list;
}
}
```

VB

```
public class Score
{
private string classname;
private string grade;
public string ClassName
```

```
{
get { return classname; }
set { classname = value; }
}
public string Grade
{
get { return grade; }
set { grade = value; }
}
}

public class Student
{
private string name;
private string id;
private ArrayList score = new ArrayList();
public string Name
{
get { return name; }
set { name = value; }
}
public string Id
{
get { return id; }
set { id = value; }
}
public ArrayList Score
{
get { return score; }
}
}

private void Form1_Load(object sender, System.EventArgs e)
{
ArrayList list = new ArrayList();
Student s = new Student();
s.Name = "John Smith";
s.Id = "100001";
Score sc = new Score();
sc.ClassName = "math";
sc.Grade = "A";
s.Score.Add(sc);
sc = new Score();
sc.ClassName = "English";
sc.Grade = "A";
s.Score.Add(sc);
list.Add(s);
s = new Student();
s.Name = "David Black";
s.Id = "100002";
sc = new Score();
sc.ClassName = "math";
sc.Grade = "B";
s.Score.Add(sc);
sc = new Score();
sc.ClassName = "English";
sc.Grade = "A";
```

```
s.Score.Add(sc);
list.Add(s);
fpSpread1_Sheet1.DataSource = list;
}
```

Creating a Hierarchical Display Manually

You can manually (programmatically) create a hierarchical display as shown in the example below. The parent is the higher level of the hierarchy and the child is the lower level.

Example

This example creates two custom **SheetView** objects: one as the parent and one as the child. In the parent sheet, the example overrides the **ChildRelationCount** (**'ChildRelationCount Property' in the on-line documentation**) property and **GetChildView** (**'GetChildView Method' in the on-line documentation**) and **FindChildView** (**'FindChildView Method' in the on-line documentation**) methods. The **ChildRelationCount** is how many relations between this object and children objects (usually this is one).

The **GetChildView** is used to get the child sheet. It calls **FindChildView** to see if the sheet is already created. If it is, then use that sheet. If it is not, create a new child sheet.

The **SheetName** (**'SheetName Property' in the on-line documentation**) property of the child **SheetView** determines which parent row it is assigned to.

Override the **ParentRowIndex** (**'ParentRowIndex Property' in the on-line documentation**) property in the child **SheetView** to return the row number that child is assigned to. The **SheetName** in creating this sheet determines this number.

VB

```
...
Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
MyBase.Load
    fpSpread1.Sheets.Clear()
    fpSpread1.Sheets.Add(New customSheet)
    fpSpread1.Sheets(0).RowCount = 10
    Dim i As Integer
    For i = 0 To 9
        If i <> 1 Then
            fpSpread1.Sheets(0).SetRowExpandable(i, False)
        End If
    Next i
End Sub
End Class

Public Class customSheet
Inherits FarPoint.Win.Spread.SheetView

    Public Overrides ReadOnly Property ChildRelationCount() As Integer
        Get
            Return 1
        End Get
    End Property

    Public Overrides Function GetChildView(ByVal row As Integer, ByVal relationIndex As
Integer) As FarPoint.Win.Spread.SheetView
        Dim child As customChild = CType(FindChildView(row, 0), customChild)
```

```

    If Not (child Is Nothing) Then Return child
        child = New customChild
        child.RowCount = 5
        child.Parent = Me
        child.SheetName = row.ToString
        ChildViews.Add(child)
    Return child
End Function

Public Overrides Function FindChildView(ByVal row As Integer, ByVal relationIndex As
Integer) As FarPoint.Win.Spread.SheetView
    Dim id As String = row.ToString
    Dim View As FarPoint.Win.Spread.SheetView
    For Each View In ChildViews
        If View.SheetName = id Then Return View
    Next
    Return Nothing
End Function
End Class

Public Class customChild
Inherits FarPoint.Win.Spread.SheetView
    Public Overrides ReadOnly Property ParentRowIndex() As Integer           Get
Return CInt(SheetName)
    End Get
    End Property
End Class

```

Creating Custom Hierarchy Icons

You can customize the icons used for expanding and collapsing the hierarchy in a hierarchical display by using the **GetImage** ('[GetImage Method](#)' in the on-line documentation) and **SetImage** ('[SetImage Method](#)' in the on-line documentation) methods in the **SpreadView** ('[SpreadView Class](#)' in the on-line documentation) class, which is described in **Customizing the User Interface Images**.

An example of how the default hierarchy icons, a simple plus sign and minus sign, appear is shown here.



For more information about the hierarchical display, refer to the **Working with Hierarchical Data Display**.

Tutorial: Binding to a Corporate Database (Visual Studio 2013 or later)

The following tutorial walks you through creating a project and binding the Spread control to a database. These instructions are based on the steps required in Visual Studio 2013 or later.

Step 1 - Add Spread Control to the Project

The following steps will help you in adding Spread control to the project:

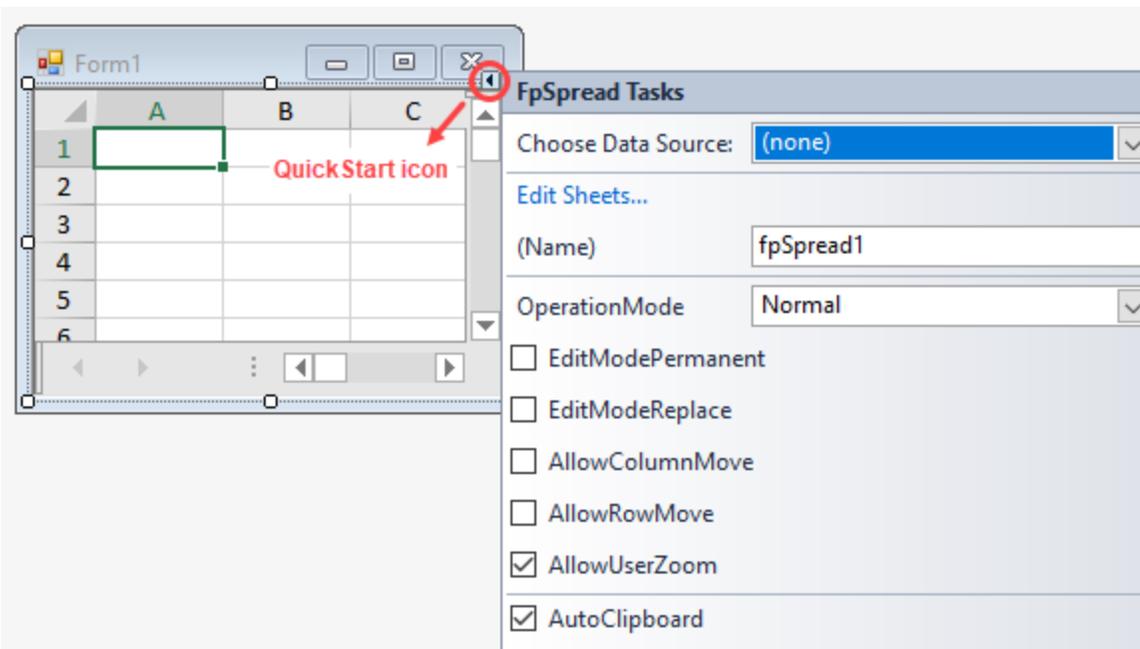
1. Start a new Visual Studio .NET project.
2. Provide a name to your project and the form file.
3. Now, add the FpSpread component to your project, and then place the component on the form.

If you do not know how to add the FpSpread component to the project, refer to **Adding a Component to a Visual Studio 2019 Project** or **Adding a Component to a Visual Studio 2015 or 2017 Project** or **Adding a Component to a Visual Studio 2013 Project (on-line documentation)**.

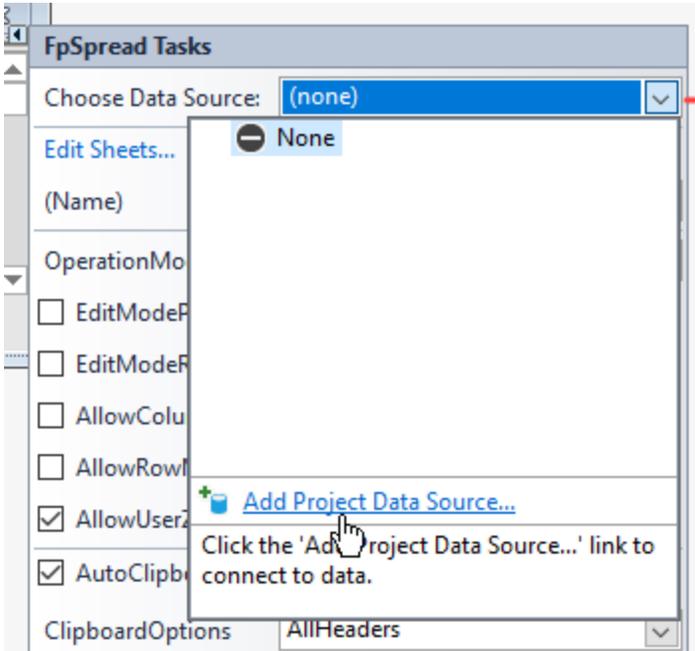
Step 2 - Configure Data Source

In this section, we will use the Quickstart menu to set up the database connection and the data set.

1. Select the form and click the **QuickStart** icon to launch the **FpSpreadTasks** menu as shown in the following image.

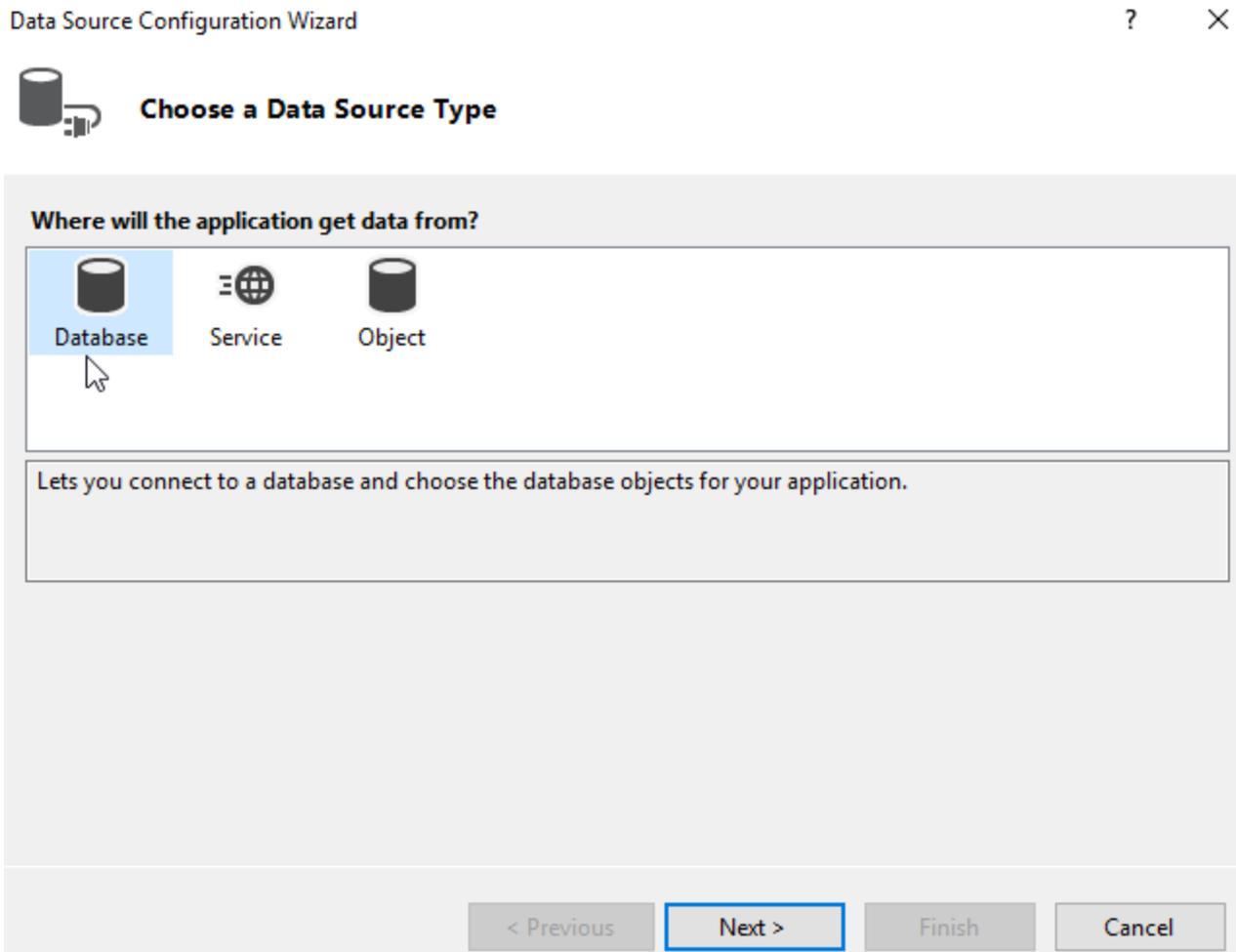


2. In the FpSpreadTasks menu, go to **Choose Data Source** and click the drop down button. In the drop down menu that appears, click **Add Project Data Source** to open the Data Source Configuration Wizard.



Click the drop down button and in the drop down menu, click "Add Project Data Source"

3. In the Data Source Configuration Wizard, click **Database** and then click **Next**.



4. Click **Dataset** to specify the database model that you want to use and then click **Next**.

Data Source Configuration Wizard



Choose a Database Model

What type of database model do you want to use?



Dataset

The database model you choose determines the types of data objects your application code uses. A dataset file will be added to your project.

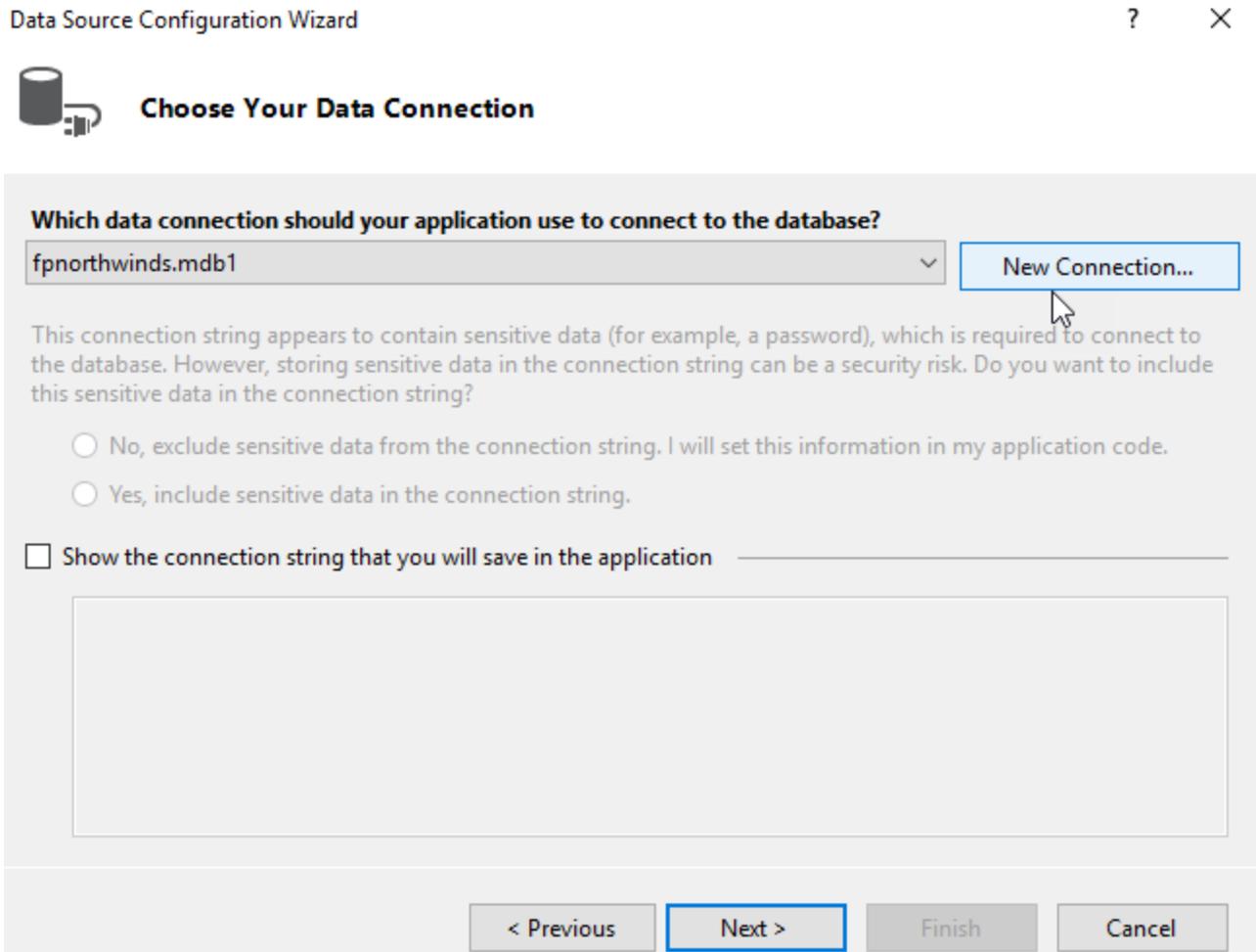
< Previous

Next >

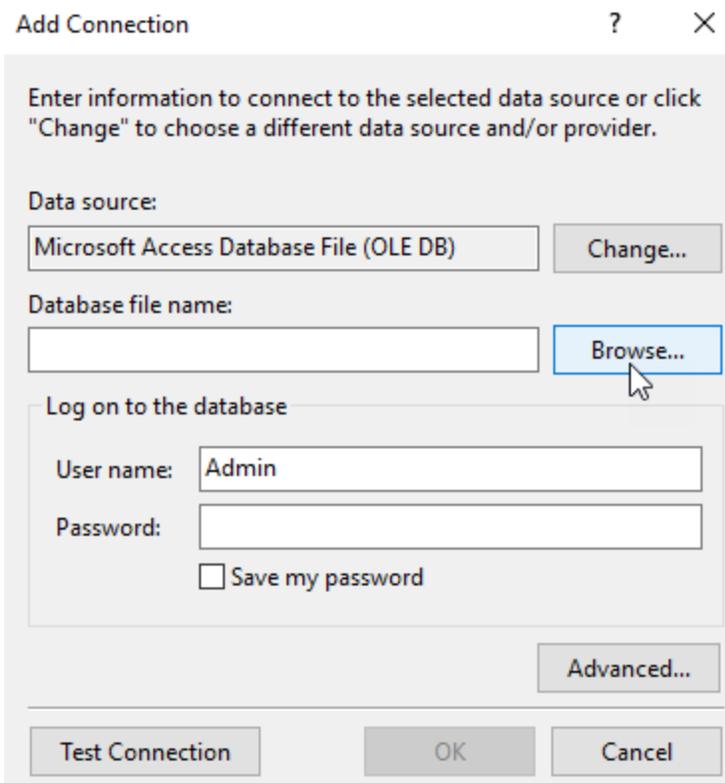
Finish

Cancel

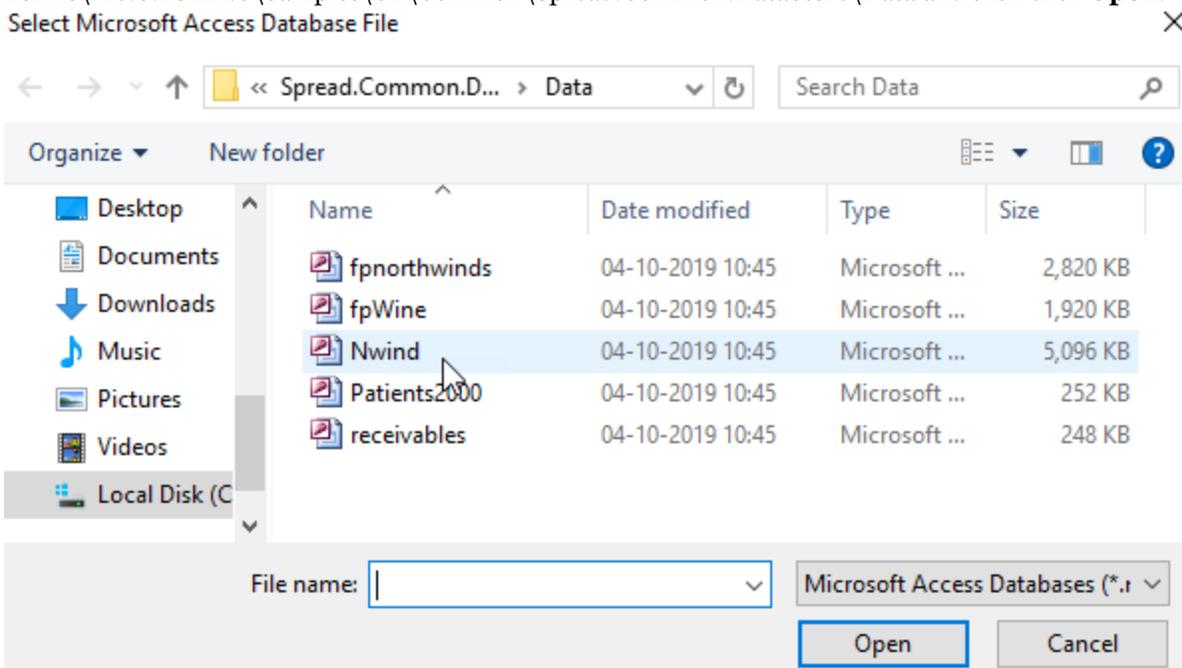
5. Under **Choose your Data Connection**, either choose an existing data connection or click **New Connection** and then click **Next** to display the **Add Connection** dialog.



6. In the **Add Connection** dialog, click **Browse** to open the **Select Microsoft Access Database File** dialog.

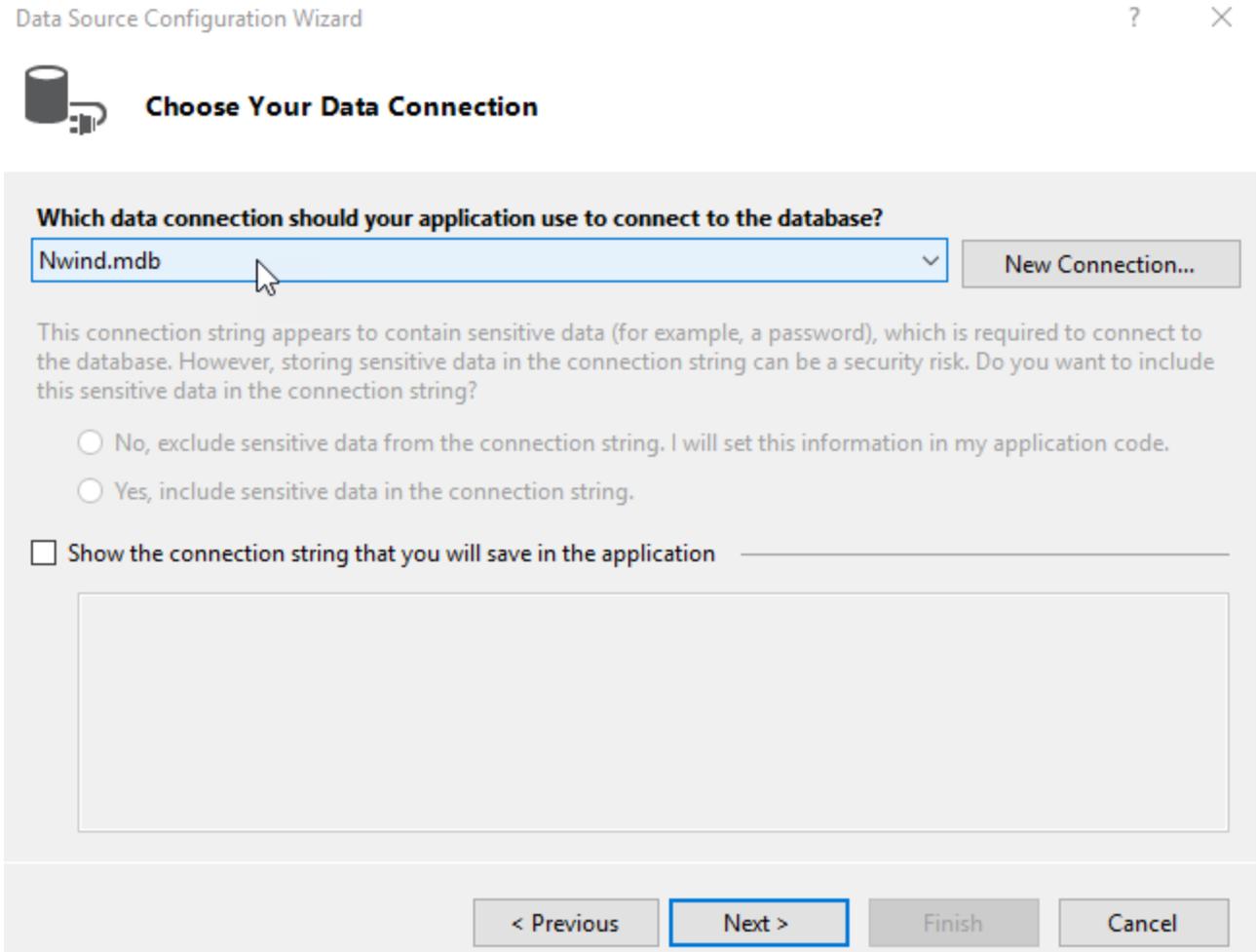


- In the **Select Microsoft Access Database File** dialog, make sure the folder path is set to C:\Program Files (x86)\Mescius\Spread.NET 16\Windows Forms\v16.0.20221.0\Samples\C#\Common\Spread.Common.DataStore\Data and then click **Open**.

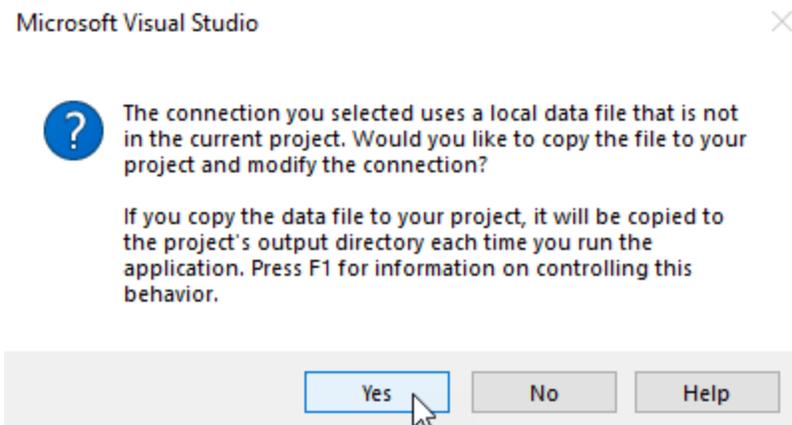


You can also click the **Test Connection** button in the **Add Connection** dialog to test the connection. If you do not receive a message stating the "Test connection succeeded" retry this step. If you receive the message "Test connection succeeded," your connection is complete. Click **OK** to close the Add Connection dialog.

- In the **Data Source Configuration Wizard**, make sure the database file that you had chosen in the above step is displayed under **Choose Your Data Connection**. Click **Next** to proceed.



- Now, Visual Studio displays a dialog that asks if you want to copy the database file to your project. Click **Yes** to continue.



- Under **Save the Connection String to the Application Configuration File**, make sure the box for **Yes, save the connection as** is checked, and accept the default name by clicking **Next**.

Data Source Configuration Wizard

? X

**Save the Connection String to the Application Configuration File**

Storing connection strings in your application configuration file eases maintenance and deployment. To save the connection string in the application configuration file, enter a name in the box and then click Next.

Do you want to save the connection string to the application configuration file?

Yes, save the connection as:

< Previous

Next >

Finish

Cancel

11. Under **Choose Your Database Objects**, click the arrow next to Tables to open the drop-down list of available tables. Click the arrow next to **Products** to display the list of fields in the **Products** table. You can choose any table that you want to include in your form.

Data Source Configuration Wizard

? X

**Choose Your Database Objects**

Which database objects do you want in your dataset?

- Tables (partially selected)
 - Categories
 - Customers
 - Employees
 - Order Details
 - Orders
 - Products
 - Shippers
 - Suppliers
- Views

DataSet name:

NwindDataSet

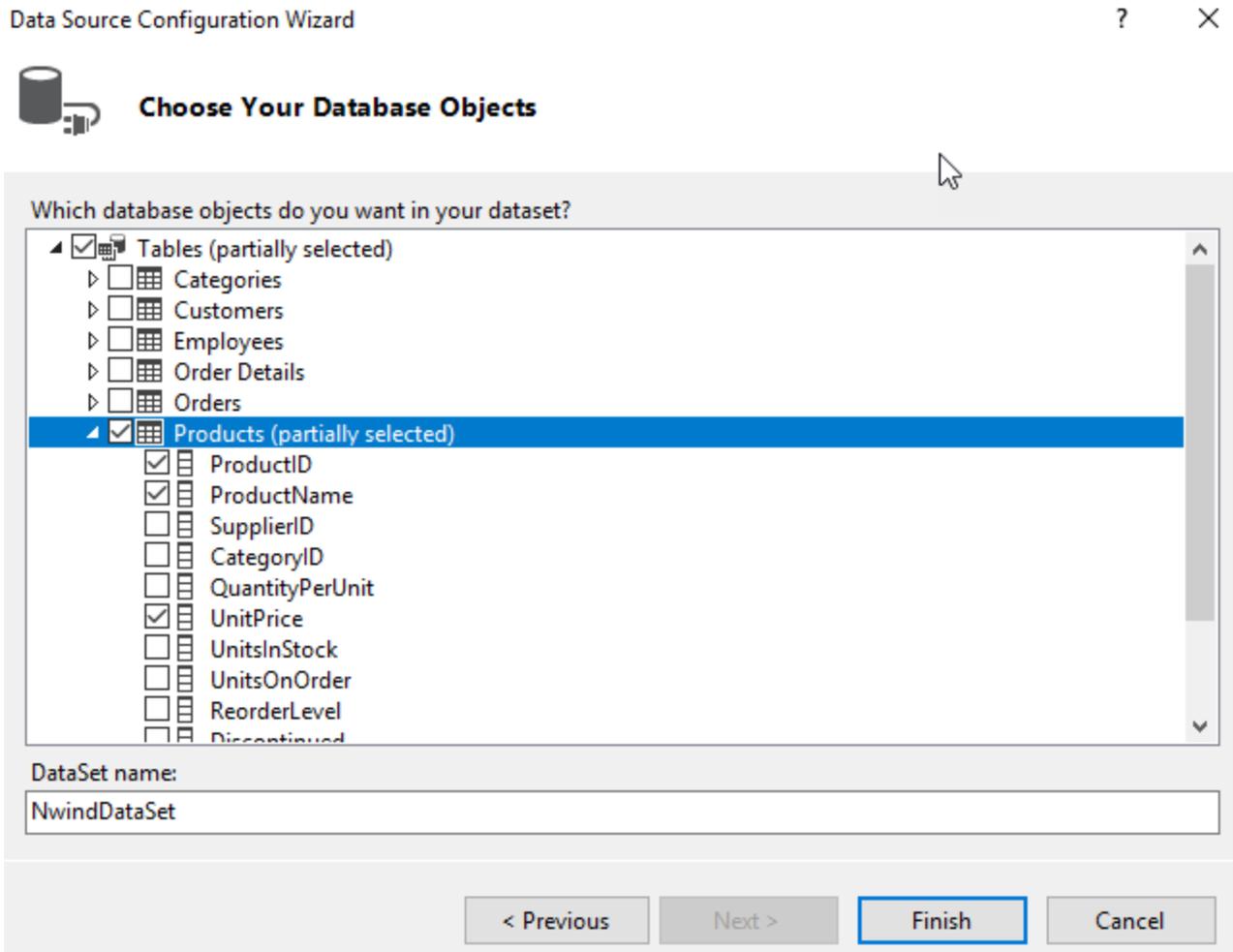
< Previous

Next >

Finish

Cancel

- In the list of fields, select **Product ID**, **Product Name** and **UnitPrice** as shown in the following image and click **Finish**. You can select as many fields as you want.



- The **NwindDataSet** is now added to your project. Notice that the column headers in Spread control get changed to the field names (**ProductID**, **ProductName** and **UnitPrice**) fetched from the **Products** table in your database.

Save your project and run it. You should see a form that looks similar to the following image with table data populated across the spreadsheet.



	ProductID	ProductName	UnitPrice
1	1	Chai	18.00
2	2	Chang	19.00
3	3	Aniseed Syrup	10.00
4	4	Chef Anton's Cajun S	22.00
5	5	Chef Anton's Gumbo	21.35
6	6	Grandma's Boysenbe	25.00
7	7	Uncle Bob's Organic I	30.00
8	8	Northwoods Cranber	40.00
9	9	Mishi Kobe Niku	97.00
10	10	Ikura	31.00
11	11	Queso Cabrales	21.00
12	12	Queso Manchego La	38.00
13	13	Konbu	6.00
14	14	Tofu	23.25
15	15	Genen Shouyu	15.50
16	16	Pavlova	17.45

If your form doesn't look similar to the image shown above, adjust the size of your Spread control and re-check the steps that you have performed so far.

Step 3 - Improve the Display of Data

In this step, you can change the cell type for one of the columns to display the data from the database in a better way.

1. Double-click on the form to open the code window.
2. Set the cell type for the **UnitPrice** column by adding the following code snippet below the code in the Form_Load event.
3. Save your project.

The following code snippet can be added to enhance the display of the data fetched from the database.

C#

```
// Bind the component to the data set.
fpSpread1.Sheets[0].AutoGenerateColumns = false;
fpSpread1.Sheets[0].DataSource = dbDataSet;
fpSpread1.Sheets[0].ColumnCount = 2;
fpSpread1.Sheets[0].BindDataColumn(0, "ID");
fpSpread1.Sheets[0].BindDataColumn(1, "Description");
```

VB

```
' Bind the component to the data set.
fpSpread1.Sheets(0).AutoGenerateColumns = False
fpSpread1.Sheets(0).DataSource = dbDataSet
fpSpread1.Sheets(0).ColumnCount = 2
fpSpread1.Sheets(0).BindDataColumn(0, "ID")
fpSpread1.Sheets(0).BindDataColumn(1, "Description")
```

Customizing the Sheet Appearance

You can customize the appearance of various parts of the Spread component.

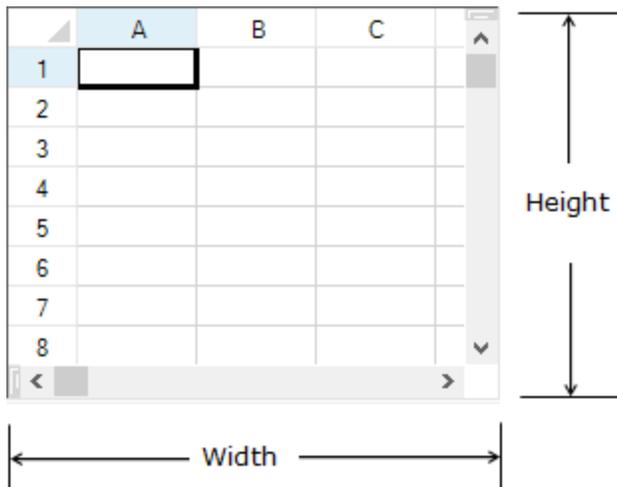
The tasks that relate to setting the appearance of objects in the Spread component include:

- **Customizing the Dimensions of the Component**
- **Customizing the Individual Sheet Appearance**
- **Customizing the Appearance of a Cell**
- **Customizing the Overall Component Appearance**
- **Creating and Applying a Style for Cells**
- **Using Conditional Formatting of Cells**
- **Customizing the Display of the Pointer**
- **Customizing the User Interface Images**
- **Using XP Themes with the Component**
- **Customizing the Renderers**
- **Handling Right-to-Left Layouts**
- **Customizing Painting of Parts of the Component**
- **Text Rendering with GDI**
- **Applying Theme to Customize the Appearance**

For information on customizing the appearance using the Spread Designer, refer to the **Spread Designer Guide (online documentation)**.

Customizing the Dimensions of the Component

You can set the overall dimensions of the Spread component and this determines the size of the visible area of the spreadsheet. The following figure shows the dimensions that you can set by setting the number of pixels for each.



Refer to the Microsoft .NET Framework documentation for more details on the Control.**Height** property or Control.**Width** property.

To calculate the height of the Spread, assuming scroll bars turned off and no headers, calculate the height of all the rows and then add one pixel for every border, so if 10 rows of 20 pixel height, $(10 \times 20) + (10 \times 1) + 1$, or 211 in this example. For the Spread width, the process is the same. For more information on setting the row height and column width, refer to **Setting the Row Height or Column Width**.

Using the Properties Window

1. Select the Spread component.
2. With the properties window open, in the **Layout** category, select the **Height** property or the **Width** property and type in a new value. The unit is pixels. Press **Enter**. The new dimension is now set. Refer to the Microsoft.NET Framework documentation for setting the units of measurement for height to something other than the default, which is pixels.

Using Code

Add a line of code that sets the specific dimension. Unless you have set it otherwise, the default for the unit of measurement is pixels. Use the **Height** and **Width** properties of the **FpSpread ('FpSpread Class' in the on-line documentation)** class.

Example

This example shows how to set the height of the Spread component to 250 pixels and the width to 300.

C#

```
fpSpread1.Height = 250;  
fpSpread1.Width = 300;
```

VB

```
fpSpread1.Height = 250  
fpSpread1.Width = 300
```

Customizing the Individual Sheet Appearance

You can have multiple sheets within a workbook. Each sheet is a separate spreadsheet and can have its own appearance and settings for user interaction. Each sheet has a unique name and sheet name tab for easy navigation between sheets.

These tasks relate to setting the appearance of the individual sheets inside the Spread component:

- **Setting the Background Colors for a Sheet**
- **Coloring a Cell**
- **Setting a Background Image for a Sheet**
- **Setting a Background Image to a Cell**

Other tasks that relate to the sheet appearance, but are part of the appearance of the Spread component include:

- **Customizing the Outline of the Component**
- **Customizing the Sheet Name Tabs**
- **Working with Hierarchical Data Display**
- **Creating and Applying a Style for Cells**

When you work with sheets, you can manipulate the objects using the short cuts in code, (**SheetView ('SheetView Class' in the on-line documentation)** and **SheetViewCollection ('SheetViewCollection Class' in the on-line documentation)** classes) or you can directly work with the model. Most developers who are not changing anything drastically find it easy to work with the short cut objects. For more information on models, refer to **Understanding the Underlying Models**.

Settings applied to a particular row or column or cell can override the settings that are set at the sheet level. Refer to **Object Parentage**.

For more details on the objects involved, refer to the **SheetView ('SheetView Class' in the on-line**

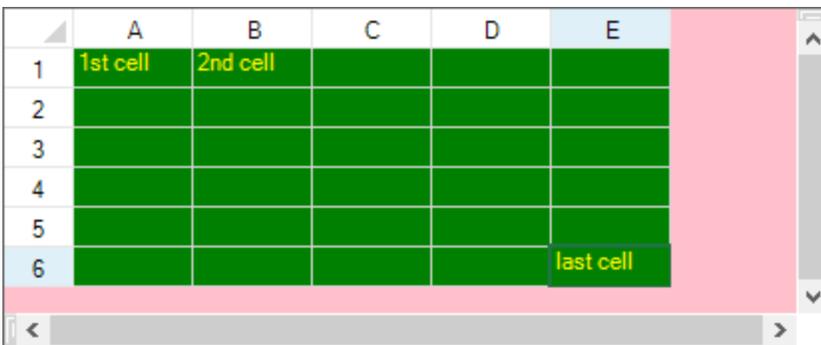
documentation) class and **SheetViewCollection** ('SheetViewCollection Class' in the on-line **documentation**) class.

Setting the Background Colors for a Sheet

There are two different background colors for a sheet. The first is the background of all the cells in the data area, which can be set at the sheet level. The second is the area beyond the cells but also set at the sheet level, which is called the gray area background color.

The background color for all the cells in the sheet, as well as other properties, can be set using the default style of the sheet. In this example, the background color of the default style for all of the cells is green. You can also set the background color for individual cells.

The gray area background color for the sheet is displayed in the area where cells are not displayed, as shown in the following figure. By default, the area is the system's Control color. This example sets the background color beyond the cells to be pink.



Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet for which to set the gray area background color.
5. Select the **GrayAreaBackColor** property in the property list, and then click the drop-down button to display the color picker.
6. Select a color in the color picker.
7. Click **OK** to close the editor.

Using a Shortcut

Set the Sheets shortcut object **GrayAreaBackColor** ('GrayAreaBackColor Property' in the on-line **documentation**) property for the sheet.

Example

This example code sets the first sheet's gray area background color to light yellow.

C#

```
// Set the first sheet's background color to light yellow.
fpSpread1.InterfaceRenderer = null;
fpSpread1.Sheets[0].GrayAreaBackColor = Color.LightYellow;
```

VB

```
' Set the first sheet's background color to light yellow.
fpSpread1.InterfaceRenderer = Nothing
fpSpread1.Sheets(0).GrayAreaBackColor = Color.LightYellow
```

Using Code

1. Create a new SheetView object.
2. Set the **GrayAreaBackColor** (**'GrayAreaBackColor Property' in the on-line documentation**) property for the SheetView object.
3. Assign the SheetView object to a sheet in the Spread component.

Example

This example code sets the first sheet's background color to light yellow.

C#

```
// Create a new SheetView object.
FarPoint.Win.Spread.SheetView newsheet = new FarPoint.Win.Spread.SheetView();
fpSpread1.InterfaceRenderer = null
// Set the SheetView object's background color to light yellow.
newsheet.GrayAreaBackColor = Color.LightYellow;
// Assign the SheetView object to the first sheet in the component.
fpSpread1.Sheets[0] = newsheet;
```

VB

```
' Create a new SheetView object.
Dim newsheet As New FarPoint.Win.Spread.SheetView()
fpSpread1.InterfaceRenderer = Nothing
' Set the SheetView object's background color to light yellow.
newsheet.GrayAreaBackColor = Color.LightYellow
' Assign the SheetView object to the first sheet in the component.
fpSpread1.Sheets(0) = newsheet
```

Using the Spread Designer

1. Select the sheet tab for the sheet for which you want to set the gray area background color.
2. In the property list, select the **GrayAreaBackColor** property.
3. Click the drop-down arrow to display the color picker.
4. Select a color from the color picker.
5. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Coloring a Cell

You can set the background and foreground (text) colors for a cell or for a group of cells. An example of the different ways to set the colors for a cell is shown in the following figure. The code that created these cell colors is provided in the example.

A	B	C	D
	This is default.		
	This is custom.		
	This is locked.		
	This is selected.		

You can specify the background color for a cell in code by using the **BackColor** (**'BackColor Property' in the on-line documentation**) property for that cell. You can specify the text color in code by using the **ForeColor** (**'ForeColor Property' in the on-line documentation**) property.

You can also specify the colors to display when the cells are selected using **SelectionBackColor** (**'SelectionBackColor Property' in the on-line documentation**) and **SelectionForeColor** (**'SelectionForeColor Property' in the on-line documentation**) for the sheet. For more information on selections, refer to **Customizing the Selection Appearance**.

You can also specify a different color (for background or for text) in locked cells using the **LockBackColor** (**'LockBackColor Property' in the on-line documentation**) and **LockForeColor** (**'LockForeColor Property' in the on-line documentation**) properties of the SheetView or Appearance objects. For more information on locked cells, refer to **Locking a Cell**.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet in which the cells appear.
5. In the properties list, select the **Cells** property and then click the button to display the **Cell, Column, and Row Editor**.
6. Select the cells for which you want to set the color.
7. In the properties list, select the **BackColor** property and select a color from the **Custom, Web, or System** tab. Select the **ForeColor** property and select that color.
8. Click **OK** to close the **Cell, Column, and Row Editor**.
9. Click **OK** to close the **SheetView Collection Editor**.

Using a Shortcut

Set the **BackColor** (**'BackColor Property' in the on-line documentation**) property or the **ForeColor** (**'ForeColor Property' in the on-line documentation**) property for the Cells object.

Example

This example code sets the background color and text color for the second cell, sets the colors for locked cells, and sets the colors for selections.

C#

```
fpSpread1.ActiveSheet.Cells[0,1].Value = "This is default.";
fpSpread1.ActiveSheet.Cells[1,1].Value = "This is custom.";
fpSpread1.ActiveSheet.Cells[2,1].Value = "This is locked.";
fpSpread1.ActiveSheet.Cells[3,1].Value = "This is selected.";
fpSpread1.ActiveSheet.Cells[1,1].BackColor = Color.LimeGreen;
fpSpread1.ActiveSheet.Cells[1,1].ForeColor = Color.Yellow;
fpSpread1.ActiveSheet.Cells[2,1].Locked = true;
```

```
fpSpread1.ActiveSheet.Protect = true;
fpSpread1.ActiveSheet.LockBackColor = Color.Brown;
fpSpread1.ActiveSheet.LockForeColor = Color.Orange;

fpSpread1.ActiveSheet.SelectionStyle =
FarPoint.Win.Spread.SelectionStyles.SelectionColors;
fpSpread1.ActiveSheet.SelectionPolicy =
FarPoint.Win.Spread.Model.SelectionPolicy.Range;
fpSpread1.ActiveSheet.SelectionUnit = FarPoint.Win.Spread.Model.SelectionUnit.Cell;
fpSpread1.ActiveSheet.SelectionBackColor = Color.Pink;
fpSpread1.ActiveSheet.SelectionForeColor = Color.Red;
```

VB

```
fpSpread1.ActiveSheet.Cells(0,1).Value = "This is default."
fpSpread1.ActiveSheet.Cells(1,1).Value = "This is custom."
fpSpread1.ActiveSheet.Cells(2,1).Value = "This is locked."
fpSpread1.ActiveSheet.Cells(3,1).Value = "This is selected."
fpSpread1.ActiveSheet.Cells(1,1).BackColor = Color.LimeGreen
fpSpread1.ActiveSheet.Cells(1,1).ForeColor = Color.Yellow
fpSpread1.ActiveSheet.Cells(2,1).Locked = True
fpSpread1.ActiveSheet.Protect = True
fpSpread1.ActiveSheet.LockBackColor = Color.Brown
fpSpread1.ActiveSheet.LockForeColor = Color.Orange

fpSpread1.ActiveSheet.SelectionStyle =
FarPoint.Win.Spread.SelectionStyles.SelectionColors
fpSpread1.ActiveSheet.SelectionPolicy = FarPoint.Win.Spread.Model.SelectionPolicy.Range
fpSpread1.ActiveSheet.SelectionUnit = FarPoint.Win.Spread.Model.SelectionUnit.Cell
fpSpread1.ActiveSheet.SelectionBackColor = Color.Pink
fpSpread1.ActiveSheet.SelectionForeColor = Color.Red
```

Using Code

Set the **BackColor** ('**BackColor Property**' in the on-line documentation) property or the **ForeColor** ('**ForeColor Property**' in the on-line documentation) property for a **Cell** ('**Cell Class**' in the on-line documentation) object.

Example

This example code sets the background color for cell A1 to Azure and the foreground color to Navy, then sets the background color for cells C3 through D4 to Bisque.

C#

```
FarPoint.Win.Spread.Cell cellA1;
cellA1 = fpSpread1.ActiveSheet.Cells[0, 0];
cellA1.BackColor = Color.Azure;
cellA1.ForeColor = Color.Navy;
FarPoint.Win.Spread.Cell cellrange;
cellrange = fpSpread1.ActiveSheet.Cells[2,2,3,3];
cellrange.BackColor = Color.Bisque;
```

VB

```
Dim cellA1 As FarPoint.Win.Spread.Cell
cellA1 = fpSpread1.ActiveSheet.Cells(0, 0)
```

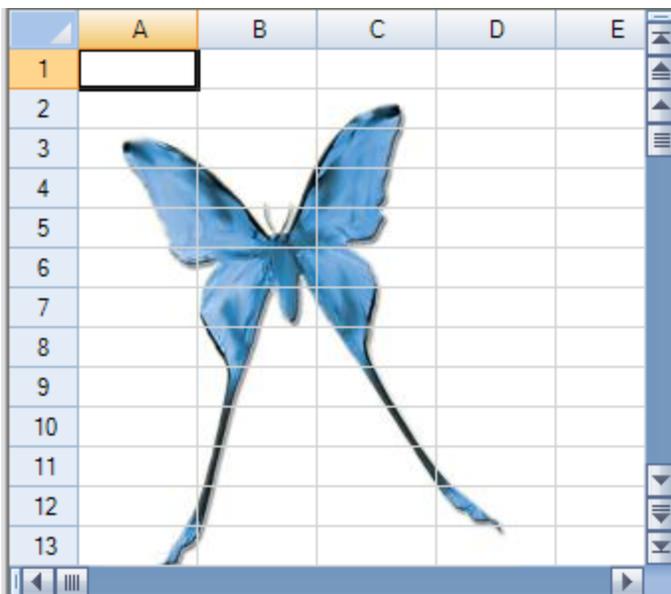
```
cellA1.BackColor = Color.Azure  
cellA1.ForeColor = Color.Navy  
Dim cellrange As FarPoint.Win.Spread.Cell  
cellrange = fpSpread1.ActiveSheet.Cells(2, 2, 3, 3)  
cellrange.BackColor = Color.Bisque
```

Using the Spread Designer

1. In the work area, select the cell or cells for which you want to set the background color.
2. In the properties list (in the **Misc** category), select the **BackColor** property to set the background color.
3. Click the drop-down button to display the color picker and choose the color from the available colors.
4. To set the text color, repeat those steps and select **ForeColor** property in the properties list.
5. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting a Background Image for a Sheet

You can set an image in the background of the cells in the data area of the sheet. Depending on the size of the graphic and the size of the spreadsheet, the image may be repeated (tiled) over the entire sheet of cells, as shown in the following figure.



For more information on setting an image in an individual cell, refer to **Setting a Background Image to a Cell**.

For more information on the image cell type, refer to **Setting an Image Cell**.

Using Code

1. Set the **BackgroundImage** (**'BackgroundImage Property' in the on-line documentation**) property.

Example

This example code sets the background image of the sheet.

C#

```
private void Form1_Load(object sender, System.EventArgs e)
```

```

{
    //Specify background images.
    fpSpread1.BackgroundImage = Image.FromFile("D:\\images\\butterfly.gif");
}

```

VB

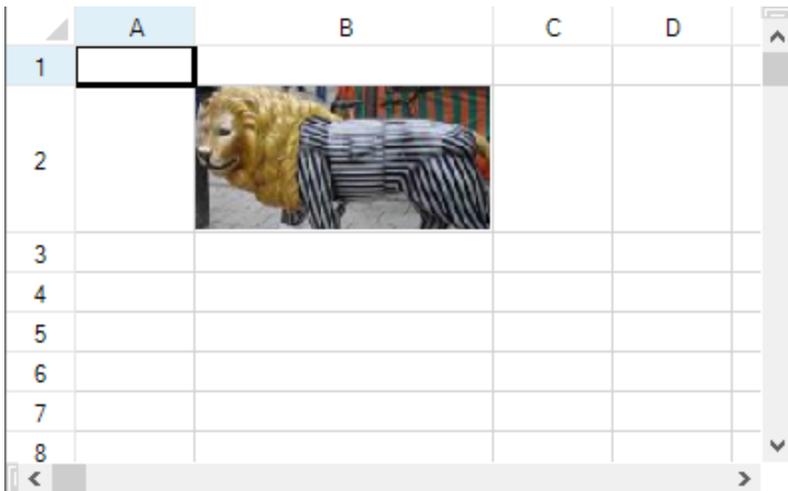
```

Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.Load
    'Specify background images.
    fpSpread1.BackgroundImage = Image.FromFile("D:\images\butterfly.gif")
End Sub

```

Setting a Background Image to a Cell

You can customize the background of a cell by adding a graphic image.



For more information on image cell types, refer to **Setting an Image Cell**.

For more information on setting an image for all the cells in a sheet (as part of the default style) refer to **Setting a Background Image for a Sheet**.

Example

The following example adds an image to a text cell.

C#

```

private void Form1_Load(object sender, System.EventArgs e)
{
    // Create an instance of a text cell.
    FarPoint.Win.Spread.CellType.TextCellType t = new
    FarPoint.Win.Spread.CellType.TextCellType();
    // Load an image file and set it to BackgroundImage property.
    FarPoint.Win.Picture p = new
    FarPoint.Win.Picture(Image.FromFile("D:\\images\\lionstatue.jpg"),
    FarPoint.Win.RenderStyle.Stretch);
    t.BackgroundImage = p;
    // Apply the text cell.
}

```

```
fpSpread1.ActiveSheet.Cells[1, 1].CellType = t;  
// Set the size of the cell so the image is displayed  
fpSpread1.ActiveSheet.Rows[1].Height = 50;  
fpSpread1.ActiveSheet.Columns[1].Width = 150;  
}
```

VB

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles  
MyBase.Load  
    ' Create an instance of a text cell.  
    Dim t As New FarPoint.Win.Spread.CellType.TextCellType  
    ' Load an image file and set it to BackgroundImage property.  
    Dim p As New FarPoint.Win.Picture(Image.FromFile("D:\\images\\lionstatue.jpg"),  
FarPoint.Win.RenderStyle.Stretch)  
    t.BackgroundImage = p  
    ' Apply the text cell.  
    fpSpread1.ActiveSheet.Cells(1, 1).CellType = t  
    ' Set the size of the cell so the image is displayed  
    fpSpread1.ActiveSheet.Rows(1).Height = 50  
    fpSpread1.ActiveSheet.Columns(1).Width = 150  
End Sub
```

Customizing the Appearance of a Cell

When you work with cells in the data area of the spreadsheet, you can work with the objects using the short cuts in code (**Cell ('Cell Class' in the on-line documentation)** and **Cells ('Cells Class' in the on-line documentation)** classes) or you can work directly with the model. Most developers who are not creating extensive customizations find it easier to work with the shortcut objects.

These tasks relate to setting the appearance of individual cells in the data area of the spreadsheet:

- **Customizing the Outline of the Component**
- **Displaying Grid Lines on a Sheet**
- **Creating and Customizing Cell Borders**



Note: The word "appearance" describes the general look of the cell, not the settings in the Appearance class, which contains only a few settings and is used for the appearance of several parts of the interface. Most of the appearance settings for a cell are in the **StyleInfo ('StyleInfo Class' in the on-line documentation)** class.

Settings applied to a particular cell override the settings that are set at the column or row level. For a more detailed explanation, refer to **Object Parentage**.

Other cell-level appearance settings are set by the cell type. For more information on settings related to cell types, refer to **Customizing Interaction with Cell Types**. You can edit properties of the **Cells** classes in the **Properties** window (in Spread Designer or in Visual Studio .NET). For more information on the **Cells, Columns, and Rows Editor** that is available from the **Properties** window, refer to the explanation of this editor in the **Spread Designer Guide (on-line documentation)**.

For more information, refer to the following topics:

- For information on header cells, refer to **Customizing the Appearance of Headers**.
- For tasks that relate to setting the user interaction at the cell level, refer to **Customizing Interaction in Cells**.
- For information on customizing the appearance of cells using the Spread Designer, refer to the **Spread Designer Guide (on-line documentation)**.

- For more information on the Cell and Cells objects, refer to the **Assembly Reference (on-line documentation)**.

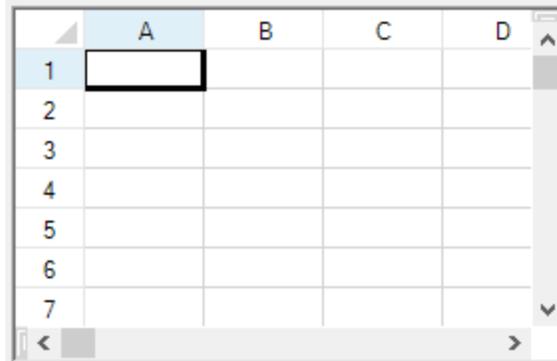
Customizing the Outline of the Component

You can set the appearance of the outline of the overall component. The following figures show the types of outlines (or border) styles.

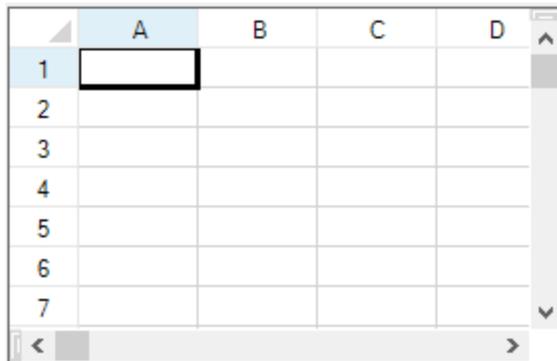
Outline (Border) Style

Fixed, three-dimensional (default)

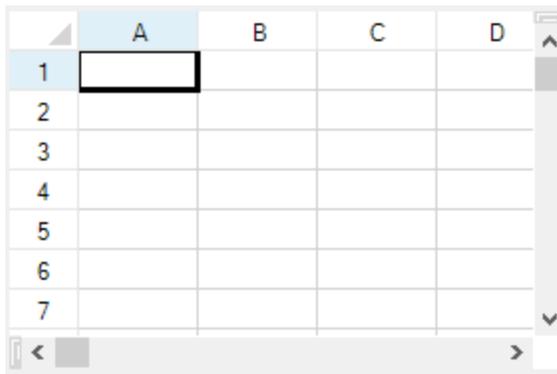
Example



Fixed, single-line



None



For more details, refer to the `FpSpread.BorderStyle` (**'BorderStyle Property' in the on-line documentation**) property and the `BorderStyle` enumeration in the Microsoft .NET Framework.

Using the Properties Window

1. Select the Spread component.
2. In the **Properties** window, in the **Appearance** category, select the **BorderStyle** property.
3. Select a value from the drop-down list. Press **Enter**. The new property is now set.

Using Code

Add a line of code that sets the specific property, the **BorderStyle** ('**BorderStyle Property**' in the on-line documentation) property of the **FpSpread** ('**FpSpread Class**' in the on-line documentation) class.

Example

This example shows how to set the border to be a single-line border.

C#

```
fpSpread1.BorderStyle = BorderStyle.FixedSingle;
```

VB

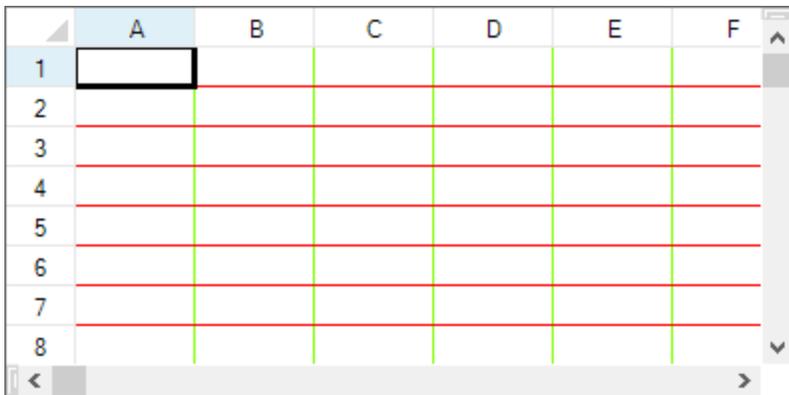
```
fpSpread1.BorderStyle = BorderStyle.FixedSingle
```

Using the Spread Designer

1. In the property list, in the **Appearance** category, select the **BorderStyle** property.
2. From the drop-down list, select the border style.
3. From the **File** menu, select **Apply and Exit** to apply your changes to the Spread component and exit Spread Designer.

Displaying Grid Lines on a Sheet

By default sheets display grid lines. You can set the color, the width, and the style of grid lines. In the following figure, the horizontal grid lines are flat and red, and the vertical grid lines are flat and green.



You can choose to display the grid lines as three-dimensional lines, with a highlight and shadow color. If you do so, set the highlight and shadow color to create the effect you want. For basic customization of grid lines, use the following properties of the **GridLine** ('**GridLine Class**' in the on-line documentation) class.

- **Color** ('**Color Property**' in the on-line documentation) - the color of the grid line
- **HighlightColor** ('**HighlightColor Property**' in the on-line documentation) - the highlight color for three-dimensional grid lines
- **ShadowColor** ('**ShadowColor Property**' in the on-line documentation) - the shadow color for three-

dimensional grid lines

- **Type ('Type Property' in the on-line documentation)** - the type of grid line
- **Width ('Width Property' in the on-line documentation)** - the width in pixels of the grid line

The **Color** property is for flat lines, and the **HighlightColor** and **ShadowColor** property are for the other types of lines.

You can hide the grid line in a particular direction by setting the **Type** property of grid line to "None". Refer to the following line of code to hide horizontal grid line.

C#

```
fpSpread1.ActiveSheet.HorizontalGridLine = new
FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.None);
```

Visual Basic

```
fpSpread1.ActiveSheet.HorizontalGridLine = New
FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.None)
```

To customize other properties, create a new **GridLine ('GridLine Class' in the on-line documentation)** object (rather than changing a property on the existing object).

For more details on how to work with the grid lines in code, refer to the examples below, the **GridLine ('GridLine Class' in the on-line documentation)** class, and **HorizontalGridLine ('HorizontalGridLine Property' in the on-line documentation)** property or **VerticalGridLine ('VerticalGridLine Property' in the on-line documentation)** property of the **SheetView ('SheetView Class' in the on-line documentation)** class.



- You can display lines around individual cells by setting cell borders. For more information, refer to **Creating and Customizing Cell Borders**.
- For information on setting the grid lines in a header, refer to **Customizing the Header Grid Lines**.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet for which you want to set the grid line color.
5. To set the horizontal grid line color,
 - a. Select the **HorizontalGridLine** object in the property list.
 - b. If you want to display three-dimensional grid lines, set the **Type** property to **Lowered** or **Raised**.
 - c. If the grid lines are not three-dimensional, select the **Color** property, and then click the drop-down button to display the color picker. Select a color in the color picker.
 - d. If the grid lines are three-dimensional, select the **HighlightColor** property, and then click the drop-down button to display the color picker. Select a color in the color picker. Do the same for the **ShadowColor** property.
6. To set the vertical grid line color,
 - a. Select the **VerticalGridLine** object in the property list
 - b. If you want to display three-dimensional grid lines, set the **Type** property to **Lowered** or **Raised**.
 - c. If the grid lines are not three-dimensional, select the **Color** property, and then click the drop-down button to display the color picker. Select a color in the color picker.
 - d. If the grid lines are three-dimensional, select the **HighlightColor** property, and then click the drop-down button to display the color picker. Select a color in the color picker. Do the same for the **ShadowColor** property.

7. Click **OK** to close the editor.

Using Code

1. Create a **GridLine ('GridLine Class' in the on-line documentation)** object, setting the color or colors and the style for the grid line in the constructor.
2. Assign the **GridLine ('GridLine Class' in the on-line documentation)** object to the specified sheet by setting the **SheetView ('SheetView Class' in the on-line documentation)** object **HorizontalGridLine ('HorizontalGridLine Property' in the on-line documentation)** or **VerticalGridLine ('VerticalGridLine Property' in the on-line documentation)** property to the **GridLine ('GridLine Class' in the on-line documentation)** object you created in step 1.

Example

This example code sets the horizontal grid line color to red and the vertical grid line color to chartreuse. Both grid lines are flat.

C#

```
FarPoint.Win.Spread.GridLine HGridLine = new
FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Flat, Color.Red);
FarPoint.Win.Spread.GridLine VGridLine = new
FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Flat, Color.Chartreuse);
fpSpread1.Sheets[0].HorizontalGridLine = HGridLine;
fpSpread1.Sheets[0].VerticalGridLine = VGridLine;
```

VB

```
Dim HGridLine As New
FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Flat, Color.Red)
Dim VGridLine As New
FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Flat, Color.Chartreuse)
fpSpread1.Sheets(0).HorizontalGridLine = HGridLine
fpSpread1.Sheets(0).VerticalGridLine = VGridLine
```

Using the Spread Designer

1. Select the sheet tab for the sheet for which you want to set the grid line colors.
2. To set the horizontal grid line color,
 - a. In the **Appearance** category, select the **HorizontalGridLine** object in the property list.
 - b. If you want to display three-dimensional grid lines, set the **Type** property to **Lowered** or **Raised**.
 - c. If the grid lines are not three-dimensional, select the **Color** property, and then click the drop-down button to display the color picker. Select a color in the color picker.
 - d. If the grid lines are three-dimensional, select the **HighlightColor** property, and then click the drop-down button to display the color picker. Select a color in the color picker. Do the same for the **ShadowColor** property.
3. To set the vertical grid line color,
 - a. In the **Appearance** category, select the **VerticalGridLine** object in the property list
 - b. If you want to display three-dimensional grid lines, set the **Type** property to **Lowered** or **Raised**.
 - c. If the grid lines are not three-dimensional, select the **Color** property, and then click the drop-down button to display the color picker. Select a color in the color picker.
 - d. If the grid lines are three-dimensional, select the **HighlightColor** property, and then click the drop-down button to display the color picker. Select a color in the color picker. Do the same for the **ShadowColor** property.

- From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Creating and Customizing Cell Borders

You can customize the appearance of the cells by setting borders for a cell or range of cells. You can also set a border for a column, row, sheet, or range of cells, the effect being the same as assigning the same border object to each individual cell. A border can be displayed on the left, right, top, or bottom, or around all four sides of a cell or cell range.

The **SheetView ('SheetView Class' in the on-line documentation)** class also provides the **SetOutlineBorder ('SetOutlineBorder Method' in the on-line documentation)** method to set a border around the outside of a cell range, where it sets individual borders in each cell in the perimeter of the range. It also provides the **SetInsideBorder ('SetInsideBorder Method' in the on-line documentation)** method to set the borders inside a cell range.

Border Display

The cell borders are drawn from left to right and top to bottom in the sheet. You can set a cell border by using the **Cell Border ('Border Property' in the on-line documentation)** property, **Column Border ('Border Property' in the on-line documentation)** property, or **Row Border ('Border Property' in the on-line documentation)** property. By default, no border is displayed.

The cell borders of the left and top edges are painted depending on the setting of the **BorderCollapse ('BorderCollapse Property' in the on-line documentation)** property.

- BorderCollapse.Separate** (Default) paints the left and top edges of the cell border just inside the grid lines. Thus, the left and top cell border edges are displayed in the left and top rows.
- BorderCollapse.Collapse** paints the left and top edges of the cell border over the grid lines to the left and top of the cell.
- BorderCollapse.Enhanced** paints the left and top edges of the cell border the same as in Excel.

The left column and top row (of a viewport, row header, column header, or sheet corner) are positioned so that those grid lines are just outside of the viewable area and thus those cell border edges are just outside of the viewable area (that is, those cell border edges are not displayed). Keep this in mind if you choose not to display a border for the Spread component or headers for the sheet, as the result might be visually confusing. The right and bottom edges of the cell border are always painted over the grid lines to the right and bottom of the cell, regardless of the **BorderCollapse** setting. For more information, see the **Overlapping Borders** section in this topic.

 **Note:** If two adjacent borders have a different style or color, the last one drawn has precedence and is the one that is displayed. Cell borders reflect the precedence used by the sheet to determine the characteristics for sheet elements. For more information, see the list of precedence in the description of **Object Parentage**.

Border Styles

A border can be displayed as any of the built-in styles shown in the following table or customized borders that you define.

Style	Example	Description	FarPoint.Win Class Name
Beveled		Has three-dimensional appearance if the highlight and shadow are set to different colors.	BevelBorder ('BevelBorder Class' in the on-line documentation)
Complex		Each side of the cell can display a different color and type of border, with border patterns such as dashed or dotted. (See Creating a Complex Border with Multiple Lines.)	ComplexBorder ('ComplexBorder Class' in the on-line documentation)

Compound		Has two beveled borders, which can be separated by a frame	CompoundBorder ('CompoundBorder Class' in the on-line documentation)
Double-line		Has two parallel lines.	DoubleLineBorder ('DoubleLineBorder Class' in the on-line documentation)
Single-line border		Has a simple, single line.	LineBorder ('LineBorder Class' in the on-line documentation)
Rounded-edge, single-line		Has a single line but the corners are rounded.	RoundedLineBorder ('RoundedLineBorder Class' in the on-line documentation)

You can specify more than one style and color for the same cell, column, row, or block of cells. Different border styles let you set different options. For example, the complex border lets you set different styles of border display for each side of the cell. For each of these border styles, you can turn off the display of the border on any side of the cell.

The following example code creates a bevel border and then sets a cell's border to be the bevel border.

C#

```
// Create the bevel border.
FarPoint.Win.BevelBorder bevelbrdr = new
FarPoint.Win.BevelBorder(FarPoint.Win.BevelBorderType.Raised, Color.Cyan,
Color.DarkCyan);
// Set the bevel border to the cell B3 border.
fpSpread1.Sheets[0].Cells[4, 3].Border = bevelbrdr;
```

VB

```
' Create the bevel border.
Dim bevelbrdr As New FarPoint.Win.BevelBorder(FarPoint.Win.BevelBorderType.Raised,
Color.Cyan, Color.DarkCyan)
' Set the bevel border to the cell B3 border.
fpSpread1.Sheets(0).Cells(4, 3).Border = bevelbrdr
```

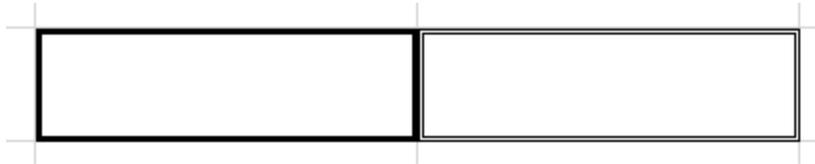
 For information on customizing borders using the Spread Designer, refer to the description of the **Border Editor (on-line documentation)** in the Spread Designer Guide.

Overlapping Borders

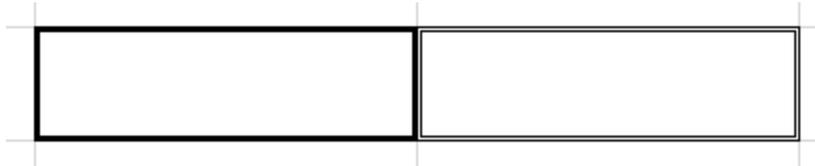
Cell borders are applied around the edge of each cell, and can overlap other cell borders. Whether borders overlap is determined by the setting of the **BorderCollapse ('BorderCollapse Property' in the on-line documentation)** property of the **FpSpread ('FpSpread Class' in the on-line documentation)** class.

The following table shows all the **BorderCollapse ('BorderCollapse Enumeration' in the on-line documentation)** enumeration options.

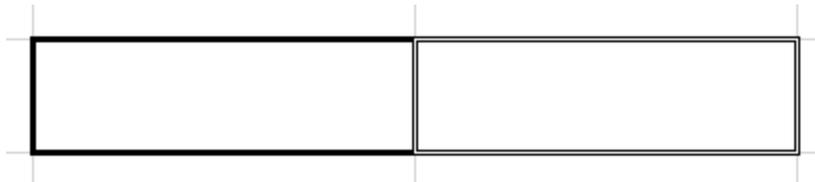
Borders	Description
Separate borders	Displays borders for adjacent cells as separate borders detached from each other.



Collapsed borders Displays borders for adjacent cells as collapsed into a single border.



Enhanced borders Displays borders for adjacent cells as merged into a single cell.
Enables the imported XLSX with complex merged cells and cell borders to look exactly as in Excel.



Note: The following should be noted when using the `BorderCollapse.Enhanced` option:

- It works with flat style mode (no `LegacyBehaviors.Style`) only. If `LegacyBehaviors.Style` is used, it will work the same as `BorderCollapse.Collapse`.
- Diagonal lines are not supported currently.

If two adjacent cells have different settings, and the property is set to have the cell borders overlap, the cell that is to the right or to the bottom has precedence. Each subsequent cell's border properties take precedence over the cell drawn before it.

Cell borders only overlap the amount of the grid line width. Therefore, if two 3 pixel borders overlap, and the grid line is 1 pixel, the overlapped borders are 5 pixels wide.

C#

```
// Adjacent cells as separate borders detached from each other
fpSpread1.BorderCollapse = FarPoint.Win.Spread.BorderCollapse.Separate;

// Adjacent cells as collapsed into a single border
fpSpread1.BorderCollapse = FarPoint.Win.Spread.BorderCollapse.Collapse;

// Adjacent cells as merged into a single cell
fpSpread1.BorderCollapse = FarPoint.Win.Spread.BorderCollapse.Enhanced;
```

Visual Basic

```
'Adjacent cells as separate borders detached from each other
FpSpread1.BorderCollapse = FarPoint.Win.Spread.BorderCollapse.Separate

'Adjacent cells as collapsed into a single border
FpSpread1.BorderCollapse = FarPoint.Win.Spread.BorderCollapse.Collapse
```

```
'Adjacent cells as merged into a single cell'  
FpSpread1.BorderCollapse = FarPoint.Win.Spread.BorderCollapse.Enhanced
```

Difference with Grid Lines

Borders are different from grid lines in that they create a border around a cell or range of cells rather than distinguishing rows and columns. Borders are drawn over the grid lines.

If you display cell borders for all the cells in a sheet, you might want to turn off the grid line display by setting the grid line type of the **HorizontalGridLine** (**'HorizontalGridLine Property' in the on-line documentation**) and **VerticalGridLine** (**'VerticalGridLine Property' in the on-line documentation**) properties of the sheet to None. For more information on grid lines refer to **Displaying Grid Lines on a Sheet**.

Using the Properties Window

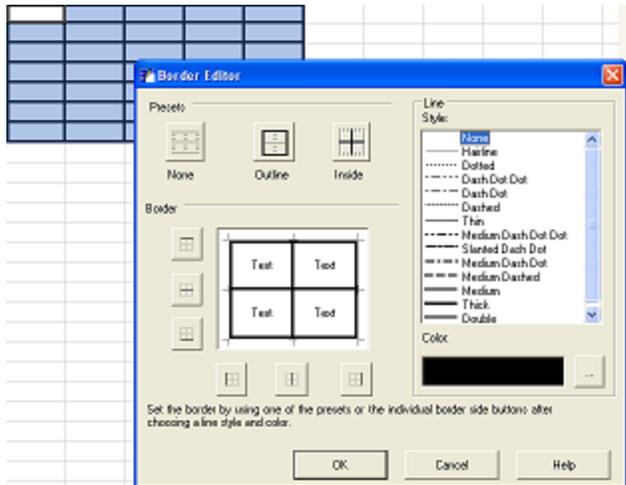
1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. Click the sheet for which you want to set cell borders.
5. Select the **Cells** property and then click the button to display the **Cell, Column, and Row Editor**.
6. Select the cell or cells for which you want to set the border.
7. In the property list, set the **Border** property.
8. If you want to customize the border you have set, double-click the **Border** property to display the border properties for the border style you have selected.
9. Set the border properties for your border.
10. Click **OK** to close the **Cell, Column, and Row Editor**.
11. Click **OK** to close the **SheetView Collection Editor**.

Using a Shortcut

1. Create a new border object of the type of border you want to create (BevelBorder, ComplexBorder, and so on).
2. Set the **Cells** shortcut object **Border** property to the new border object you created.

Using the Spread Designer

1. Select the sheet tab for the sheet for which you want to set a cell border.
2. Select the cell or range of cells for which you want to set the border.
3. Right-click and select **Borders**, or in the property list (in the **Misc** category), set the **Border** property.
4. If you want to customize the border you have set, double-click the **Border** property to display the border properties for the border style you have selected.
5. Set the border properties for your border.
6. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.



Creating a Complex Border with Multiple Lines

You can create a cell border with multiple lines using the complex border.

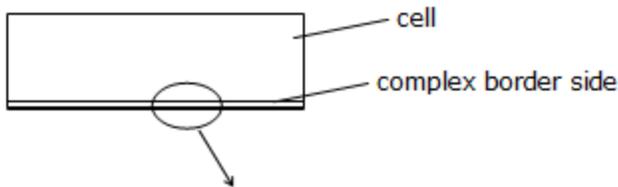
Using Code

Use the **CompoundArray** property of the **ComplexBorderSide** ('ComplexBorderSide Class' in the on-line **documentation**) class to create multiple-line borders for a cell.

Review the following examples for more information.

Example

This example code creates a ComplexBorderSide that has two underlines (two lines with a blank space in between) each taking a third of the width of the pen.



```
compoundArray = {0.00,0.33,0.66,1.00}
```

C#

```
// Create a new complex border side with two lines.
FarPoint.Win.ComplexBorderSide bottomborder = new FarPoint.Win.ComplexBorderSide(true,
```

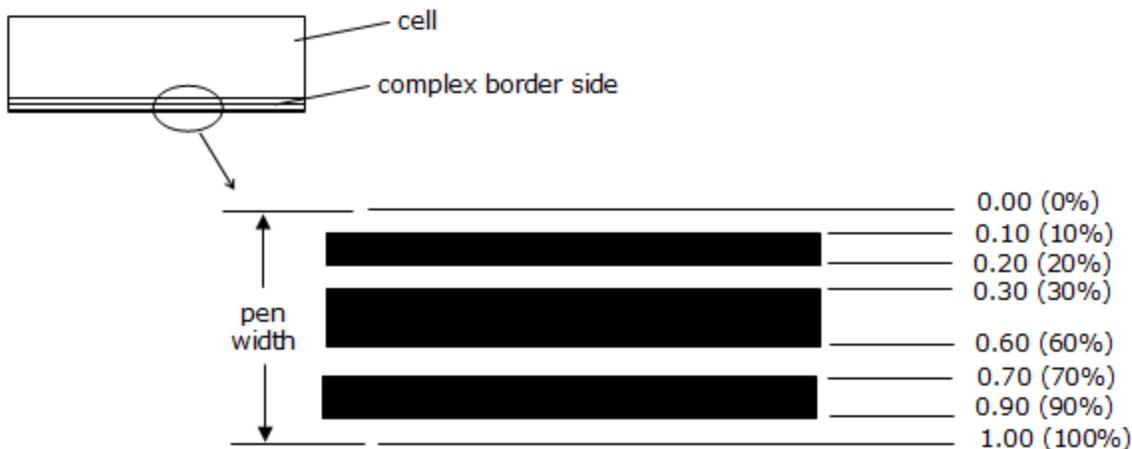
```
Color.Black, 3, System.Drawing.Drawing2D.DashStyle.Solid, null, new Single[] {0f,
0.33f, 0.66f, 1f});
fpSpread1.Sheets[0].Cells[3, 7].Border = new FarPoint.Win.ComplexBorder(null, null,
null, bottomborder);
```

VB

```
' Create a new complex border side with two lines.
Dim bottomborder As New FarPoint.Win.ComplexBorderSide(Color.Black, 3,
System.Drawing.Drawing2D.DashStyle.Solid, Nothing, New Single() {0, 0.33, 0.66, 1})
fpSpread1.Sheets(0).Cells(3, 7).Border = New FarPoint.Win.ComplexBorder(Nothing,
Nothing, Nothing, bottomborder)
```

Example

This example code creates a ComplexBorderSide that has three lines with varying amounts of thickness and with some blank space on either edge of the pen.



```
compoundArray = {0.10,0.20,0.30,0.6.0,0.70,0.90}
```

C#

```
// Create a new complex border side with three lines.
FarPoint.Win.ComplexBorderSide bottomborder = new FarPoint.Win.ComplexBorderSide(true,
Color.Black, 3, System.Drawing.Drawing2D.DashStyle.Solid, null, new Single[] {0.1f,
0.2f, 0.3f, 0.6f, 0.7f, 0.9f});
fpSpread1.Sheets[0].Cells[3, 7].Border = new FarPoint.Win.ComplexBorder(null, null,
null, bottomborder);
```

VB

```
' Create a new complex border side with three lines.
Dim bottomborder As New FarPoint.Win.ComplexBorderSide(True, Color.Black, 3,
System.Drawing.Drawing2D.DashStyle.Solid, Nothing, New Single() {0.1, 0.2, 0.3, 0.6,
0.7, 0.9})
fpSpread1.Sheets(0).Cells(3, 7).Border = New FarPoint.Win.ComplexBorder(Nothing,
Nothing, Nothing, bottomborder)
```

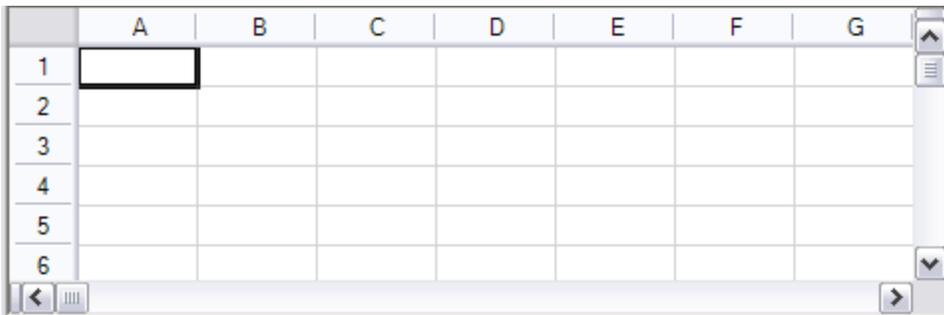
Customizing the Overall Component Appearance

You can quickly customize the appearance of the spreadsheet component by applying a "skin" to it. A skin is simply a collection of appearance properties that apply to an entire component or to an individual sheet such as colors, grid lines, and whether to show headers. This saves you the time and effort of setting the properties individually. Spread includes several built-in skins that are ready for you to use. You can also create your own custom skin and save it so that you can use it in other Spread components for a common format. These tasks relate to setting the appearance of the overall component:

- **Setting the Component to the Original Appearance**
- **Applying a Skin to the Component**
- **Creating a Custom Skin for a Component**
- **Applying a Skin to a Sheet**
- **Creating a Custom Skin for a Sheet**
- **Saving and Loading a Skin**
- **Creating and Applying a Style for Cells**

Setting the Component to the Original Appearance

You can set the appearance of the spreadsheet component to the original default look. This involves setting the renderers for the overall component, column header, row header, scroll bars, sheet corner, and focus indicator. When no skin is set to the spreadsheet, Excel2016 is used as the default skin for the spreadsheet.



If Spread Skin is applied to the spreadsheet, the painting logic will be used as before. The default skin will not replace or intercede the old spread skin.

Example

This example shows how to set DefaultSkin to the spreadsheet.

C#

```
fpSpread1.Skin = null;  
fpSpread1.DefaultSkin = FarPoint.Win.Spread.DefaultSpreadSkins.Office2016Colorful;
```

C#

```
fpSpread1.Skin = Nothing  
fpSpread1.DefaultSkin = FarPoInteger.Win.Spread.DefaultSpreadSkins.Office2016Colorful
```

Example

This example shows how to set Spread Skin of the spreadsheet.

C#

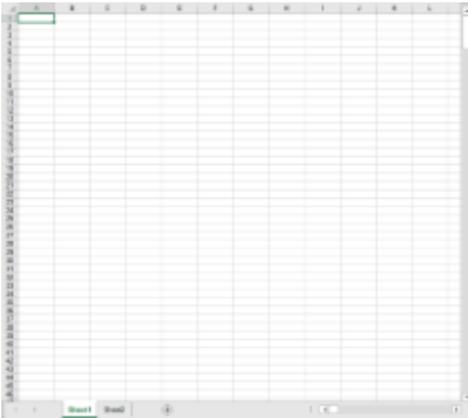
```
fpSpread1.Skin = FarPoint.Win.Spread.DefaultSpreadSkins.Office2016DarkGray;
```

VB

```
fpSpread1.Skin = FarPoInteger.Win.Spread.DefaultSpreadSkins.Office2016DarkGray
```

Applying a Skin to the Component

The skin applied to the control is defined in the SpreadSkin class. You can apply built-in skins (default skin) for creating common formats and custom skins that you create yourself. The built-in skins are defined in the DefaultSkins class. If no skin is applied to the spreadsheet component then default skin (Excel 2016 Colorful) is used. Refer to the following image for the same.



For more information and instructions about

Creating and applying your own skins
Creating and applying your own cell-level styles
Saving the skin to a file or stream
Customizing a skin in the Spread Designer
Spread skins

See

Creating a Custom Skin for a Component
Creating and Applying a Style for Cells
Saving and Loading a Skin
SpreadSkin Editor (on-line documentation)
SpreadSkin ('SpreadSkin Class' in the on-line documentation) class

Using the SpreadSkin Editor

1. If you want to create your own skin, follow the instructions provided in **Creating a Custom Skin for a Component** to create a skin and apply it. To apply a default skin to the entire spreadsheet component, follow these directions.
2. In the **Form** window, click the Spread component for which you want to set the skin (or right click on the component and choose the **Edit Skins** menu).
3. At the bottom of the **Properties** window, click the **Edit Skins** verb.
4. In the **SpreadSkin Editor**, select one of the Pre-Defined skins in the list of predefined skins, then click **OK** to close the editor.

Using a Shortcut

1. If you want to create your own skin, follow the instructions provided in **Creating a Custom Skin for a**

Component to create a sheet skin and apply it. To apply a default sheet skin, follow these directions.

2. Use the **Apply ('Apply Method' in the on-line documentation)** method on the selected default skin (set by the property in the **DefaultSkins ('DefaultSkins Class' in the on-line documentation)** object) to apply a specified default skin to a specific Spread component, collection of sheets, or sheet.

Example

This example code sets the component to use the Classic predefined skin.

C#

```
FarPoint.Win.Spread.DefaultSpreadSkins.Classic.Apply(fpSpread1);
```

VB

```
FarPoint.Win.Spread.DefaultSpreadSkins.Classic.Apply(fpSpread1)
```

Using Code

1. If you want to create your own skin, follow the instructions provided in **Creating a Custom Skin for a Component** to create a skin and apply it. To apply a default skin, follow these directions.
2. Use the **Apply ('Apply Method' in the on-line documentation)** method on the selected default skin (set by the property in the DefaultSpreadSkins object) to apply a specified default skin to a specific Spread component.

Example

This example code sets the first sheet to use the Classic predefined skin.

C#

```
// Create new SheetView object.  
FarPoint.Win.Spread.SheetView newsheet = new FarPoint.Win.Spread.SheetView();  
// Apply a skin to the SheetView object.  
FarPoint.Win.Spread.DefaultSpreadSkins.Classic.Apply(newsheet);  
// Assign the SheetView object to the first sheet in the component.  
fpSpread1.Sheets[0] = newsheet;
```

VB

```
' Create new SheetView object.  
Dim newsheet As New FarPoint.Win.Spread.SheetView()  
' Apply a skin to the SheetView object.  
FarPoint.Win.Spread.DefaultSpreadSkins.Classic.Apply(newsheet)  
' Assign the SheetView object to the first sheet in the component.  
fpSpread1.Sheets(0) = newsheet
```

Using the Spread Designer

1. From the **Settings** menu, choose the Spread Skin icon.
2. In the **Spread Skin Editor**, select one of the predefined skins from the **Pre-Defined** tab, or a saved custom skin from the **Saved** tab.
3. Click **OK** to close the editor.
4. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Creating a Custom Skin for a Component

You can quickly customize the appearance of the spreadsheet component by applying a "skin" to it. Some built-in skins are provided with Spread to create common formats. You can create your own custom skin and save it to use again, similar to a template.

For more information and instructions about

Applying the built-in skins
 Creating and applying your own cell-level styles
 Saving the skin to a file or stream
 Underlying model for skins
 Customizing a skin in the Spread Designer
 Spread skins

See

Applying a Skin to the Component
Creating and Applying a Style for Cells
Saving and Loading a Skin
Understanding the Style Model
SpreadSkin Editor (on-line documentation)
SpreadSkin ('SpreadSkin Class' in the on-line documentation) class

Using the SpreadSkin Editor

1. In the **Form** window, click the Spread component for which you want to create the skin.
2. At the bottom of the **Properties** window, click the **Edit Skins** verb.
3. In the **SpreadSkin Editor**, select the **Custom** tab.
4. Set the properties in the **Custom** tab to create the skin you want.
5. Set the Name property to specify the name for your custom skin.
6. Click the **Save Skin** button to save the skin.
 A dialog appears saying the skin has been saved.
7. Click **OK** to close the editor and apply the skin you created to the sheet, or click **Cancel** to close the editor and not apply the skin you created.

Using a Shortcut

1. Use the SpreadSkin object constructor, and set its parameters to specify the settings for the skin.
2. Use the **Apply ('Apply Method' in the on-line documentation)** method of the SpreadSkin object to apply it to the component.

Example

This example code creates and uses a custom skin.

C#

```
fpSpread1.Sheets.Count = 3;
FarPoint.Win.Spread.StyleInfo chd = new FarPoint.Win.Spread.StyleInfo();
chd.BackColor = Color.LightGreen;
FarPoint.Win.Spread.StyleInfo cds = new FarPoint.Win.Spread.StyleInfo();
cds.BackColor = Color.LightGreen;
FarPoint.Win.Spread.StyleInfo rhd = new FarPoint.Win.Spread.StyleInfo();
rhd.BackColor = Color.LightGreen;
FarPoint.Win.Spread.StyleInfo def = new FarPoint.Win.Spread.StyleInfo();
FarPoint.Win.Spread.GradientSelectionRenderer gsr = new
FarPoint.Win.Spread.GradientSelectionRenderer();
gsr.Color1 = Color.Green;
gsr.Color2 = Color.LightGreen;
```

```

gsr.Opacity = 50;
def.BackColor = Color.Honeydew;
FarPoint.Win.Spread.EnhancedInterfaceRenderer intl = new
FarPoint.Win.Spread.EnhancedInterfaceRenderer();
intl.ArrowColorDisabled = Color.Green;
intl.ArrowColorEnabled = Color.LightSeaGreen;
intl.ScrollBoxBackgroundColor = Color.Aqua;
intl.TabShape =
FarPoint.Win.Spread.EnhancedInterfaceRenderer.SheetTabShape.RoundedRectangle;
intl.TabStripButtonStyle =
FarPoint.Win.Spread.EnhancedInterfaceRenderer.ButtonStyles.Enhanced;
intl.TabStripButtonFlatStyle = FlatStyle.Popup;
intl.SheetTabBorderColor = Color.Aquamarine;
intl.SheetTabLowerActiveColor = Color.DarkSeaGreen;
intl.SheetTabLowerNormalColor = Color.DarkOliveGreen;
intl.SheetTabUpperActiveColor = Color.ForestGreen;
intl.SheetTabUpperNormalColor = Color.LightSeaGreen;
intl.SplitBarBackgroundColor = Color.Aquamarine;
intl.SplitBarDarkColor = Color.DarkGreen;
intl.SplitBarLightColor = Color.LightGreen;
intl.SplitBoxBackgroundColor = Color.Green;
intl.SplitBoxBorderColor = Color.LimeGreen;
intl.TabStripBackgroundColor = Color.Aquamarine;
FarPoint.Win.Spread.NamedStyle chstyle = new
FarPoint.Win.Spread.NamedStyle("ColumnHeaders", "HeaderDefault", chd);
FarPoint.Win.Spread.NamedStyle corner = new
FarPoint.Win.Spread.NamedStyle("CornerHeaders", "HeaderDefault", cds);
FarPoint.Win.Spread.NamedStyle rowhstyle = new
FarPoint.Win.Spread.NamedStyle("RowHeaders", "HeaderDefault", rhd);
FarPoint.Win.Spread.NamedStyle ds = new FarPoint.Win.Spread.NamedStyle("Default",
"DataAreaDefault", def);

FarPoint.Win.Spread.MarqueeFocusIndicatorRenderer focusrend = new
FarPoint.Win.Spread.MarqueeFocusIndicatorRenderer(Color.LightSeaGreen, 2);
FarPoint.Win.Spread.EnhancedScrollBarRenderer ScrollBarR = new
FarPoint.Win.Spread.EnhancedScrollBarRenderer(Color.Green, Color.LightGreen,
Color.Green, Color.Aqua, Color.DarkGreen, Color.DarkSeaGreen, Color.Turquoise,
Color.SpringGreen, Color.Teal, Color.PaleGreen, Color.ForestGreen);

FarPoint.Win.Spread.SpreadSkin skin = new FarPoint.Win.Spread.SpreadSkin("MySkin",
intl, ScrollBarR, focusrend, gsr, ds, chstyle, rowhstyle, corner);
skin.Apply(fpSpread1);

```

VB

```

' Create a custom skin.
fpSpread1.Sheets.Count = 3
Dim chd As New FarPoint.Win.Spread.StyleInfo
chd.BackColor = Color.LightGreen
Dim cds As New FarPoint.Win.Spread.StyleInfo
cds.BackColor = Color.LightGreen
Dim rhd As New FarPoint.Win.Spread.StyleInfo
rhd.BackColor = Color.LightGreen
Dim def As New FarPoint.Win.Spread.StyleInfo
Dim gsr As New FarPoint.Win.Spread.GradientSelectionRenderer
gsr.Color1 = Color.Green
gsr.Color2 = Color.LightGreen

```

```

gsr.LinearGradientMode = Drawing2D.LinearGradientMode.BackwardDiagonal
gsr.Opacity = 50
def.BackColor = Color.Honeydew
Dim int As New FarPoint.Win.Spread.EnhancedInterfaceRenderer
int.ArrowColorDisabled = Color.Green
int.ArrowColorEnabled = Color.LightSeaGreen
int.ScrollBoxBackgroundColor = Color.Aqua
int.TabShape =
FarPoint.Win.Spread.EnhancedInterfaceRenderer.SheetTabShape.RoundedRectangle
int.TabStripButtonStyle =
FarPoint.Win.Spread.EnhancedInterfaceRenderer.ButtonStyles.Enhanced
int.TabStripButtonFlatStyle = FlatStyle.Popup
int.SheetTabBorderColor = Color.Aquamarine
int.SheetTabLowerActiveColor = Color.DarkSeaGreen
int.SheetTabLowerNormalColor = Color.DarkOliveGreen
int.SheetTabUpperActiveColor = Color.ForestGreen
int.SheetTabUpperNormalColor = Color.LightSeaGreen
int.SplitBarBackgroundColor = Color.Aquamarine
int.SplitBarDarkColor = Color.DarkGreen
int.SplitBarLightColor = Color.LightGreen
int.SplitBoxBackgroundColor = Color.Green
int.SplitBoxBorderColor = Color.LimeGreen
int.TabStripBackgroundColor = Color.Aquamarine
Dim chstyle As New FarPoint.Win.Spread.NamedStyle("ColumnHeaders", "HeaderDefault",
chd)
Dim corner As New FarPoint.Win.Spread.NamedStyle("CornerHeaders", "HeaderDefault", cds)
Dim rowhstyle As New FarPoint.Win.Spread.NamedStyle("RowHeaders", "HeaderDefault", rhd)
Dim ds As New FarPoint.Win.Spread.NamedStyle("Default", "DataAreaDefault", def)
Dim focusrend As New
FarPoint.Win.Spread.MarqueeFocusIndicatorRenderer(Color.LightSeaGreen, 2)

Dim ScrollBarR As New FarPoint.Win.Spread.EnhancedScrollBarRenderer(Color.Green,
Color.LightGreen, Color.Green, Color.Aqua, Color.DarkGreen,
Color.DarkSeaGreen, Color.Turquoise, Color.SpringGreen, Color.Teal, Color.PaleGreen,
Color.ForestGreen)

Dim skin As New FarPoint.Win.Spread.SpreadSkin("MySkin", int, ScrollBarR, focusrend,
gsr, ds, chstyle, rowhstyle, corner)
skin.Apply(fpSpread1)

```

Using the Spread Designer

1. From the **Settings** menu, choose the **SpreadSkin** icon.
2. In the **Spread Skin Editor**, select the **Custom** tab.
3. Set the properties for the new custom skin, including the **Name** property to name your skin.
4. Select the **Save Skin** button.
A message box appears telling you your custom skin has been saved.
5. Click **OK** to close the **Spread Skin Editor**.
6. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Applying a Skin to a Sheet

You can quickly customize the appearance of a sheet by applying a "skin" to it. A skin is simply a collection of

appearance properties that apply to an entire sheet such as colors, grid lines, and whether to show headers. This saves you the time and effort of setting the properties individually.

Spread includes several built-in skins that are already made and ready for you to use. You can also create your own custom skin and save it so that you can use it in other Spread components for a common format.

If you apply a skin to a sheet with a hierarchy of child sheets, be sure to apply the skin to the parent sheet as well as to each of the child sheets. For more information on hierarchical displays, refer to **Working with Hierarchical Data Display**.

For more information and instructions about

Creating and applying your own sheet skins
 Creating and applying your own cell-level styles
 Saving the sheet skin to a file or stream
 Customizing a skin in the Spread Designer

Sheet skins

See

Creating a Custom Skin for a Sheet
Creating and Applying a Style for Cells
Saving and Loading a Skin
SheetSkin Editor (on-line documentation) in the Spread Designer Guide
SheetSkin ('SheetSkin Class' in the on-line documentation) class

Using the SheetSkin Editor

1. If you want to create your own sheet skin, follow the instructions provided in **Creating a Custom Skin for a Sheet** to create a sheet skin and apply it. To apply a default sheet skin to a sheet, follow these directions.
2. In the **Form** window, click the SheetView object for which you want to set the skin.
3. At the bottom of the **Properties** window, click the **Edit Skins** verb.
4. In the **SheetSkin Editor**, select one of the predefined skins in the Pre-Defined list, then click **OK** to close the editor.

Using a Shortcut

1. If you want to create your own sheet skin, follow the instructions provided in **Creating a Custom Skin for a Sheet** to create a sheet skin and apply it. To apply a default sheet skin, follow these directions.
2. Use the **Apply ('Apply Method' in the on-line documentation)** method on the selected default skin (set by the property in the **DefaultSkins ('DefaultSkins Class' in the on-line documentation)** object) to apply a specified default skin to a specific Spread component, collection of sheets, or sheet.

Example

This example code sets the first sheet to use the Colorful2 predefined skin.

C#

```
FarPoint.Win.Spread.DefaultSkins.Colorful2.Apply(fpSpread1.Sheets[0]);
```

VB

```
FarPoint.Win.Spread.DefaultSkins.Colorful2.Apply(fpSpread1.Sheets(0))
```

Using Code

1. If you want to create your own sheet skin, follow the instructions provided in **Creating a Custom Skin for a Sheet** to create a sheet skin and apply it. To apply a default sheet skin, follow these directions.

2. Use the **Apply ('Apply Method' in the on-line documentation)** method on the selected default skin (set by the property in the **DefaultSkins ('DefaultSkins Class' in the on-line documentation)** object) to apply a specified default skin to a specific Spread component, collection of sheets, or sheet.

Example

This example code sets the first sheet to use the Colorful2 predefined skin.

C#

```
// Create new SheetView object.
FarPoint.Win.Spread.SheetView newsheet = new FarPoint.Win.Spread.SheetView();
// Apply a skin to the SheetView object.
FarPoint.Win.Spread.DefaultSkins.Colorful2.Apply(newsheet);
// Assign the SheetView object to the first sheet in the component.
fpSpread1.Sheets[0] = newsheet;
```

VB

```
' Create new SheetView object.
Dim newsheet As New FarPoint.Win.Spread.SheetView()
' Apply a skin to the SheetView object.
FarPoint.Win.Spread.DefaultSkins.Colorful2.Apply(newsheet)
' Assign the SheetView object to the first sheet in the component.
fpSpread1.Sheets(0) = newsheet
```

Using the Spread Designer

1. Select the sheet tab for the sheet for which you want to set the skin.
2. From the **Settings** menu, choose **Sheet Skin Designer**.
3. In the **Sheet Skin Editor**, select one of the predefined skins from the **Pre-Defined** tab, or a saved custom skin from the **Saved** tab.
4. Click **OK** to close the editor.
5. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Creating a Custom Skin for a Sheet

You can quickly customize the appearance of a sheet by applying a "skin" to it. Some built-in skins are provided with Spread to create common formats. You can create your own custom skin and save it to use again, similar to a template. A skin, whether built-in or custom, can be applied to any number of sheets. Just as a style can be applied to cells, so a skin can be applied to an entire sheet.

For more information and instructions about

Applying the built-in sheet skins

Creating and applying your own cell-level styles

Saving the sheet skin to a file or stream

Underlying model for skins

Customizing a skin in the Spread Designer

Sheet skins

See

Applying a Skin to a Sheet

Creating and Applying a Style for Cells

Saving and Loading a Skin

Understanding the Style Model

SheetSkin Editor (on-line documentation) in the Spread Designer Guide

SheetSkin ('SheetSkin Class' in the on-line

documentation) class

Using the SheetSkin Editor

1. In the **Form** window, click the SheetView object for which you want to create the skin.
2. In the **Properties** window, in the **Appearance** category, select the **ActiveSkin** property and click on the button to launch the **SheetSkin Editor**.
3. In the **SheetSkin Editor**, select the **Custom** tab.
4. Set the properties in the **Custom** tab to create the skin you want.
5. Set the **Name** property to specify the name for your custom skin.
6. Click the **Save Skin** button to save the skin.
A dialog appears saying the skin has been saved.
7. Click **OK** to close the editor and apply the skin you created to the sheet, or click **Cancel** to close the editor and not apply the skin you created.

Using a Shortcut

1. Use the **SheetSkin ('SheetSkin Class' in the on-line documentation)** object constructor, and set its parameters to specify the settings for the skin.
2. Use the **Apply ('Apply Method' in the on-line documentation)** method of the **SheetSkin ('SheetSkin Class' in the on-line documentation)** object to apply it to the component, a sheet, or a set of sheets.

Example

This example code sets the first sheet to use a custom skin.

C#

```
// Create a custom skin.
FarPoint.Win.Spread.SheetSkin myskin = new FarPoint.Win.Spread.SheetSkin("MySkin",
Color.AliceBlue, Color.BlanchedAlmond, Color.Navy, Color.CornflowerBlue,
FarPoint.Win.Spread.GridLines.Both, Color.Coral, Color.Navy, Color.Bisque,
Color.Crimson, Color.AntiqueWhite, Color.BlanchedAlmond, true, true, true, true, true);
// Apply the custom skin to the first sheet in the component.
myskin.Apply(fpSpread1.Sheets[0]);
```

VB

```
' Create a custom skin.
Dim myskin As New FarPoint.Win.Spread.SheetSkin("MySkin", Color.AliceBlue,
Color.BlanchedAlmond, Color.Navy, Color.CornflowerBlue,
FarPoint.Win.Spread.GridLines.Both, Color.Coral, Color.Navy, Color.Bisque,
Color.Crimson, Color.AntiqueWhite, Color.BlanchedAlmond, True, True, True, True, True)
' Apply the custom skin to the first sheet in the component.
myskin.Apply(fpSpread1.Sheets(0))
```

Using Code

1. Call the SheetSkin object constructor, and set its parameters to specify the settings for the skin.
2. Call the SheetSkin object **Apply** method to apply it to the component, a sheet, or a set of sheets.

Example

This example code sets the first sheet to use a custom skin.

C#

```
// Create a custom skin.
FarPoint.Win.Spread.SheetSkin myskin = new FarPoint.Win.Spread.SheetSkin("MySkin",
Color.AliceBlue, Color.BlanchedAlmond, Color.Navy, Color.CornflowerBlue,
FarPoint.Win.Spread.GridLines.Both, Color.Coral, Color.Navy, Color.Bisque,
Color.Crimson, Color.AntiqueWhite, Color.BlanchedAlmond, true, true, true, true, true);
// Create a new SheetView object.
FarPoint.Win.Spread.SheetView newsheet = new FarPoint.Win.Spread.SheetView();
// Apply the custom skin to the SheetView object.
myskin.Apply(newsheet);
// Assign the SheetView object to the first sheet in the component.
fpSpread1.Sheets[0] = newsheet
```

VB

```
' Create a custom skin.
Dim myskin As New FarPoint.Win.Spread.SheetSkin("MySkin", Color.AliceBlue,
Color.BlanchedAlmond, Color.Navy, Color.CornflowerBlue,
FarPoint.Win.Spread.GridLines.Both, Color.Coral, Color.Navy, Color.Bisque,
Color.Crimson, Color.AntiqueWhite, Color.BlanchedAlmond, True, True, True, True, True)
' Create a new SheetView object.
Dim newsheet As New FarPoint.Win.Spread.SheetView()
' Apply the custom skin to the SheetView object.
myskin.Apply(newsheet)
' Assign the SheetView object to the first sheet in the component.
fpSpread1.Sheets(0) = newsheet
```

Using the Spread Designer

1. Select the sheet tab for the sheet for which you want to set the skin.
2. From the **Settings** menu, choose **SheetSkin Editor**.
3. In the **SheetSkin Editor**, select the **Custom** tab.
4. Set the properties for the new custom sheet skin, including the **Name** property to name your skin.
5. Select the **Save Skin** button.
A message box appears telling you your custom skin has been saved.
6. Click **OK** to close the **Sheet Skin Editor**.
7. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Saving and Loading a Skin

You can save skin settings for a sheet or the entire control to a file (.SKN) and then load it into another project or use it as a template for other projects. The **SheetSkin ('SheetSkin Class' in the on-line documentation)** class has **Load** and **Save** methods for a sheet. The **SpreadSkin ('SpreadSkin Class' in the on-line documentation)** class has **Load** and **Save** methods for the entire control. If you want to have a certain look for the entire control, use a spread skin. If you want each sheet to have a different look, use a sheet skin.

- **Saving a Skin**
- **Loading a Skin**

For more information on creating a skin for a sheet, refer to **Creating a Custom Skin for a Sheet**.

For more information on creating a skin for the entire control, refer to **Creating a Custom Skin for a Component**.

For more information on applying a skin, refer to **Applying a Skin to a Sheet** or **Applying a Skin to the Component**.

For information on customizing a skin in the Spread Designer, refer to the explanation of the **SheetSkin Editor (on-line documentation)** or the **SpreadSkin Editor (on-line documentation)** in the Spread Designer Guide.

For more details about skins, refer to the **SheetSkin ('SheetSkin Class' in the on-line documentation)** class or the **SpreadSkin ('SpreadSkin Class' in the on-line documentation)** class.

Saving a Skin

You can save the skin to a file or a stream.

If you do not specify a path, the file is saved in the Visual Studio project bin folder in a Debug subfolder. If you specify a path that includes a folder that does not exist, an error message appears during code compile.

Use the **Save ('Save Method' in the on-line documentation)** methods in the **SheetSkin ('SheetSkin Class' in the on-line documentation)** or **SpreadSkin ('SpreadSkin Class' in the on-line documentation)** class.

Using Code

Use the **Save(SheetSkin,String)** method in the **SheetSkin ('SheetSkin Class' in the on-line documentation)** class for saving to a file; use the **Save(SheetSkin,Stream)** method for saving to a stream.

Example

This example code saves a skin to a file.

C#

```
// define a skin and save it
FarPoint.Win.Spread.SheetSkin sk = new FarPoint.Win.Spread.SheetSkin("BlueSkin",
Color.White, Color.DarkGray, Color.Black, Color.Blue,
FarPoint.Win.Spread.GridLines.Both, Color.Blue, Color.Red, Color.White, Color.Black,
Color.LightGray, Color.Gray, false, false, true, true, true);
FarPoint.Win.Spread.SheetSkin.Save(sk, "C:\\BlueSkinTemplate.skn");
// FarPoint.Win.Spread.SpreadSkin.Save for a SpreadSkin
```

VB

```
' define a skin and save it
Dim sk As New FarPoint.Win.Spread.SheetSkin("BlueSkin", Color.White, Color.DarkGray,
Color.Black, Color.Blue, FarPoint.Win.Spread.GridLines.Both, Color.Blue, Color.Red,
Color.White, Color.Black, Color.LightGray, Color.Gray, False, False, True, True, True)
FarPoint.Win.Spread.SheetSkin.Save(sk, "C:\BlueSkinTemplate.skn")
' FarPoint.Win.Spread.SpreadSkin.Save for a SpreadSkin
```

Loading a Skin

You can load a skin from a file or a stream.

Use the **Load ('Load Method' in the on-line documentation)** methods in the **SheetSkin ('SheetSkin Class' in the on-line documentation)** or **SpreadSkin ('SpreadSkin Class' in the on-line documentation)** class.

Using Code

Use the **Load(String)** method in the **SheetSkin ('SheetSkin Class' in the on-line documentation)** class (or **SpreadSkin ('SpreadSkin Class' in the on-line documentation)** class) for loading from a file; use the **Load(Stream)** method for loading from a stream.

Example

This example code loads a skin from a file.

C#

```
// load a sheet skin
FarPoint.Win.Spread.SheetSkin.Load("C:\\BlueSkinTemplate.skn").Apply(fpSpread1.Sheets[0];
// load a spread skin
// FarPoint.Win.Spread.SpreadSkin.Load("C:\\farpoint.skn").Apply(fpSpread1);
```

VB

```
' load a sheet skin
FarPoint.Win.Spread.SheetSkin.Load("C:\\BlueSkinTemplate.skn").Apply(fpSpread1.Sheets(0))
' load a Spread Skin
'FarPoint.Win.Spread.SpreadSkin.Load("C:\\farpoint.skn").Apply(fpSpread1)
```

Creating and Applying a Style for Cells

You can quickly customize the appearance of a cell or range of cells (or rows or columns) by applying a "style". You can create your own named style and save it to use again, similar to a template, or you can simply change the properties of the default style. The style includes appearance settings that apply to cells, such as background color, text color, font, borders, and cell type. A style can be applied to any number of cells. Just as a skin can be applied to a sheet, so a style can be applied to cells.



Note: The word "appearance" is used for the general look of the cell, not simply the settings in the Appearance class, which contains only a few settings and is used for the appearance of several parts of the interface. Most of the appearance settings for a cell are in the **StyleInfo ('StyleInfo Class' in the on-line documentation)** class.

You typically set the style for the cell by using the **StyleName ('StyleName Property' in the on-line documentation)** property for the cell. You can also use the **ParentStyleName ('ParentStyleName Property' in the on-line documentation)** to set a style for a range of cells that may individually have different StyleName values set. A cell inherits all the style information from the parent style (**ParentStyleName**). When you set the parent style, you are setting the **Parent** property of the **StyleInfo** object assigned to each cell in the range. The parent for a named style can also be set by the **Parent** property of the **NamedStyle** object. So different cells (for example, cells in different rows or columns) may have different named styles but have the same parent style. For example, the cells may have different text colors (set in the named style) but inherit the same background color (set in the parent style).

For more information, refer to the **DefaultStyleCollection ('DefaultStyleCollection Class' in the on-line documentation)** class and the **NamedStyle ('NamedStyle Class' in the on-line documentation)** class. When you set the style (**StyleName**), you set the entire **StyleInfo ('StyleInfo Class' in the on-line documentation)** object in the style model for each cell in the range to the one in the **NamedStyleCollection ('NamedStyleCollection Class' in the on-line documentation)** with the specified name. The default parent style is set in the **DataAreaDefault** field in the **DefaultStyleCollection** class.

You can also create and apply appearance settings to an entire sheet by using sheet skins. For instructions on creating sheet skins, see **Creating a Custom Skin for a Sheet**.

For more information on the underlying model for styles, refer to **Understanding the Style Model**.

Using Code

1. Call the **NamedStyle** ('NamedStyle Class' in the on-line documentation) object constructor, and set its parameters to specify the name and settings for the style. You can also set the parent name.
2. Set the style settings.
3. Set the custom named style by assigning the style name to the cell or cells.

Example

This example code sets the first square of cells on the active sheet to use the same custom style that sets the background color to blue and sets the text color depending on which column.

C#

```
FarPoint.Win.Spread.NamedStyle backstyle = new
FarPoint.Win.Spread.NamedStyle("BlueBack");
backstyle.BackColor = Color.Blue;
FarPoint.Win.Spread.NamedStyle text1style = new
FarPoint.Win.Spread.NamedStyle("OrangeText", "BlueBack");
text1style.ForeColor = Color.Orange;
FarPoint.Win.Spread.NamedStyle text2style = new
FarPoint.Win.Spread.NamedStyle("YellowText", "BlueBack");
text2style.ForeColor = Color.Yellow;
fpSpread1.NamedStyles.Add(backstyle);
fpSpread1.NamedStyles.Add(text1style);
fpSpread1.NamedStyles.Add(text2style);
fpSpread1.ActiveSheet.Cells[0,0,4,0].StyleName = "OrangeText";
fpSpread1.ActiveSheet.Cells[0,1,4,1].StyleName = "YellowText";
```

VB

```
Dim backstyle As New FarPoint.Win.Spread.NamedStyle("BlueBack")
backstyle.BackColor = Color.Blue
Dim text1style As New FarPoint.Win.Spread.NamedStyle("OrangeText", "BlueBack")
text1style.ForeColor = Color.Orange
Dim text2style As New FarPoint.Win.Spread.NamedStyle("YellowText", "BlueBack")
text2style.ForeColor = Color.Yellow
fpSpread1.NamedStyles.Add(backstyle)
fpSpread1.NamedStyles.Add(text1style)
fpSpread1.NamedStyles.Add(text2style)
fpSpread1.ActiveSheet.Cells(0,0,4,0).StyleName = "OrangeText"
fpSpread1.ActiveSheet.Cells(0,1,4,1).StyleName = "YellowText"
```

Example

Configure the default style for the entire sheets by specifying **DefaultStyle** property (StyleInfo) in **SheetView** class. This approach is convenient when you want to apply one unique style to all cells in a sheet, as shown in the data area of the spreadsheet in this figure.

	A	B	C	D	E
1	0.00	1.00	2.00	3.00	4.00
2	1.00	2.00	3.00	4.00	5.00
3	2.00	3.00	4.00	5.00	6.00
4	3.00	4.00	5.00	6.00	7.00
5	4.00	5.00	6.00	7.00	8.00

C#

```
private void Form1_Load(object sender, System.EventArgs e)
{
    fpSpread1.ActiveSheet.RowCount = 5;
    fpSpread1.ActiveSheet.ColumnCount = 5;
    // Configure respective default styles.
    fpSpread1.ActiveSheet.DefaultStyle.BackColor = Color.LemonChiffon;
    fpSpread1.ActiveSheet.DefaultStyle.ForeColor = Color.Red;
    fpSpread1.ActiveSheet.DefaultStyle.CellType = new
FarPoint.Win.Spread.CellType.NumberCellType();
    fpSpread1.ActiveSheet.DefaultStyle.HorizontalAlignment =
FarPoint.Win.Spread.CellHorizontalAlignment.Center;
    fpSpread1.ActiveSheet.DefaultStyle.Border = new
FarPoint.Win.LineBorder(Color.Green);
    for (int i = 0; i < fpSpread1.ActiveSheet.RowCount; i++)
    {
        for (int j = 0; j < fpSpread1.ActiveSheet.ColumnCount; j++)
        {
            fpSpread1.ActiveSheet.SetValue(i, j, i + j);
        }
    }
}
```

VB

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
MyBase.Load
    fpSpread1.ActiveSheet.RowCount = 5
    fpSpread1.ActiveSheet.ColumnCount = 5
    ' Configure respective default styles.
    fpSpread1.ActiveSheet.DefaultStyle.BackColor = Color.LemonChiffon
    fpSpread1.ActiveSheet.DefaultStyle.ForeColor = Color.Red
    fpSpread1.ActiveSheet.DefaultStyle.CellType = New
FarPoint.Win.Spread.CellType.NumberCellType
    fpSpread1.ActiveSheet.DefaultStyle.HorizontalAlignment =
FarPoint.Win.Spread.CellHorizontalAlignment.Center
    fpSpread1.ActiveSheet.DefaultStyle.Border = New FarPoint.Win.LineBorder(Color.Green)
    For i As Integer = 0 To fpSpread1.ActiveSheet.RowCount - 1
        For j As Integer = 0 To fpSpread1.ActiveSheet.ColumnCount - 1
            fpSpread1.ActiveSheet.SetValue(i, j, i + j)
        Next
    Next
End Sub
```

Using the NamedStyleCollection Editor

1. In the Form window, click the Spread component or the Sheet object for which you want to create the style in the **NamedStyleCollection**. For the Spread component, in the **Appearance** category, select the **NamedStyles** property. For the Sheet object, in the **Misc** category, select the **NamedStyles** property.
2. Click on the button to launch the **NamedStyleCollection Editor**.
3. In the **NamedStyleCollection Editor**, select the **Add** tab.
4. Set the properties in the **Named Style Properties** list to create the style you want.
5. Set the **Name** property to specify the name for your custom style.
6. Click **OK** to close the editor.

7. Select the cells (or rows or columns) to apply the style to.
8. In the property window, set the **StyleName** to the custom named style previously added.

Using Conditional Formatting of Cells

You can customize the user interaction with individual cells (or a range of cells). You can use rules or conditional operators in the conditional format.

To customize this aspect of user interaction, you can perform the following tasks:

- **Creating Conditional Formatting with Rules**
- **Setting up Conditional Formatting of a Cell**

Creating Conditional Formatting with Rules

You can set the visual appearance of cells using rules. The following classes are available when creating conditional formatting with rules:

- **AverageConditionalFormattingRule Class (on-line documentation)**
- **BetweenValuesConditionalFormattingRule Class (on-line documentation)**
- **BlankConditionalFormattingRule Class (on-line documentation)**
- **DatabarConditionalFormattingRule Class (on-line documentation)**
- **FormulaConditionalFormattingRule Class (on-line documentation)**
- **ErrorConditionalFormattingRule Class (on-line documentation)**
- **IconSetConditionalFormattingRule Class (on-line documentation)**
- **PrePaintConditionalFormattingRule Class (on-line documentation)**
- **PrePaintTextConditionalFormattingRule Class (on-line documentation)**
- **TextConditionalFormattingRule Class (on-line documentation)**
- **ThreeColorScaleConditionalFormattingRule Class (on-line documentation)**
- **TimePeriodConditionalFormattingRule Class (on-line documentation)**
- **TopRankedValuesConditionalFormattingRule Class (on-line documentation)**
- **TwoColorScaleConditionalFormattingRule Class (on-line documentation)**
- **UnaryComparisonConditionalFormattingRule Class (on-line documentation)**
- **UniqueOrDuplicatedConditionalFormattingRule Class (on-line documentation)**

The average rule checks for values above or under the average. The cell value rule compares values. The date rule compares dates. The formula rule allows you to use formulas when checking the condition.

The scale rule uses a sliding color scale. For example if 1 is yellow and 50 is green, then 25 would be light green.

The specific text rule searches for text strings. The top 10 rule checks for values in the top or bottom of the range. The unique rule checks to see if the value is the only one of that value in the range (if the duplicate option is false). The duplicate rule checks for duplicate values.

The data bar rule displays a bar in the cell based on the cell value in the range. The icon set rule displays icons based on the values.

You can add rules with the **SetConditionalFormatting** ('SetConditionalFormatting Method' in the on-line documentation) method or the **ConditionalFormatting** class.

The following topics provide additional information about specific conditional formatting rules.

- **Color Scale Rules**
- **Data Bar Rule**
- **Highlighting Rules**

- **Icon Set Rule**
- **Top, Bottom, or Average Rules**

Color Scale Rules

Color scales are visual guides that help you understand data distribution and variation. A two-color scale compares a range of cells by using a gradation of two colors. The shade of the color represents higher or lower values. For example, in a green and red color scale, you can specify that higher value cells are closer to a green color and lower value cells are closer to a red color. You can specify the value type, value, and color for the minimum and maximum properties.

A three-color scale compares a range of cells by using a gradation of three colors. The shade of the color represents higher, middle, or lower values. For example, in a green, yellow, and red color scale, you can specify that higher value cells have a green color, middle value cells have a yellow color, and lower value cells have a red color. You can specify the value type, value, and color for the minimum, middle, and maximum properties.

The following image uses the three color rule. The A2 cell is a gradation of the middle and low color values.

	A	B	C	D
1	3		1	
2	2	10		
3				
4				
5				

Using Code

Set the properties of the **TwoColorScaleConditionalFormattingRule** (**'TwoColorScaleConditionalFormattingRule Class' in the on-line documentation**) class or the **ThreeColorScaleConditionalFormattingRule** (**'ThreeColorScaleConditionalFormattingRule Class' in the on-line documentation**) class and then apply the formatting.

Example

This example code creates a three color rule and uses the **SetConditionalFormatting** (**'SetConditionalFormatting Method' in the on-line documentation**) method to apply the rule.

C#

```
private void Form1_Load(object sender, EventArgs e)
{
    fpSpread1.Sheets[0].Cells[0, 0].Value = 3;
    fpSpread1.Sheets[0].Cells[1, 0].Value = 2;
    fpSpread1.Sheets[0].Cells[1, 1].Value = 10;
    fpSpread1.Sheets[0].Cells[0, 2].Value = 1;
}

private void button1_Click(object sender, EventArgs e)
{
    FarPoint.Win.Spread.Model.CellRange celRange1 = new
    FarPoint.Win.Spread.Model.CellRange(0, 0, 3, 3);
    FarPoint.Win.Spread.ThreeColorScaleConditionalFormattingRule rule = new
    FarPoint.Win.Spread.ThreeColorScaleConditionalFormattingRule(Color.Aqua, Color.Bisque,
    Color.BlueViolet);
    fpSpread1.Sheets[0].SetConditionalFormatting(new FarPoint.Win.Spread.Model.CellRange[]
    { celRange1 }, rule);
}
```

```
}

```

VB

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    fpSpread1.Sheets(0).Cells(0, 0).Value = 3
    fpSpread1.Sheets(0).Cells(1, 0).Value = 2
    fpSpread1.Sheets(0).Cells(1, 1).Value = 10
    fpSpread1.Sheets(0).Cells(0, 2).Value = 1
End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim celRange1 As New FarPoint.Win.Spread.Model.CellRange(0, 0, 3, 3)
    Dim rule As New
FarPoint.Win.Spread.ThreeColorScaleConditionalFormattingRule(Color.Aqua, Color.Bisque,
Color.BlueViolet)
    fpSpread1.Sheets(0).SetConditionalFormatting(New
FarPoint.Win.Spread.Model.CellRange() {celRange1}, rule)
End Sub

```

Using the Spread Designer

1. In the work area, select the cell or cells for which you want to set the conditional format.
2. Under the **Home** menu, select the **Conditional Formatting** icon in the **Style** section, then select the **Color Scales** option, and then choose the color set.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Data Bar Rule

The data bar rule uses a bar that is displayed as the background for each cell. The length of the bar corresponds to the size of the data relative to the other data in the worksheet. The longer the bar, the greater the value in the cell.

The following image displays data bars in a cell range:

	A	B
1	3	
2	2	
3	10	
4	1	
5		

You can specify the value type and the value to compare in the conditional format.

Value Type Description

- | | |
|---------|---|
| Percent | The minimum value in the range of cells that the conditional formatting rule applies to plus x percent of the difference between the maximum and minimum values in the range of cells that the conditional formatting rule applies to. For example, if the minimum and maximum values in the range are 1 and 10 respectively, and x is 10, then the value is 1.9. |
| Highest | The maximum value in the range of cells that the conditional formatting rule applies to. |

Value

Lowest Value	The minimum value in the range of cells that the conditional formatting rule applies to.
Formula	The result of the formula determines the minimum or maximum value of the cell range that the rule applies to. If the result is not numeric, it is treated as zero.
Percentile	The result of the function percentile applied to the range with x.
Automatic	The smaller or larger or the minimum or maximum value in the range of cells that the conditional format applies to.
Number	Number, date, or time value in the range of cells that the conditional formatting rule applies to.

Valid percentiles are from 0 (zero) to 100. A percentile cannot be used if the range of cells contains more than 8,191 data points.

Use a percentile when you want to visualize a group of high values (such as the top 20th percentile) in one data bar and low values (such as the bottom 20th percentile) in another data bar. This is useful if you have extreme values that might skew the visualization of your data.

Valid percent values are from 0 (zero) to 100. Percent values should not use a percent sign. Use a percentage when you want to visualize all values proportionally because the distribution of values is proportional.

Start formulas with an equal sign (=). Invalid formulas result in no formatting applied.

The minimum and maximum types can be different. The **Maximum ('Maximum Property' in the on-line documentation)** property should not be set to a **ConditionalFormattingValue** value such as **ConditionalFormattingValueType.Min** or **ConditionalFormattingValueType.AutoMin**. An exception will occur in this case. The **Minimum ('Minimum Property' in the on-line documentation)** property should not be set to a **ConditionalFormattingValue** value such as **ConditionalFormattingValueType.Max** or **ConditionalFormattingValueType.AutoMax**. An exception will occur in this case.

You can also specify borders, colors, and an axis.

Using Code

Set the properties of the data bar rule class and then apply the formatting.

Example

This example code creates a data bar rule and uses the **SetConditionalFormatting ('SetConditionalFormatting Method' in the on-line documentation)** method to apply the rule.

C#

```
private void Form1_Load(object sender, EventArgs e)
{
    fpSpread1.Sheets[0].Cells[0, 0].Value = 3;
    fpSpread1.Sheets[0].Cells[1, 0].Value = 2;
    fpSpread1.Sheets[0].Cells[2, 0].Value = 10;
    fpSpread1.Sheets[0].Cells[3, 0].Value = 1;
}

private void button1_Click(object sender, EventArgs e)
{
    FarPoint.Win.Spread.DatabarConditionalFormattingRule d = new
FarPoint.Win.Spread.DatabarConditionalFormattingRule();
    d.BorderColor = Color.Red;
    d.ShowBorder = true;
    d.Minimum = new FarPoint.Win.Spread.ConditionalFormattingValue(0,
```

```

FarPoint.Win.Spread.ConditionalFormattingValueType.Number);
    d.Maximum = new FarPoint.Win.Spread.ConditionalFormattingValue(15,
FarPoint.Win.Spread.ConditionalFormattingValueType.Max);
    fpSpread1.ActiveSheet.SetConditionalFormatting(0, 0, 4, 1, d);
}

```

VB

```

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    fpSpread1.Sheets(0).Cells(0, 0).Value = 3
    fpSpread1.Sheets(0).Cells(1, 0).Value = 2
    fpSpread1.Sheets(0).Cells(2, 0).Value = 10
    fpSpread1.Sheets(0).Cells(3, 0).Value = 1
End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim d As New FarPoint.Win.Spread.DatabarConditionalFormattingRule()
    d.BorderColor = Color.Red
    d.ShowBorder = True
    d.Minimum = New FarPoint.Win.Spread.ConditionalFormattingValue(0,
FarPoint.Win.Spread.ConditionalFormattingValueType.Number)
    d.Maximum = New FarPoint.Win.Spread.ConditionalFormattingValue(15,
FarPoint.Win.Spread.ConditionalFormattingValueType.Max)
    fpSpread1.ActiveSheet.SetConditionalFormatting(0, 0, 4, 1, d)
End Sub

```

Using the Spread Designer

1. In the work area, select the cell or cells for which you want to set the conditional format.
2. Under the **Home** menu, select the **Conditional Formatting** icon in the **Style** section, then select the **Data Bars** option, and then choose the color set.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Icon Set Rule

You can set rules that display certain icons when a cell value is greater than, equal to, or less than a value.

You can use built-in icon sets for the rule. You can also specify individual icons to use in the icon set with the **IconRuleSet** (**'IconRuleSet Property' in the on-line documentation**) property and the **IconSetConditionalFormattingRule** (**'IconSetConditionalFormattingRule Class' in the on-line documentation**) class. You can use custom icons with the **AddIcon** (**'AddIcon Method' in the on-line documentation**) method and the **CustomIconContainer** (**'CustomIconContainer Property' in the on-line documentation**) property.

The following figure illustrates cells that display the built-in icons.

	A	B
1	8	
2	5	
3	10	
4	1	
5		

Using Code

1. Set the properties of the icon set rule class.
2. Apply the formatting.

Example

This example code creates an icon set rule and uses the **SetConditionalFormatting** (**'SetConditionalFormatting Method' in the on-line documentation**) method to apply the rule.

C#

```
private void Form1_Load(object sender, EventArgs e)
{
    fpSpread1.Sheets[0].Cells[0, 0].Value = 8;
    fpSpread1.Sheets[0].Cells[1, 0].Value = 5;
    fpSpread1.Sheets[0].Cells[2, 0].Value = 10;
    fpSpread1.Sheets[0].Cells[3, 0].Value = 1;
}

private void button1_Click(object sender, EventArgs e)
{
    FarPoint.Win.Spread.Model.CellRange celRange1 = new FarPoint.Win.Spread.Model.CellRange(0, 0, 4, 1);
    FarPoint.Win.Spread.IconSetConditionalFormattingRule rule = new
    FarPoint.Win.Spread.IconSetConditionalFormattingRule(FarPoint.Win.Spread.ConditionalFormattingIconSetStyle.ThreeRimmedTrafficLights
    );
    fpSpread1.Sheets[0].SetConditionalFormatting(new FarPoint.Win.Spread.Model.CellRange[] { celRange1 }, rule);
}
```

VB

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    fpSpread1.Sheets(0).Cells(0, 0).Value = 8
    fpSpread1.Sheets(0).Cells(1, 0).Value = 5
    fpSpread1.Sheets(0).Cells(2, 0).Value = 10
    fpSpread1.Sheets(0).Cells(3, 0).Value = 1
End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    Dim celRange1 As New FarPoint.Win.Spread.Model.CellRange(0, 0, 4, 1)
    Dim rule As New
    FarPoint.Win.Spread.IconSetConditionalFormattingRule(FarPoint.Win.Spread.ConditionalFormattingIconSetStyle.ThreeRimmedTrafficLights)
    fpSpread1.Sheets(0).SetConditionalFormatting(New FarPoint.Win.Spread.Model.CellRange() {celRange1}, rule)
End Sub
```

Using the Spread Designer

1. In the work area, select the cell or cells for which you want to set the conditional format.
2. Under the **Home** menu, select the **Conditional Formatting** icon in the **Style** section, then select the **Icon Sets** option, and then choose the icon set.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Highlighting Rules

You can use this rule to highlight data that meets one of the following conditions:

- is greater than a value
- is less than a value
- is between a high and low value
- is equal to a value
- contains a specific value
- is a date that occurs in a particular range
- is either unique or duplicated elsewhere in the worksheet

After you choose one of the options above, enter a value or formula against which each cell is compared. If the cell data satisfies that criteria, then the formatting is applied.

You can select a predefined highlight style or create a custom highlight style. The following rules are highlight style rules:

- **BetweenValuesConditionalFormattingRule** ('BetweenValuesConditionalFormattingRule Class' in the on-line documentation)
- **BlankConditionalFormattingRule** ('BlankConditionalFormattingRule Class' in the on-line documentation)
- **ErrorConditionalFormattingRule** ('ErrorConditionalFormattingRule Class' in the on-line documentation)
- **FormulaConditionalFormattingRule** ('FormulaConditionalFormattingRule Class' in the on-line documentation)
- **TextConditionalFormattingRule** ('TextConditionalFormattingRule Class' in the on-line documentation)

documentation)

- **TimePeriodConditionalFormattingRule** ('TimePeriodConditionalFormattingRule Class' in the on-line documentation)
- **UnaryComparisonConditionalFormattingRule** ('UnaryComparisonConditionalFormattingRule Class' in the on-line documentation)
- **UniqueOrDuplicatedConditionalFormattingRule** ('UniqueOrDuplicatedConditionalFormattingRule Class' in the on-line documentation)

This figure illustrates the following example.

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

Using Code

1. Set the properties of the rule class.
2. Apply the formatting.

Example

This example code creates the between values rule and uses the **SetConditionalFormatting** ('SetConditionalFormatting Method' in the on-line documentation) method to apply the rule.

C#

```
private void Form1_Load(object sender, EventArgs e)
{
    fpSpread1.Sheets[0].Cells[0, 0].Value = 3;
    fpSpread1.Sheets[0].Cells[1, 0].Value = 2;
    fpSpread1.Sheets[0].Cells[1, 1].Value = 5;
    fpSpread1.Sheets[0].Cells[0, 2].Value = 1;
}

private void button1_Click(object sender, EventArgs e)
{
    FarPoint.Win.Spread.BetweenValuesConditionalFormattingRule between = new
FarPoint.Win.Spread.BetweenValuesConditionalFormattingRule(true, 10, false, 20, false);
    between.FirstValue = 10;
    between.SecondValue = 20;
    between.IsNotBetween = true;
    between.BackColor = Color.Bisque;
    fpSpread1.ActiveSheet.SetConditionalFormatting(1, 1, between);
}
```

VB

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    fpSpread1.Sheets(0).Cells(0, 0).Value = 3
    fpSpread1.Sheets(0).Cells(1, 0).Value = 2
    fpSpread1.Sheets(0).Cells(1, 1).Value = 5
    fpSpread1.Sheets(0).Cells(0, 2).Value = 1

```

```
End Sub
```

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim between As New
FarPoint.Win.Spread.BetweenValuesConditionalFormattingRule(True, 10, False, 20, False)
    between.FirstValue = 10
    between.SecondValue = 20
    between.IsNotBetween = True
    between.BackColor = Color.Bisque
    fpSpread1.ActiveSheet.SetConditionalFormatting(1, 1, between)
End Sub
```

Using the Spread Designer

1. In the work area, select the cell or cells for which you want to set the conditional format.
2. Under the **Home** menu, select the **Conditional Formatting** icon in the **Style** section, then select the **Highlight Cells Rules** option, and then choose the condition.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Top, Bottom, or Average Rules

The top or bottom rules apply formatting to cells whose values fall in the top or bottom percent. The top ranked rule specifies the top or bottom values. The average rule applies to the greater or lesser average value of the entire range.

The following figure shows a top rule that sets the style for the above average value in the range of values; the code to create the example is provided in the example.

	A	B	C
1	3		1
2	2	10	

The following options are available:

- top 10
- top 10%
- bottom 10
- bottom 10%
- above average
- below average

Using Code

1. Set the properties of the rule class.
2. Apply the formatting.

Example

This example code creates an average rule and uses the **SetConditionalFormatting ('SetConditionalFormatting Method' in the on-line documentation)** method to apply the rule.

C#

```
private void Form1_Load(object sender, EventArgs e)
```

```

    {
        fpSpread1.Sheets[0].Cells[0, 0].Value = 3;
        fpSpread1.Sheets[0].Cells[1, 0].Value = 2;
        fpSpread1.Sheets[0].Cells[1, 1].Value = 10;
        fpSpread1.Sheets[0].Cells[0, 2].Value = 1;
    }
    private void button1_Click(object sender, EventArgs e)
    {
        //Average CF
        FarPoint.Win.Spread.AverageConditionalFormattingRule average = new
FarPoint.Win.Spread.AverageConditionalFormattingRule(true, true);
        average.IsAbove = true;
        average.IsIncludeEquals = true;
        average.StandardDeviation = 2;
        average.FontStyle = new
FarPoint.Win.Spread.SpreadFontStyle(FarPoint.Win.Spread.RegularBoldItalicFontStyle.Bold);
        fpSpread1.ActiveSheet.SetConditionalFormatting(1, 1, average);
    }

```

VB

```

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    fpSpread1.Sheets(0).Cells(0, 0).Value = 3
    fpSpread1.Sheets(0).Cells(1, 0).Value = 2
    fpSpread1.Sheets(0).Cells(1, 1).Value = 10
    fpSpread1.Sheets(0).Cells(0, 2).Value = 1
End Sub
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    'Average CF
    Dim average As New FarPoint.Win.Spread.AverageConditionalFormattingRule(True,
True)
    average.IsAbove = True
    average.IsIncludeEquals = True
    average.StandardDeviation = 2
    average.FontStyle = New
FarPoint.Win.Spread.SpreadFontStyle(FarPoint.Win.Spread.RegularBoldItalicFontStyle.Bold)
    fpSpread1.ActiveSheet.SetConditionalFormatting(1, 1, average)
End Sub

```

Using the Spread Designer

1. In the work area, select the cell or cells for which you want to set the conditional format.
2. Under the **Home** menu, select the **Conditional Formatting** icon in the **Style** section, then select the **Top/Bottom Rules** option, and then choose the condition.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting up Conditional Formatting of a Cell

You can set up conditional formats within cells that determine the formatting of the cell based on the outcome of a conditional statement. You can specify various formatting options such as borders and colors to apply if the condition statement is valid, that is, if the operation is satisfied.

For example, you may want to change the background color of a cell based on the value of the cell. If the value is below

100 then the background color would be changed to red. The condition statement is "less than 100" and consists of a comparison operator "less than" and a condition, in this case a single constant "100". The condition can be a constant (expressed as a string) or an expression. Some condition statements have two conditions and an operator: for instance, if the cell value is between 0 and 100, then change the background color. In this case, the comparison operator is "between" and the first condition is 0 and the last condition is 100. For more information about the possible style settings, refer to **Creating and Applying a Style for Cells**.

If two conditional formats are set to the same cell, the second conditional format takes effect.

The conditional formatting can be done using the **ConditionalFormatting** class and any of the following members of the **SheetView ('SpreadView Class' in the on-line documentation)** class:

- **GetConditionalFormatting ('GetConditionalFormatting Method' in the on-line documentation)** method
- **SetConditionalFormatting ('SetConditionalFormatting Method' in the on-line documentation)** method
- **ClearConditionalFormatting ('ClearConditionalFormatting Method' in the on-line documentation)** method

When you use the **GetConditionalFormatting ('GetConditionalFormatting Method' in the on-line documentation)** method, the conditions, operator, and style information are returned as a **ConditionalFormatting** object. The first condition can be either a string or expression (**FirstCondition** or **FirstConditionExpression**.) Similarly, the last condition can be a string or expression (**LastCondition** or **LastConditionExpression**). If only one condition is set, it is in the **FirstCondition** and the **LastCondition** is null. The **ComparisonOperator** is the comparison operator for the conditional format. The style settings to apply to the cell when the condition statement is true are set as an object.

Refer to the following code examples to see how to set conditional formatting for a range of cells that would result in different background colors, for instance, as shown in the following figure.

	A	B	C	D	E	F
1	2	22	42	62	82	102
2						

For some cell types that allow input of multiple data types, such as general cell type, conditional formatting works whether you type in numbers or strings. For example, if you have conditional formatting set for values between various ranges such as 10 to 20 and 20 to 30, then typing 16 results in the formatting for the range 10 to 20. If you then type 16m, the cell treats this like a string and since "16m" is between strings "10" and "20", the conditional formatting still applies.

Use the **ClearConditionalFormatting ('ClearConditionalFormatting Method' in the on-line documentation)** method to clear only the conditional formats of a cell without affecting the other formatting or the contents of the cell.

Using Code

1. Define formatting rule.
2. Set conditional formatting for a range of cells.

Example

This example code defines the conditional formatting rule for a range of cells and changes the coloring of the cells based on the specified rule. To see how it works, type a number in cell B2, then either change cells or leave edit mode to see if the formatting is applied.

C#

```
FarPoint.Win.Spread.NamedStyle styleCold = new FarPoint.Win.Spread.NamedStyle();
FarPoint.Win.Spread.NamedStyle styleCool = new FarPoint.Win.Spread.NamedStyle();
```

```
FarPoint.Win.Spread.NamedStyle styleMild = new FarPoint.Win.Spread.NamedStyle();
FarPoint.Win.Spread.NamedStyle styleWarm = new FarPoint.Win.Spread.NamedStyle();
FarPoint.Win.Spread.NamedStyle styleHot = new FarPoint.Win.Spread.NamedStyle();
styleCold.BackColor = Color.Blue;
styleCold.ForeColor = Color.White;
styleCool.BackColor = Color.Cyan;
styleMild.BackColor = Color.Lime;
styleWarm.BackColor = Color.Yellow;
styleHot.BackColor = Color.Red;
for (int col = 0; col < 6; col++)
{
    fpSpread1.ActiveSheet.SetConditionalFormat(0, col, styleCold,
FarPoint.Win.Spread.ComparisonOperator.LessThanOrEqualTo, "32");
    fpSpread1.ActiveSheet.SetConditionalFormat(0, col, styleCool,
FarPoint.Win.Spread.ComparisonOperator.Between, "32", "55");
    fpSpread1.ActiveSheet.SetConditionalFormat(0, col, styleMild,
FarPoint.Win.Spread.ComparisonOperator.Between, "55", "75");
    fpSpread1.ActiveSheet.SetConditionalFormat(0, col, styleWarm,
FarPoint.Win.Spread.ComparisonOperator.Between, "75", "85");
    fpSpread1.ActiveSheet.SetConditionalFormat(0, col, styleHot,
FarPoint.Win.Spread.ComparisonOperator.GreaterThan, "85");
}
```

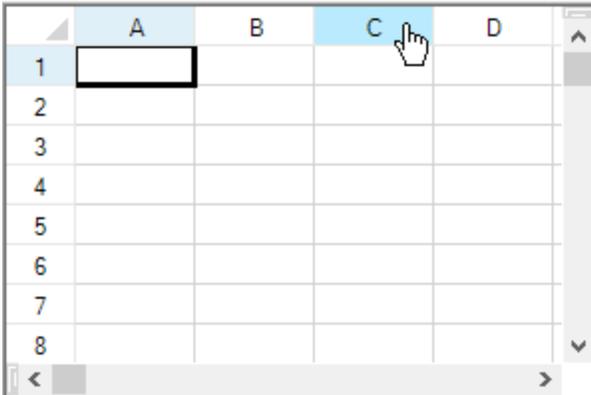
VB

```
Dim styleCold As New FarPoint.Win.Spread.NamedStyle()
Dim styleCool As New FarPoint.Win.Spread.NamedStyle()
Dim styleMild As New FarPoint.Win.Spread.NamedStyle()
Dim styleWarm As New FarPoint.Win.Spread.NamedStyle()
Dim styleHot As New FarPoint.Win.Spread.NamedStyle()
styleCold.BackColor = Color.Blue
styleCold.ForeColor = Color.White
styleCool.BackColor = Color.Cyan
styleMild.BackColor = Color.Lime
styleWarm.BackColor = Color.Yellow
styleHot.BackColor = Color.Red
For col As Integer = 0 To 5
    FpSpread1.ActiveSheet.SetConditionalFormat(0, col, styleCold,
FarPoint.Win.Spread.ComparisonOperator.LessThanOrEqualTo, "32")
    FpSpread1.ActiveSheet.SetConditionalFormat(0, col, styleCool,
FarPoint.Win.Spread.ComparisonOperator.Between, "32", "55")
    FpSpread1.ActiveSheet.SetConditionalFormat(0, col, styleMild,
FarPoint.Win.Spread.ComparisonOperator.Between, "55", "75")
    FpSpread1.ActiveSheet.SetConditionalFormat(0, col, styleWarm,
FarPoint.Win.Spread.ComparisonOperator.Between, "75", "85")
    FpSpread1.ActiveSheet.SetConditionalFormat(0, col, styleHot,
FarPoint.Win.Spread.ComparisonOperator.GreaterThan, "85")
Next col
```

Customizing the Display of the Pointer

You can set the cursor or pointer to appear differently for different parts of the display. To determine the pointer to display, use the **GetCursor** ('**GetCursor Method**' in the on-line documentation) method and **SetCursor** ('**SetCursor Method**' in the on-line documentation) method and the **CursorType** ('**CursorType Enumeration**' in the on-line documentation) enumeration. The code for setting the pointer to change when it is

over a header cell, as shown in this figure, is given in the example below.



The Spread component uses one pointer for locked cells (**CursorType ('CursorType Enumeration' in the on-line documentation)** enumeration equal to Locked) and one pointer for unlocked cells (**CursorType ('CursorType Enumeration' in the on-line documentation)** enumeration equal to Normal). The component does not support different pointers for each cell type.

Cell Types and Reserved Locations

The built-in hyperlink cell type by default uses the hand pointer over the link area. For more information, refer to **Setting a Hyperlink Cell**.

Some cell types (for example, button, check box, combo box, and hyperlink) reserve areas within the cell that require special mouse processing. For example, clicking a mouse button while over the drop-down button in a combo box cell immediately enters edit mode. The location of the special area and the pointer used over the special area is determined by the **IsReservedLocation ('IsReservedLocation Method' in the on-line documentation)** and **GetReservedCursor ('GetReservedCursor Method' in the on-line documentation)** methods in the cell type classes. If you do not like the pointer supplied by a built-in cell type class then you could derive a class from the built-in cell type class and override the **GetReservedCursor ('GetReservedCursor Method' in the on-line documentation)** method.

The **GetReservedCursor ('GetReservedCursor Method' in the on-line documentation)** method is only called for locations where **IsReservedLocation ('IsReservedLocation Method' in the on-line documentation)** returns non-null. By returning null (Nothing in VB) or non-null, the **IsReservedLocation** method essentially divides the cell rectangle into two subregions (normal region and reserved region). In the normal subregion, a mouse down is processed by the Spread component and starts a cell selection. Since the mouse down is processed by the Spread component, the mouse pointer is determined by the Spread component (that is, **GetReservedCursor** is not called). In the reserved subregion, a mouse down immediately starts a cell edit and the mouse down gets passed to the cell editor for processing. Since the mouse down is processed by the cell editor, the mouse pointer is determined by the cell type via the **GetReservedCursor** method. While you can supply cell type specific pointers for the reserved subregion, you can not supply cell type specific pointers for the normal subregions.

Example

This example sets the pointer to display as a hand as shown in the figure above.

C#

```
private void Form1_Load(object sender, System.EventArgs e){
    // Change the pointer shape on column headers.
    fpSpread1.SetCursor(FarPoint.Win.Spread.CursorType.ColumnHeader,
        System.Windows.Forms.Cursors.Hand);
}
```

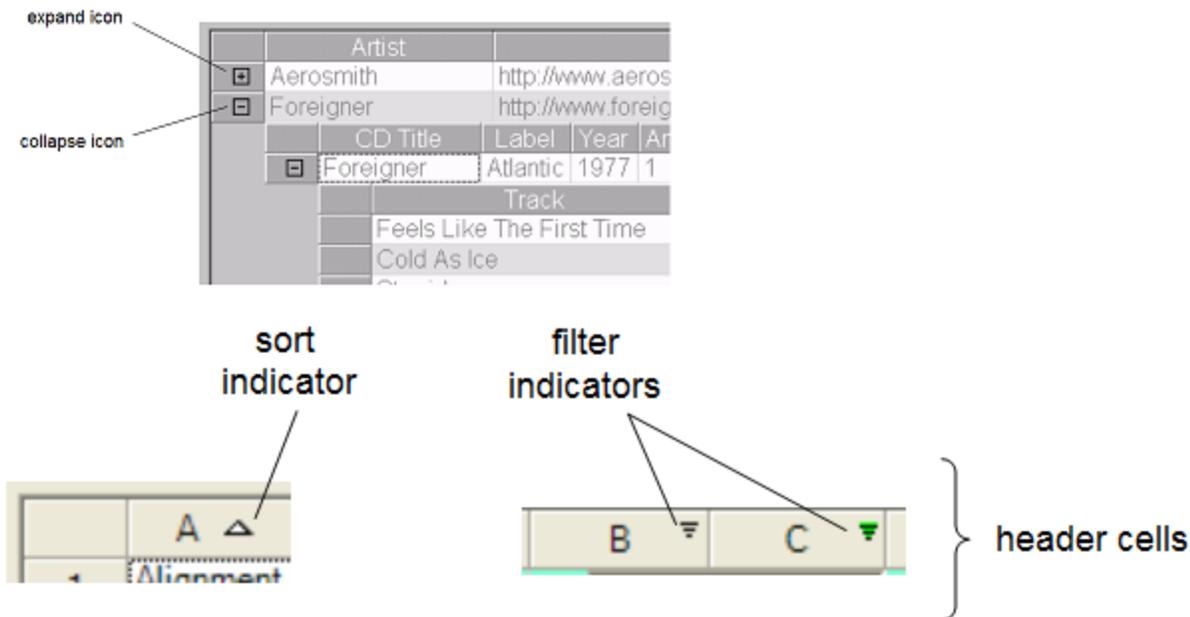
VB

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.Load
    ' Change the pointer shape on column headers.
    fpSpread1.SetCursor(FarPoint.Win.Spread.CursorType.ColumnHeader,
        System.Windows.Forms.Cursors.Hand)
End Sub
```

Customizing the User Interface Images

You can customize various images in the user interface by selecting your own custom images and applying them to replace default images. The parts of the user interface that you can customize are:

- Hierarchy (expanding and collapsing) icons
- Filtering indicators
- Sorting indicators
- Row selector



To determine the images for these parts of the user interface, use the **GetImage** ('**GetImage Method**' in the on-line documentation) and **SetImage** ('**SetImage Method**' in the on-line documentation) methods in the **SpreadView** ('**SpreadView Class**' in the on-line documentation) class. The various fields of the **SpreadView** ('**SpreadView Class**' in the on-line documentation) class allow you to specify to which part of the interface the graphic image is assigned. These images can be set at run time only, not at design time.

For an example of these methods refer to the examples given for the individual fields:

- **CollapseImage** ('**CollapseImage Field**' in the on-line documentation)
- **CollapseImageDisabled** ('**CollapseImageDisabled Field**' in the on-line documentation)
- **ExpandImage** ('**ExpandImage Field**' in the on-line documentation)
- **ExpandImageDisabled** ('**ExpandImageDisabled Field**' in the on-line documentation)
- **FilterActive** ('**FilterActive Field**' in the on-line documentation)
- **FilterActiveDisabled** ('**FilterActiveDisabled Field**' in the on-line documentation)
- **FilterBarFilterActive** ('**FilterBarFilterActive Field**' in the on-line documentation)

- **FilterBarFilterDateTime** ('FilterBarFilterDateTime Field' in the on-line documentation)
- **FilterBarFilterInactive** ('FilterBarFilterInactive Field' in the on-line documentation)
- **FilterInactive** ('FilterInactive Field' in the on-line documentation)
- **FilterInactiveDisabled** ('FilterInactiveDisabled Field' in the on-line documentation)
- **RowSelectorImage** ('RowSelectorImage Field' in the on-line documentation)
- **RowSelectorImageDisabled** ('RowSelectorImageDisabled Field' in the on-line documentation)
- **SortAscendingImage** ('SortAscendingImage Field' in the on-line documentation)
- **SortAscendingImageDisabled** ('SortAscendingImageDisabled Field' in the on-line documentation)
- **SortDescendingImage** ('SortDescendingImage Field' in the on-line documentation)
- **SortDescendingImageDisabled** ('SortDescendingImageDisabled Field' in the on-line documentation)
- **SortUnsortedImage** ('SortUnsortedImage Field' in the on-line documentation)
- **SortUnsortedImageDisabled** ('SortUnsortedImageDisabled Field' in the on-line documentation)

To reset an image back to a Spread default image, simply set the image value to null in the **SetImage** ('SetImage Method' in the on-line documentation) method.

Another way to set the images for the filtering and sorting indicators, is to override the **PaintFilterIndicator** ('PaintFilterIndicator Method' in the on-line documentation) and **PaintSortIndicator** ('PaintSortIndicator Method' in the on-line documentation) methods in the **CellTypeColumnHeaderRenderer** ('ColumnHeaderRenderer Class' in the on-line documentation) class. For more information, refer to:

- **Setting the Appearance of Filter Indicators**
- **Setting the Appearance of Sort Indicators**

For more information about features, see the following topics:

- **Working with Hierarchical Data Display**
- **Allowing the User to Automatically Sort Rows**
- **Understanding Simple Row Filtering**

Using XP Themes with the Component

You can set support for XP themes for the cells and graphical elements in the Spread component. The **VisualStyles** ('VisualStyles Property' in the on-line documentation) property for the Spread component causes the cells to have the appearance of the XP theme. You can create a manifest file so that the scroll bars have a theme appearance.

By default the **VisualStyles** ('VisualStyles Property' in the on-line documentation) property is set to Auto, so the graphical cell types paint the way the theme is set to paint (for example the button is themed so you cannot set a background color with the **BackColor** property). You either need to turn off VisualStyles for the Spread component or create a custom cell type where you override the **PaintCell** and **GetEditorControl** methods and then set the **VisualStyles** property of the Appearance object to off. A third alternative is to leave it "on" for the Spread component but turn it "off" for the individual control (such as the FpProgress control for the progress indicator cell).

Applying a sheet skin causes the **VisualStyles** property to be set to false.

If you are on Windows XP, then you need to add the following line of code to turn off the XP themes in the Spread.

C#

```
fpSpread1.VisualStyles = FarPoint.Win.VisualStyles.Off;
```

Visual Basic

```
FpSpread1.VisualStudio = FarPoint.Win.VisualStudio.Off
```

**Note:**

- Setting the **VisualStyles** property of the Spread to Off should return the look of the Spread to the classic look.
- Buttons are not displayed as expected if you have changed the SelectionStyle and have VisualStyles on. The problem is that when VisualStyles are on certain cell types ignore certain settings, such as the button ignoring the setting of the **ButtonColor** property. You would need to take that into account and possibly set the SelectionForeColor, for example, to something different. The SelectionColors setting for SelectionStyle is an older style and mixing it with XP themes is not recommended.
- The **VisualStyles** property is used on controls that Spread renders such as the button in the ButtonCellType. The scroll bars are child controls rendered by Visual Studio. To have them render with XP themes, you would need to set up a manifest for your application.

Using the Properties Window

1. Select the Spread component.
2. In the **Properties** window, select the **VisualStyles** property, and choose an option from the drop-down list.

Using Code

Add a line of code that allows the theme support by setting the **VisualStyles ('VisualStyles Property' in the on-line documentation)** property for the Spread component.

Example

This example sets the **VisualStyles ('VisualStyles Property' in the on-line documentation)** property to on to allow XP themes.

C#

```
fpSpread1.VisualStudio = FarPoint.Win.VisualStudio.On;
```

VB

```
fpSpread1.VisualStudio = FarPoint.Win.VisualStudio.On
```

Using the Spread Designer

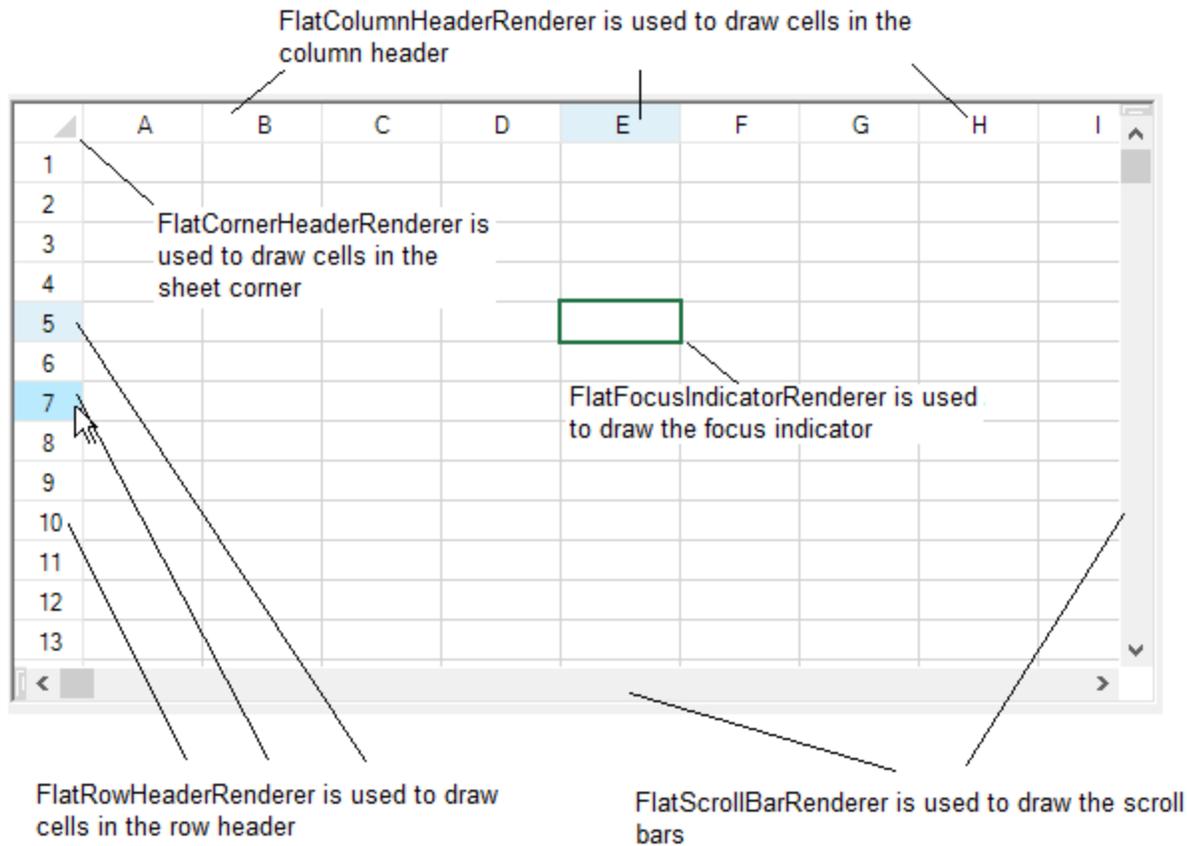
1. Select **Sheet** from the drop-down list located on the top right side of the Designer.
2. From the **Appearance** section, select an option for the **VisualStyles** property.
3. From the **File** menu, select **Save and Exit** to save the changes.

Customizing the Renderers

You can customize the renderers used to create the default styles.

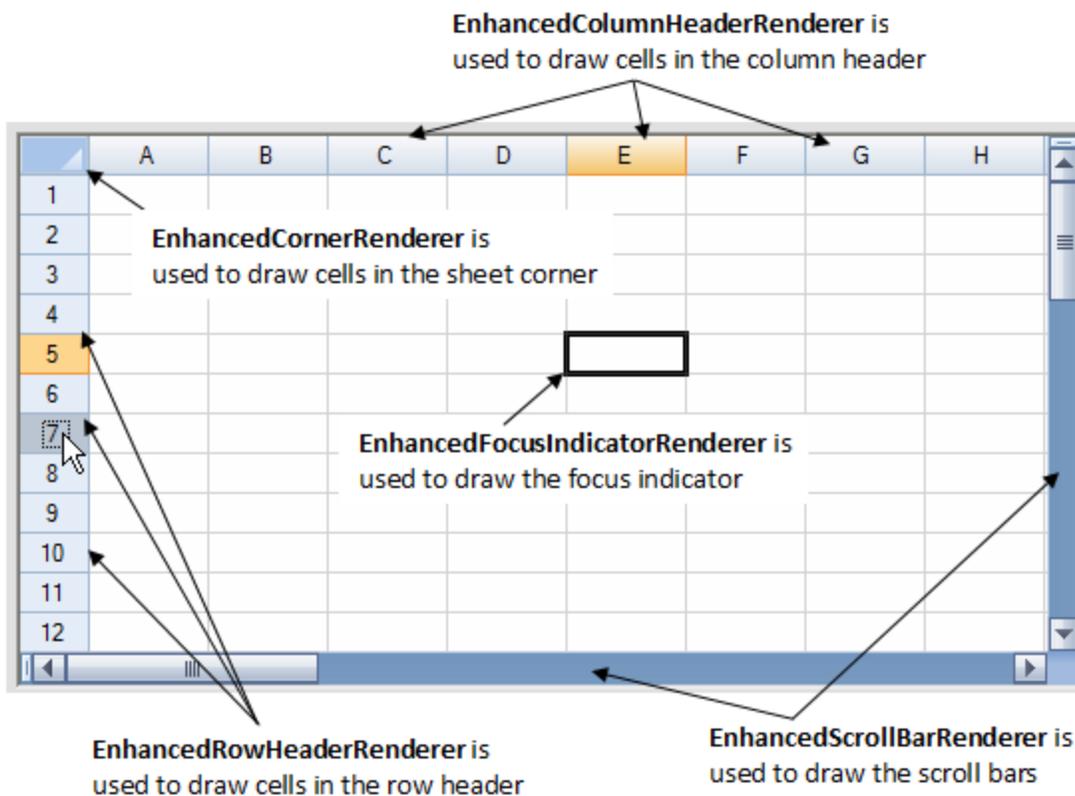
The Office2013 or Office2016 style uses the **FlatCornerHeaderRenderer ('FlatCornerHeaderRenderer Class' in the on-line documentation)**, **FlatColumnHeaderRenderer ('FlatColumnHeaderRenderer Class' in the on-line documentation)**, **FlatRowHeaderRenderer ('FlatRowHeaderRenderer Class' in the on-line documentation)**, **FlatScrollBarRenderer ('FlatScrollBarRenderer Class' in the on-line documentation)**,

and **FlatFocusIndicatorRenderer** ('FlatFocusIndicatorRenderer Class' in the on-line documentation) classes.



The default style uses the **ColumnHeaderDefaultEnhanced** ('ColumnHeaderDefaultEnhanced Field' in the on-line documentation), **CornerDefaultEnhanced** ('CornerDefaultEnhanced Field' in the on-line documentation), **CornerFooterDefaultEnhanced** ('CornerFooterDefaultEnhanced Field' in the on-line documentation), **FilterBarDefaultEnhanced** ('FilterBarDefaultEnhanced Field' in the on-line documentation), and **RowHeaderDefaultEnhanced** ('RowHeaderDefaultEnhanced Field' in the on-line documentation) fields.

The Office2007 style uses the **EnhancedCornerRenderer** ('EnhancedCornerRenderer Class' in the on-line documentation), **EnhancedFocusIndicatorRenderer** ('EnhancedFocusIndicatorRenderer Class' in the on-line documentation), **EnhancedColumnHeaderRenderer** ('EnhancedColumnHeaderRenderer Class' in the on-line documentation), **EnhancedScrollBarRenderer** ('EnhancedScrollBarRenderer Class' in the on-line documentation), and **EnhancedRowHeaderRenderer** ('EnhancedRowHeaderRenderer Class' in the on-line documentation) classes.



The classic style uses the **ColumnHeaderRenderer** ('ColumnHeaderRenderer Class' in the on-line documentation), **RowHeaderRenderer** ('RowHeaderRenderer Class' in the on-line documentation), and **CornerRenderer** ('CornerRenderer Class' in the on-line documentation) classes.

Using Code

1. Create a new renderer and set the renderer properties.
2. Set the renderer for the default style area such as column footer.
3. Apply the new corner styles to the control.

Example

This example code customizes the renderers for the column header and footer, row header, corner header, and corner footer.

C#

```
//header/footer column
fpSpread1.ActiveSheet.ColumnFooter.Visible = true;
fpSpread1.ActiveSheet.ColumnFooter.RowCount = 3;
fpSpread1.ActiveSheet.ColumnHeader.RowCount = 3;
FarPoint.Win.Spread.CellType.FlatColumnHeaderRenderer flatcolumnheader = new
FarPoint.Win.Spread.CellType.FlatColumnHeaderRenderer();
fpSpread1.ActiveSheet.ColumnHeader.DefaultStyle.Renderer = flatcolumnheader;
FarPoint.Win.Spread.CellType.FlatColumnFooterRenderer flatcolumnfooter = new
FarPoint.Win.Spread.CellType.FlatColumnFooterRenderer();
fpSpread1.ActiveSheet.ColumnFooter.DefaultStyle.Renderer = flatcolumnfooter;

//header row
```

```

fpSpread1.ActiveSheet.RowHeader.ColumnCount = 3;
FarPoint.Win.Spread.CellType.FlatRowHeaderRenderer flatrowheader = new
FarPoint.Win.Spread.CellType.FlatRowHeaderRenderer();
fpSpread1.ActiveSheet.RowHeader.DefaultStyle.Renderer = flatrowheader;

//sheet corner header render
FarPoint.Win.Spread.CellType.FlatCornerHeaderRenderer flatconrnerheader = new
FarPoint.Win.Spread.CellType.FlatCornerHeaderRenderer();
fpSpread1.ActiveSheet.SheetCorner.DefaultStyle.Renderer = flatconrnerheader;

//sheet corner footer render
FarPoint.Win.Spread.SpreadSkin a1 = new
FarPoint.Win.Spread.SpreadSkin(FarPoint.Win.Spread.DefaultSpreadSkins.Default);
a1.Apply(fpSpread1);
fpSpread1.ActiveSheet.ColumnFooter.Visible = true;
FarPoint.Win.Spread.CellType.FlatCornerFooterRenderer flatconrnerfooter = new
FarPoint.Win.Spread.CellType.FlatCornerFooterRenderer();
flatconrnerfooter.NormalTriangleColor = Color.Aquamarine;
FarPoint.Win.Spread.NamedStyle conner = new FarPoint.Win.Spread.NamedStyle("conner",
"HeaderDefault");
conner.BackColor = Color.Olive;
conner.Renderer = flatconrnerfooter;
fpSpread1.NamedStyles.Add(conner);
a1.CornerFooterDefaultStyle = conner;

```

VB

```

'header/footer column
fpSpread1.ActiveSheet.ColumnFooter.Visible = True
fpSpread1.ActiveSheet.ColumnFooter.RowCount = 3
fpSpread1.ActiveSheet.ColumnHeader.RowCount = 3
Dim flatcolumnheader As New FarPoint.Win.Spread.CellType.FlatColumnHeaderRenderer()
fpSpread1.ActiveSheet.ColumnHeader.DefaultStyle.Renderer = flatcolumnheader
Dim flatcolumnfooter As New FarPoint.Win.Spread.CellType.FlatColumnFooterRenderer()
fpSpread1.ActiveSheet.ColumnFooter.DefaultStyle.Renderer = flatcolumnfooter

'header row
fpSpread1.ActiveSheet.RowHeader.ColumnCount = 3
Dim flatrowheader As New FarPoint.Win.Spread.CellType.FlatRowHeaderRenderer()
fpSpread1.ActiveSheet.RowHeader.DefaultStyle.Renderer = flatrowheader

'sheet corner header render
Dim flatconrnerheader As New FarPoint.Win.Spread.CellType.FlatCornerHeaderRenderer()
fpSpread1.ActiveSheet.SheetCorner.DefaultStyle.Renderer = flatconrnerheader

'sheet corner footer render
Dim a1 As New
FarPoint.Win.Spread.SpreadSkin(FarPoint.Win.Spread.DefaultSpreadSkins.Default)
a1.Apply(fpSpread1)
fpSpread1.ActiveSheet.ColumnFooter.Visible = True
Dim flatconrnerfooter As New FarPoint.Win.Spread.CellType.FlatCornerFooterRenderer()
flatconrnerfooter.NormalTriangleColor = Color.Aquamarine
Dim conner = New FarPoint.Win.Spread.NamedStyle("conner", "HeaderDefault")
conner.BackColor = Color.Olive
conner.Renderer = flatconrnerfooter
fpSpread1.NamedStyles.Add(conner)
a1.CornerFooterDefaultStyle = conner

```

Handling Right-to-Left Layouts

The Spread component can support right-to-left layouts. Right-to-left features support applications where the language is written from right to left, such as Hebrew, Arabic, or Farsi, so the user interface would be displayed naturally with right-to-left orientation.

Most of the right-to-left functionality for the Spread component is inherited from the underlying .NET framework, but there are a few properties that can be set to instruct Spread to display the layout right-to-left. Currently these aspects of the layout are changed with the right to left support:

- cell editing
- column resizing on right instead of left
- direction attribute in HTML export
- filter and sort indicators
- order of columns
- popup and sticky notes and cell note indicators
- scroll bars
- shapes
- sheet corners
- sheet name tabs
- viewport columns

The members that are used with right-to-left support include:

Member

RightToLeft ('RightToLeft Property' in the on-line documentation) property

OnRightToLeftChanged ('OnRightToLeftChanged Method' in the on-line documentation) method

SpreadActions ('SpreadActions Class' in the on-line documentation) class

Description

Gets or sets whether the object should paint right to left

Raises the RightToLeft event in the Microsoft.NET Framework

These actions are added:

- **ExtendToNextColumnVisual** ('ExtendToNextColumnVisual Field' in the on-line documentation)
- **ExtendToPreviousColumnVisual** ('ExtendToPreviousColumnVisual Field' in the on-line documentation)
- **MoveToNextColumnVisual** ('MoveToNextColumnVisual Field' in the on-line documentation)
- **MoveToPreviousColumnVisual** ('MoveToPreviousColumnVisual Field' in the on-line documentation)
- **ScrollToNextColumnVisual** ('ScrollToNextColumnVisual Field' in the on-line documentation)
- **ScrollToPreviousColumnVisual** ('ScrollToPreviousColumnVisual Field' in the on-line documentation)

Customizing Painting of Parts of the Component

You can customize the painting of various parts of the component's display by looking for events and painting (rendering) those parts the way you want.

The members that are used with custom painting include:

Member	Description
OnPaintTabStrip ('OnPaintTabStrip Method' in the on-line documentation)	Raises the PaintTabStrip event
OnPaintTabStripButton ('OnPaintTabStripButton Method' in the on-line documentation)	Raises the PaintTabStripButton event
OnPaintTabStripTab ('OnPaintTabStripTab Method' in the on-line documentation)	Raises the PaintTabStripTab event
PaintTabStrip ('PaintTabStrip Event' in the on-line documentation) event	Occurs when the TabStrip needs painting
PaintTabStripButton ('PaintTabStripButton Event' in the on-line documentation) event	Occurs when a TabStrip button needs painting
PaintTabStripTab ('PaintTabStripTab Event' in the on-line documentation) event	Occurs when a TabStrip tab needs painting
PaintTabStripEventArgs ('PaintTabStripEventArgs Class' in the on-line documentation)	Contains data related to this event
PaintTabStripButtonEventArgs ('PaintTabStripButtonEventArgs Class' in the on-line documentation)	Contains data related to this event
PaintTabStripTabEventArgs ('PaintTabStripTabEventArgs Class' in the on-line documentation)	Contains data related to this event

Text Rendering with GDI

You can use GDI (instead of GDI+) for drawing text in Visual Studio 2005 by using the special FarPoint.Win.TextRenderer DLL and adding it to the references in a project. If you use the text renderer feature in your project for text drawing that is GDI-based rather than GDI+-based, in version 2.0 of the .NET Framework, then you must distribute the FarPoint.Win.TextRenderer.dll file. Only users who are using Visual Studio 2005 can benefit from this file, as that environment is required to target version 2.0 of the .NET Framework.

The FarPoint.Win.TextRenderer.dll is a thin wrapper for the new System.Windows.Forms.TextRenderer class in .NET 2.0. The Spread component tries to locate the FarPoint.Win.TextRenderer assembly if it can find the framework System.Windows.Forms.TextRenderer class in the System.Windows.Forms assembly (that is, if Spread is running under .NET 2.0). If it can, Spread loads the FarPoint TextRenderer and uses reflection to make dynamic calls into it to do the text drawing with System.Windows.Forms.TextRenderer instead of System.Drawing.Graphics.DrawString. Spread uses dynamic calls through reflection so that the FarPoint Spread assembly is not dependent on the FarPoint.Win.TextRenderer assembly; if the FarPoint assembly is not found, or if the System.Windows.Forms.TextRenderer class is not found in the System.Windows.Forms assembly, then Spread uses Graphics.DrawString to draw text as it has always done. For more information about how the new TextRenderer class in .NET 2.0 differs from the way the Graphics object draws text, in particular how TextRenderer uses GDI drawing APIs in Windows instead of GDI+ APIs in the .NET framework, refer to the .NET framework documentation about using TextRenderer class.



Spread can use GDI+ drawing while other text drawing in the application uses GDI with the new TextRenderer class; Spread can use the new TextRenderer class for GDI drawing while other text drawing in the application uses

GDI+. The differences are very small (a few pixels in the spacing and alignment) and not very noticeable.

This DLL must be installed to the directory where the application's executable file resides (the bin folder) on systems where GDI drawing in the Spread is preferred. The file may be installed to the GAC to ensure that all Spread controls in all applications that use version 2.0 of the .NET Framework will use the TextRenderer for text drawing. The file can be installed to the bin folder on an application-by-application basis, and the Framework will find it there.

For more information on GDI-based text drawing, refer to the web site, [spwin-celltypecustomize](#).

For more information in the Microsoft .NET Framework documentation, refer to [Microsoft .NET TextRenderer Class](#).

Applying Theme to Customize the Appearance

You can now easily modify the look & feel of your workbook using the provided themes. Spread themes include options to set the theme color, theme fonts, and theme effects. You can also customize these individual elements and manage overall appearance of the workbook. The tasks that relate to setting the appearance of the workbook include:

- **Applying a New Theme**
- **Customizing the Theme Fonts**
- **Customizing the Theme Colors**
- **Removing a Theme**

Spread themes are shared across all Spread components to provide a uniform look. In case a theme is not specified, default theme settings of are applied to the workbook.

Applying a New Theme

Use the following code to apply any theme to the workbook to modify the appearance.

C#

```
// To apply a new theme.
var fileName = @"C:\Program Files\Microsoft Office\root\Document Themes
16\Gallery.thmx";
fpSpread1.AsWorkbook().ApplyTheme(fileName);
fpSpread2.AsWorkbook().ApplyTheme(fpSpread1.AsWorkbook().Theme);
```

VB

```
' To apply a new theme.
Dim fileName As var = "C:\Program Files\Microsoft Office\root\Document Themes
16\Gallery.thmx"
fpSpread1.AsWorkbook().ApplyTheme(fileName)
fpSpread2.AsWorkbook().ApplyTheme(fpSpread1.AsWorkbook().Theme)
```

Spread provides a number of built-in themes as well. To apply a built-in theme to the workbook, use the **BuiltInThemes** enumeration option from **GrapeCity.Core** namespace as shown below.

```
fpSpread1.AsWorkbook().Theme = Theme.GetTheme(BuiltInThemes.Facet);
```

Using Spread Designer, you can also apply your desired built-in themes. For more information, refer to **Applying and Customizing Themes (on-line documentation)**.

Customizing the Theme Fonts

Theme fonts contain a heading font and a body text font. You can change both of these fonts to create your own set of theme fonts. Refer to the following code snippet to customize the font scheme of your workbook.

C#

```
// To customize theme fonts.
string fileName = @"C:\Program Files\Microsoft Office\root\Document Themes 16\Theme
Fonts\Century Gothic.xml";
fpSpread1.AsWorkbook().Theme.FontScheme.Load(fileName);
fpSpread2.AsWorkbook().Theme.FontScheme.Load(fileName);
```

VB

```
' To customize theme fonts.
Dim fileName As String = "C:\Program Files\Microsoft Office\root\Document Themes
16\Theme Fonts\Century Gothic.xml"
fpSpread1.AsWorkbook().Theme.FontScheme.Load(fileName)
fpSpread2.AsWorkbook().Theme.FontScheme.Load(fileName)
```

Customizing the Theme Colors

Refer to the following code snippet to customize the theme colors of your workbook.

C#

```
// To customize theme colors.
string fileName = @"C:\Program Files\Microsoft Office\root\Document Themes 16\Theme
Colors\Yellow Orange.xml";
fpSpread1.AsWorkbook().Theme.ColorScheme.Load(fileName);
fpSpread2.AsWorkbook().Theme.ColorScheme.Load(fileName);
```

VB

```
' To customize theme colors.
Dim fileName As String = "C:\Program Files\Microsoft Office\root\Document Themes
16\Theme Colors\Yellow Orange.xml"
fpSpread1.AsWorkbook().Theme.ColorScheme.Load(fileName)
fpSpread2.AsWorkbook().Theme.ColorScheme.Load(fileName)
```

Removing a Theme

Refer to the following code snippet to remove the currently applied theme from your workbook and change it back to the default theme.

C#

```
//To remove a theme
fpSpread1.AsWorkbook().ApplyTheme("");
fpSpread2.AsWorkbook().ApplyTheme("");
```

VB

```
' To remove a theme
fpSpread1.AsWorkbook().ApplyTheme("")
fpSpread2.AsWorkbook().ApplyTheme("")
```

Customizing Interaction in Cells

You can customize various aspects of user interaction in the Spread component. The tasks that relate to customizing the user interaction with the spreadsheet include:

- **Using Edit Mode and Focus**
- **Customizing User Selection and Deselection of Data**
- **Using Drag Operations to Fill Cells**
- **Using Validation in Cells**
- **Using Visible Indicators in the Cell**
- **Customizing Undo and Redo Actions**
- **Customizing Interaction Based on Events**
- **Adding a Context Menu to a Component**

You can customize what you allow the user to do. There are several features that are covered in various topics in this section, but they are summarized in one topic for easy reference in **Allowing User Functionality**.

You can also determine how the user interacts based on the cell type. Refer to **Cell Types**.

You can also determine how the user interacts with the keyboard. Refer to **Managing Keyboard Interaction**.

Using Edit Mode and Focus

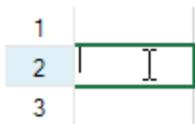
Important aspects of working with cells, and with user interaction at a cell level, include the use of edit mode and focus. The following topics describe how you can customize interaction including edit mode and focus:

- **Understanding Edit Mode in a Cell**
- **Locking a Cell**

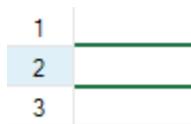
Understanding Edit Mode in a Cell

Typically, when the end user double-clicks in the cell, the editor control is made available and the user can type in the cell. This ability to edit in a cell is called edit mode. Several properties and methods can customize the use of edit mode.

When the cell is in edit mode, the active cell typically displays a flashing I-beam cursor, as shown in the figure below. When the cell is not in edit mode, the active cell typically displays a focus rectangle, also as shown.



Cell in edit mode



Cell selected but not in edit mode

A cell enters edit mode (edit mode is turned on) when

- the user starts typing in the cell
- the user double-clicks the cell
- the **EditMode** (**'EditMode Property' in the on-line documentation**) property is set to true

A cell leaves edit mode (edit mode is turned off) when

- the user presses the Enter key
- the user makes another cell the active cell
- the application loses the focus

- the **EditMode** ('**EditMode Property**' in the on-line documentation) property is set to false

When a cell enters edit mode, by default the cursor is positioned at the end of the existing text in the cell. You can change it to select the existing text in the cell by setting the **EditModeReplace** ('**EditModeReplace Property**' in the on-line documentation) property.

If you prefer, you can specify that a cell is always in edit mode when it becomes the active cell using the **EditModePermanent** ('**EditModePermanent Property**' in the on-line documentation) property.

When a cell enters edit mode, the **EditModeOn** ('**EditModeOn Event**' in the on-line documentation) event occurs; when a cell leaves edit mode, the **EditModeOff** ('**EditModeOff Event**' in the on-line documentation) event occurs.

You can set the position of the cursor in the edit control when it receives the focus by using the `SuperEditBase.EditModeCursorPosition` property.

You can start and stop edit mode by using the **StartCellEditing** ('**StartCellEditing Method**' in the on-line documentation) and **StopCellEditing** ('**StopCellEditing Method**' in the on-line documentation) methods.

You can keep the cell alignment when editing with the **AllowEditorVerticalAlign** ('**AllowEditorVerticalAlign Property**' in the on-line documentation) property.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. In the **Properties** window, select the **EditModePermanent** or **EditModeReplace** properties.

Using the Spread Designer

1. Select the **Settings** menu.
2. Select the **Edit** option (**Spread Settings** section).
3. Check the **Cells Always in EditMode** option for **EditModePermanent** or **Editing Replaces Existing Text** for **EditModeReplace**.
4. Use the **File** menu, then **Apply and Exit** to save the changes.

Locking a Cell

You can lock a cell or range of cells and make it unavailable for editing by the end user. You can make the appearance of locked cells different so that the locked cells are noticeable by the user.

You can lock cells using the **Locked** ('**Locked Property**' in the on-line documentation) property in the **Cell** ('**Cell Class**' in the on-line documentation), **Column** ('**Column Class**' in the on-line documentation), **Row** ('**Row Class**' in the on-line documentation), or **AlternatingRow** ('**AlternatingRow Class**' in the on-line documentation) objects. You can also set the **Locked** property for the **StyleInfo** ('**StyleInfo Class**' in the on-line documentation) object and apply that style to the cells you want locked. You also need to set the **Protect** ('**Protect Property**' in the on-line documentation) property of the **SheetView** ('**SheetView Class**' in the on-line documentation) object to lock the cells. The **Locked** property marks the cells to be locked, and the **Protect** property sets whether to lock those cells. For cells marked as locked to be locked from user input, the **Protect** ('**Protect Property**' in the on-line documentation) property of the sheet must be set to True, which is its default value. If it is set to False, the user can still interact with the cells.

Another way to lock cells is to make them text cells (using the **TextCellType** ('**TextCellType Class**' in the on-line documentation)) and set the **ReadOnly** property. This makes the cells non-editable.

You can also specify a different color (for background or for text) or font in locked cells using the **LockBackColor** ('**LockBackColor Property**' in the on-line documentation), **LockForeColor** ('**LockForeColor Property**' in the on-line documentation), and **LockFont** ('**LockFont Property**' in the on-line documentation) properties of the **SheetView** ('**SheetView Class**' in the on-line documentation), **Appearance** ('**Appearance Class**' in

the on-line documentation), Cell ('Cell Class' in the on-line documentation), Column ('Column Class' in the on-line documentation), Row ('Row Class' in the on-line documentation), NamedStyle ('NamedStyle Class' in the on-line documentation), or StyleInfo ('StyleInfo Class' in the on-line documentation) objects.

Locking a cell does not lock any shapes (floating objects) over that cell. A protected sheet only means that all the cells in that sheet marked as locked are locked; it does not apply to shapes over that sheet. For locking shapes, refer to **Customizing Drawing**.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Properties** window on the right side of the **SheetView Collection Editor**, select the **Cells** property for the sheet.
5. Click the button to display the **Cell, Column, and Row Editor**.
6. In the editor, select the cells to mark as locked.
7. Select the **Locked** property and set the value to **True**.
8. If you want to apply this change, click **Apply**.
9. Click **OK** to close the **Cell, Column, and Row** editor.
10. In the **Properties** window on the right side of the **SheetView Collection Editor**, in the **Behavior** group, select the **Protect** property and set it to **True** if you want the cells to be locked from user input.
11. Click **OK** to close the **SheetView Collection Editor**.

Using Code

1. Set the **Locked ('Locked Property' in the on-line documentation)** property for the **Cell ('Cell Class' in the on-line documentation)** object (or **Row ('Row Class' in the on-line documentation)** or **Column ('Column Class' in the on-line documentation)** object for all the cells in that row or column) to mark some cells as locked.
2. Set the **Protect ('Protect Property' in the on-line documentation)** property for the sheet (**SheetView ('SheetView Class' in the on-line documentation)** object) to **True** if you want the cells that are marked as locked in that sheet to be locked from user input.

Example

Making sure that the **Protect ('Protect Property' in the on-line documentation)** property is true for the sheet, you can lock specified columns of cells and then unlock some of the cells in one row, as shown in the following example.

C#

```
fpSpread1.ActiveSheet.Protect = true;
fpSpread1.ActiveSheet.LockBackColor = Color.LightCyan;
fpSpread1.ActiveSheet.LockForeColor = Color.Green;
FarPoint.Win.Spread.Column columnobj;
columnobj = fpSpread1.ActiveSheet.Columns[0, 3];
columnobj.Locked = true;
FarPoint.Win.Spread.Cell cellobj;
cellobj = fpSpread1.ActiveSheet.Cells[1,1,1,2];
cellobj.Locked = false;
fpSpread1.ActiveSheet.Cells[1,0,1,4].Text = "First Five";
```

VB

```
fpSpread1.ActiveSheet.Protect = True
fpSpread1.ActiveSheet.LockBackColor = Color.LightCyan
fpSpread1.ActiveSheet.LockForeColor = Color.Green
Dim columnobj As FarPoint.Win.Spread.Column
columnobj = fpSpread1.ActiveSheet.Columns(0, 3)
columnobj.Locked = True
Dim cellobj As FarPoint.Win.Spread.Cell
cellobj = fpSpread1.ActiveSheet.Cells(1,1,1,2)
cellobj.Locked = False
fpSpread1.ActiveSheet.Cells(1,0,1,4).Text = "First Five"
```

Using a Shortcut

1. Set the **Locked ('Locked Property' in the on-line documentation)** property for the **Cells** shortcut (or **Rows** or **Columns** shortcut for all the cells in that row or column) to mark some cells as locked.
2. Set the **Protect ('Protect Property' in the on-line documentation)** property for the **Sheets** shortcut to **True** if you want the cells that are marked as locked in that sheet to be locked from user input.

Example

Making sure that the **Protect** property is **True** for the sheet, you can lock specific columns of cells and then unlock some of the cells in one row, as shown in the following example.

C#

```
fpSpread1.ActiveSheet.Protect = true;
fpSpread1.ActiveSheet.LockBackColor = Color.LightCyan;
fpSpread1.ActiveSheet.LockForeColor = Color.Green;
fpSpread1.ActiveSheet.Columns[0, 3].Locked = true;
fpSpread1.ActiveSheet.Cells[1,1,1,2].Locked = false;
```

VB

```
fpSpread1.ActiveSheet.Protect = True
fpSpread1.ActiveSheet.LockBackColor = Color.LightCyan
fpSpread1.ActiveSheet.LockForeColor = Color.Green
fpSpread1.ActiveSheet.Columns(0, 3).Locked = True
fpSpread1.ActiveSheet.Cells(1,1,1,2).Locked = False
```

Using the Spread Designer

1. In the work area, select the cell or cells for which you want to lock the cells either by dragging over a range of cells or selecting row or column headers (for entire rows or columns).
(Another way of doing this is to select the **Cells** property, click on the button to call up the **Cell, Column, and Row** editor, and select the cells in that editor.)
2. In the properties list (in the **Misc** group), select the **Locked** property and choose **True**.
3. Click the sheet name tab for the sheet that contains the cells. From the properties list (in the **Behavior** category), select the **Protect** property button to display the **SheetView Collection Editor**.
4. In the **Properties** window in the **SheetView Collection Editor**, in the **Behavior** group, select the **Protect** property and set it to **True** if you want the cells to be locked from user input.
5. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Customizing User Selection and Deselection of Data

You can customize what the user can select and how the selection appears. To customize aspects of selections, you can perform the following tasks:

- **Specifying What the User Can Select**
- **Customizing the Selection Appearance**
- **Working with Selections**
- **Hiding the Selection When Focus is Lost**

Specifying What the User Can Select

By default, sheets allow users to select a cell, a column, a row, a range of cells, or the entire sheet. You can customize how selection occurs and what can be selected by working with the operation mode of the sheet and with the selection policy and selection unit of the sheet.

The following table summarizes the options available for specifying what users can select:

What user can select	When setting this for the sheet
Cells	<code>FpSpread.SelectionBlockOptions ('SelectionBlockOptions Property' in the on-line documentation).Cells</code>
Rows	<code>FpSpread.SelectionBlockOptions ('SelectionBlockOptions Property' in the on-line documentation).Rows</code>
Columns	<code>FpSpread.SelectionBlockOptions ('SelectionBlockOptions Property' in the on-line documentation).Columns</code>
Sheet	<code>FpSpread.SelectionBlockOptions ('SelectionBlockOptions Property' in the on-line documentation).Sheet</code>
Combination	<code>FpSpread.SelectionBlockOptions ('SelectionBlockOptions Property' in the on-line documentation).number</code> where number is some addition of the numbers for the individual settings (such as $6 = 2 + 4$, Rows and Columns)
Cells, ranges of cells, or multiple ranges of cells	<code>OperationMode ('OperationMode Property' in the on-line documentation).Normal</code> with <code>SelectionPolicy ('SelectionPolicy Property' in the on-line documentation)</code> property
Only rows, no editing	<code>OperationMode ('OperationMode Property' in the on-line documentation).SingleSelect</code>
Only rows, editing	<code>OperationMode ('OperationMode Property' in the on-line documentation).RowMode</code>
Multiple contiguous rows, no editing	<code>OperationMode ('OperationMode Property' in the on-line documentation).MultiSelect</code>
Multiple noncontiguous rows, no editing	<code>OperationMode ('OperationMode Property' in the on-line documentation).ExtendedSelect</code>

Note that the `FpSpread.SelectionBlockOptions ('SelectionBlockOptions Property' in the on-line documentation)` are settings at the Spread component level, while the `OperationMode ('OperationMode Property' in the on-line documentation)` settings are at the sheet level.

The settings of the `OperationMode ('OperationMode Property' in the on-line documentation)` and the `SelectionBlockOptions ('SelectionBlockOptions Property' in the on-line documentation)` properties affect user interaction with the sheet, that is, what the user can select, but not necessarily what the application can select. If you want to customize what the user and the application both can select, set the `SelectionUnit ('SelectionUnit Property' in the on-line documentation)` property.

You can also restrict which cells can be edited by using the **RestrictRows** ('RestrictRows Property' in the on-line documentation) and **RestrictColumns** ('RestrictColumns Property' in the on-line documentation) methods for the sheet. This restricts users from entering data beyond the next row or column. For more information, refer to the **SelectionPolicy** ('SelectionPolicy Property' in the on-line documentation) property and the **SelectionUnit** ('SelectionUnit Property' in the on-line documentation) property of the **SheetView** ('SheetView Class' in the on-line documentation) class, and the **SelectionBlockOptions** ('SelectionBlockOptions Property' in the on-line documentation) property of the **FpSpread** ('FpSpread Class' in the on-line documentation) class. For more details, see the **OperationMode** ('OperationMode Enumeration' in the on-line documentation) and the **SelectionBlockOptions** ('SelectionBlockOptions Enumeration' in the on-line documentation) enumerations.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet for which to set the operation mode.
5. Select the **OperationMode** property, then select one of the values from the drop-down list of values.
6. If you set the **OperationMode** property to Normal, and you want to allow users to select only a cell or to select multiple ranges of cells, set the **SelectionPolicy** property to Single or to MultiRange.
7. To set the overall selection interaction for the sheet, including how users and the application can select items, set the **SelectionUnit** property to specify the unit of selection allowed.
8. Click **OK** to close the editor.
9. If you want to customize what users can select in all sheets in the Spread component, set the **SelectionBlockOptions** property to specify whether they can select cells, columns, rows, the sheet or a combination of these.

Using a Shortcut

1. To set the overall user interaction mode of the sheet, set the **Sheets OperationMode** ('OperationMode Property' in the on-line documentation) property.
2. If you set the **OperationMode** ('OperationMode Property' in the on-line documentation) property to Normal,
 - a. If you want to customize what users can select in all sheets in the Spread component, set the **FpSpread SelectionBlockOptions** ('SelectionBlockOptions Property' in the on-line documentation) property to specify whether they can select cells, columns, rows, the sheet or a combination of these.
 - b. If you want to allow users to select only a cell or to select multiple ranges of cells, set the **Sheets SelectionPolicy** ('SelectionPolicy Property' in the on-line documentation) property to Single or to MultiRange.
3. To set the overall selection interaction for the sheet, including how users and the application can select items, set the **Sheets SelectionUnit** ('SelectionUnit Property' in the on-line documentation) property to specify the unit of selection allowed.

Example

This example code sets the sheet to allow users to select only cells or ranges of cells, including multiple ranges of cells. They cannot select columns, rows, or the entire sheet in this example.

C#

```
// Set option so users can select only cells.
fpSpread1.SelectionBlockOptions = FarPoint.Win.Spread.SelectionBlockOptions.Cells; //
Set operation mode and let users select multiple blocks of
cells.fpSpread1.Sheets[0].OperationMode = FarPoint.Win.Spread.OperationMode.Normal;
```

```
fpSpread1.Sheets[0].SelectionPolicy =
FarPoint.Win.Spread.Model.SelectionPolicy.MultiRange;
```

VB

```
' Set option so users can select only cells.
fpSpread1.SelectionBlockOptions = FarPoint.Win.Spread.SelectionBlockOptions.Cells
' Set operation mode and let users select multiple blocks of cells.
fpSpread1.Sheets(0).OperationMode = FarPoint.Win.Spread.OperationMode.Normal
fpSpread1.Sheets(0).SelectionPolicy =
FarPoint.Win.Spread.Model.SelectionPolicy.MultiRange
```

Using Code

- To set the overall user interaction mode of the sheet, set the **OperationMode ('OperationMode Property' in the on-line documentation)** property for a **SheetView** object.
- If you set the **OperationMode ('OperationMode Property' in the on-line documentation)** property to Normal,
 - If you want to customize what users can select in all sheets in the Spread component, set the **FpSpread SelectionBlockOptions ('SelectionBlockOptions Property' in the on-line documentation)** property to specify whether they can select cells, columns, rows, the sheet or a combination of these.
 - If you want to allow users to select only a cell or to select multiple ranges of cells, set the **SheetView** object **SelectionPolicy ('SelectionPolicy Property' in the on-line documentation)** property to Single or to MultiRange.
- To set the overall selection interaction for the sheet, including how users and the application can select items, set the **SheetView** object **SelectionUnit ('SelectionUnit Property' in the on-line documentation)** property to specify the unit of selection allowed.
- Assign the **SheetView** object you have created to one of the sheets in the Spread component.

Example

This example code sets the sheet to allow users to select only cells or ranges of cells, including multiple ranges of cells. They cannot select columns, rows, or the entire sheet in this example.

C#

Type your example code here. It will be automatically colorized when you switch to Preview or build the help system.

VB

```
' Set option so users can select only cells.
fpSpread1.SelectionBlockOptions = FarPoint.Win.Spread.SelectionBlockOptions.Cells
' Set operation mode and let users select multiple blocks of cells.
Dim newsheet As New FarPoint.Win.Spread.SheetView()
newsheet.OperationMode = FarPoint.Win.Spread.OperationMode.Normal
newsheet.SelectionPolicy = FarPoint.Win.Spread.Model.SelectionPolicy.MultiRange
' Assign the SheetView object to a sheet.
fpSpread1.Sheets(0) = newsheet
```

Using the Spread Designer

- Select the sheet tab for the sheet for which you want to set the selection operation mode.
- From the **Settings** menu, select general (**Sheet Settings** section). In the **Sheet Settings** dialog, on the **General** tab, select one of the choices from the **Operation Mode** area.

3. Click **OK** to close the **Sheet Settings** dialog.
4. If you set the **OperationMode** property to Normal,
 - a. If you want to customize what users can select in all sheets in the Spread component, set the **SelectionBlockOptions** property (for the selected Spread) in the **Properties** list to specify whether they can select cells, columns, rows, the sheet or a combination of these.
 - b. If you want to allow users to select only a cell or to select multiple ranges of cells, set the **SelectionPolicy** property (for the selected Sheet) in the **Properties** list to Single or to MultiRange.
5. To set the overall selection interaction for the sheet, including how users and the application can select items, set the **SelectionUnit** property (for the selected Sheet) in the **Properties** list to specify the unit of selection allowed.
6. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Customizing the Selection Appearance

Selections have a default appearance provided by the Spread component and the selection renderer. You can change that appearance, including the background and foreground colors and font. You can also specify a row selector icon with the **ShowRowSelector** ('**ShowRowSelector Property**' in the on-line documentation) property. You can specify a row edit icon with the **ShowEditingRowSelector** ('**ShowEditingRowSelector Property**' in the on-line documentation) property as illustrated in the following image.

	A	B	C
1			
2			
 3		test	
4			

You can specify whether to display the selection header, border, or active cell with the **PaintSelectionHeader** ('**PaintSelectionHeader Property**' in the on-line documentation), **PaintSelectionBorder** ('**PaintSelectionBorder Property**' in the on-line documentation), or **PaintActiveCellInSelection** ('**PaintActiveCellInSelection Property**' in the on-line documentation) property.

By default, the Spread component uses the appearance set by the selection renderer. When something is selected, the renderer changes the color of the background of the selection. Instead of using this rendering, you can specify specific colors to use for the background and text colors of selections. Alternatively, you can use both the renderer's appearance and colors you set. Finally, you can specify that no appearance is used to highlight selections.

The following figure shows cells selected using the default renderer style, then cells selected using set colors, and finally, cells selected using both the renderer style and set colors.

	A	B
1		
2		
3		4,000
4		

SelectionStyle=Renderer

	A	B
1		
2		
3		4,000
4		

SelectionStyle=Colors

	A	B
1		
2		
3		4,000
4		

SelectionStyle=Both

If no color is set for the selection, then the color is Color.FromArgb(100, 193, 224, 255).

Painting of selected cells is determined by the various properties in the **SheetView** ('**SheetView Class**' in the on-line documentation) class:

SheetView Property

Description

SelectionBackColor (' SelectionBackColor Property ' in the on-line documentation)	Determines the background color of selections
SelectionForeColor (' SelectionForeColor Property ' in the on-line documentation)	Determines the text color of selections
SelectionMode (' SelectionMode Property ' in the on-line documentation)	Determines how the selections are styled using either the colors or a custom renderer or both
SelectionFont (' SelectionFont Property ' in the on-line documentation)	Determines the font of the selected text

When the **SelectionMode** ('**SelectionMode Property**' in the on-line documentation) is **SelectionColors**, the cell is painted using the **SelectionBackColor** ('**SelectionBackColor Property**' in the on-line documentation) and **SelectionForeColor** ('**SelectionForeColor Property**' in the on-line documentation) settings in place of the cell's **ForeColor** ('**ForeColor Property**' in the on-line documentation) and **BackColor** ('**BackColor Property**' in the on-line documentation) property settings. When the **SelectionMode** ('**SelectionMode Property**' in the on-line documentation) is **SelectionRenderer**, the cell is painted using the cell's **ForeColor** ('**ForeColor Property**' in the on-line documentation) and **BackColor** ('**BackColor Property**' in the on-line documentation) property settings. Then a semi-transparent layer is painted over the cell. The semi-transparent layer is accomplished using the following.

C#

```
Brush selectionBrush = new SolidBrush(Color.FromArgb(100, 193, 224, 255));
g.FillRectangle(selectionBrush, x, y, width, height);
```

For more information, refer to the **SelectionMode** ('**SelectionMode Enumeration**' in the on-line documentation) enumeration and the **ISelectionRenderer** ('**ISelectionRenderer Interface**' in the on-line documentation) interface.

For more information on selection settings, refer to the **SelectionRenderer** ('**SelectionRenderer Property**' in the on-line documentation) property and the **RetainSelectionBlock** ('**RetainSelectionBlock Property**' in the on-line documentation) property in the **FpSpread** ('**FpSpread Class**' in the on-line documentation) class.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet for which to set the selection appearance.
5. Select the **SelectionMode** property, then select one of the values from the drop-down list of values.
6. If you set the **SelectionMode** to **SelectionColors** or **Both**, set the **SelectionBackColor** and **SelectionForeColor** properties to specify the background and text colors for the selection highlighting.
7. Click **OK** to close the editor.

Using a Shortcut

1. To specify how to draw the selection highlighting, set the **Sheets SelectionMode** ('**SelectionMode Property**' in the on-line documentation) property.
2. If you set the **SelectionMode** ('**SelectionMode Property**' in the on-line documentation) to **SelectionColors** or **Both**, set the **Sheets SelectionBackColor** ('**SelectionBackColor Property**' in the on-line documentation) and **SelectionForeColor** ('**SelectionForeColor Property**' in the on-line documentation) to specify the colors to use for the background and for the text.

Example

This example code sets the selection highlighting to use the renderer settings and colors.

C#

```
// Use the selection renderer and colors.
fpSpread1.Sheets[0].SelectionMode = FarPoint.Win.Spread.SelectionStyles.Both;
// Set the background and text colors.
fpSpread1.Sheets[0].SelectionBackColor = System.Drawing.Color.Pink;
fpSpread1.Sheets[0].SelectionForeColor = System.Drawing.Color.Navy;
```

VB

```
' Use the selection renderer and colors.
fpSpread1.Sheets(0).SelectionMode = FarPoint.Win.Spread.SelectionStyles.Both
' Set the background and text colors.
fpSpread1.Sheets(0).SelectionBackColor = System.Drawing.Color.Pink
fpSpread1.Sheets(0).SelectionForeColor = System.Drawing.Color.Navy
```

Using Code

1. To specify how to draw the selection highlighting, set the **SelectionMode** (**'SelectionMode Property' in the on-line documentation**) property for a **SheetView** (**'SheetView Class' in the on-line documentation**) object.
2. If you set the **SelectionMode** (**'SelectionMode Property' in the on-line documentation**) to SelectionColors or Both, set the **SheetView** object **SelectionBackColor** (**'SelectionBackColor Property' in the on-line documentation**) and **SelectionForeColor** (**'SelectionForeColor Property' in the on-line documentation**) to specify the colors to use for the background and for the text.
3. Assign the **SheetView** (**'SheetView Class' in the on-line documentation**) object you have created to one of the sheets in the component.

Example

This example code sets the selection highlighting to use the renderer settings and colors.

C#

```
FarPoint.Win.Spread.SheetView newsheet=new FarPoint.Win.Spread.SheetView();
// Use the selection renderer and colors.
newsheet.SelectionMode = FarPoint.Win.Spread.SelectionStyles.Both;
newsheet.SelectionBackColor = System.Drawing.Color.AliceBlue;
// Set the background and text colors.
newsheet.SelectionForeColor = System.Drawing.Color.Navy;
// Assign the SheetView to a sheet in the component.
fpSpread1.Sheets[0] = newsheet;
```

VB

```
Dim newsheet As New FarPoint.Win.Spread.SheetView()
' Use the selection renderer and colors.
newsheet.SelectionMode = FarPoint.Win.Spread.SelectionStyles.Both
' Set the background and text colors.
newsheet.SelectionBackColor = System.Drawing.Color.AliceBlue
newsheet.SelectionForeColor = System.Drawing.Color.Navy
' Assign the SheetView to a sheet in the component.
fpSpread1.Sheets(0) = newsheet
```

Working with Selections

When a user selects a range of cells, that range of cells can have a separate background color and foreground color to distinguish it from the other cells in the spreadsheet. The range is called a selection. There are many aspects of selections that you can manage programmatically. In code you can add and remove selections and you can find out what is selected. This topic summarizes some of the tasks you can perform with selections in code.

- To add a selection (a range of cells that are displayed as selected), use the **Sheets AddSelection ('AddSelection Method' in the on-line documentation)** method and specify the starting row and column, and the number of rows and columns in the selection.
- To get all the ranges of cells that are presently selected, use the **Sheets GetSelections ('GetSelections Method' in the on-line documentation)** method. To return a specific selection, use the **Sheets GetSelection ('GetSelection Method' in the on-line documentation)** method.
- To remove all of the selections, use the **Sheets ClearSelection ('ClearSelection Method' in the on-line documentation)** method. To remove a specific selection, use the **Sheets RemoveSelection ('RemoveSelection Method' in the on-line documentation)** method and specify the row and column, and the number of rows and columns to remove from the selection.
- To clear all selections when a new active cell is set programmatically, using the Boolean **clearSelection** parameter in the **SetActiveCell ('SetActiveCell Method' in the on-line documentation)** method.
- To keep a selection highlighted, use the **RetainSelectionBlock ('RetainSelectionBlock Property' in the on-line documentation)** property of the **FpSpread ('FpSpread Class' in the on-line documentation)** class.
- You can move a selected cell in the view using the **MoveActiveOnFocus ('MoveActiveOnFocus Property' in the on-line documentation)** property of the **FpSpread ('FpSpread Class' in the on-line documentation)** class.
- To work with events regarding selections, refer to the **SelectionChangedEventArgs ('SelectionChangedEventArgs Class' in the on-line documentation)** class.

To select all the cells in a sheet use the **RowCount ('RowCount Property' in the on-line documentation)** and **ColumnCount ('ColumnCount Property' in the on-line documentation)** properties for that sheet, as in this line of code:

```
fpSpread1.ActiveSheet.Models.Selection.SetSelection(0, 0, fpSpread1.ActiveSheet.RowCount, fpSpread1.ActiveSheet.ColumnCount)
```

The **DefaultSheetSelectionModel** class (and **IDisjointSelection** interface) **GetSelections** method returns -1 for either the RowCount or the ColumnCount if all the cells in that row or column are selected, as when the end user clicks on a header to make a selection.

For information on the underlying model for selections, refer to **Understanding the Selection Model**.

Using a Shortcut

To add a selection, use the **SheetView's AddSelection ('AddSelection Method' in the on-line documentation)** method from the **Sheets** shortcut, specifying the necessary parameters.

Example

This example code selects two ranges of cells.

C#

```
// Set the sheet to allow multiple range selections.
fpSpread1.Sheets[0].SelectionPolicy = FarPoint.Win.Spread.Model.SelectionPolicy.MultiRange;
// Select cells C3 through D4.
fpSpread1.Sheets[0].AddSelection(2, 2, 2, 2);
// Select cells F6 through H8.
fpSpread1.Sheets[0].AddSelection(5, 5, 3, 3);

// Select cells F6 through H8.
fpSpread1.Sheets[0].AddSelection(5, 5, 3, 3);
```

VB

```
' Set the sheet to allow multiple range selections.
fpSpread1.Sheets(0).SelectionPolicy = FarPoint.Win.Spread.Model.SelectionPolicy.MultiRange
' Select cells C3 through D4.
fpSpread1.Sheets(0).AddSelection(2, 2, 2, 2)
' Select cells F6 through H8.
fpSpread1.Sheets(0).AddSelection(5, 5, 3, 3)
```

Using Code

To add a selection, use the **AddSelection ('AddSelection Method' in the on-line documentation)** method from the **Sheets** shortcut, specifying the necessary parameters. Then assign the **SheetView ('SheetView Class' in the on-line documentation)** object to a sheet in the component.

Example

This example code selects two ranges of cells.

C#

```
FarPoint.Win.Spread.SheetView newsheet=new FarPoint.Win.Spread.SheetView();
// Add two selections.
newsheet.SelectionPolicy = FarPoint.Win.Spread.Model.SelectionPolicy.MultiRange;
newsheet.AddSelection(2, 2, 2, 2);
newsheet.AddSelection(5, 5, 3, 3);
```

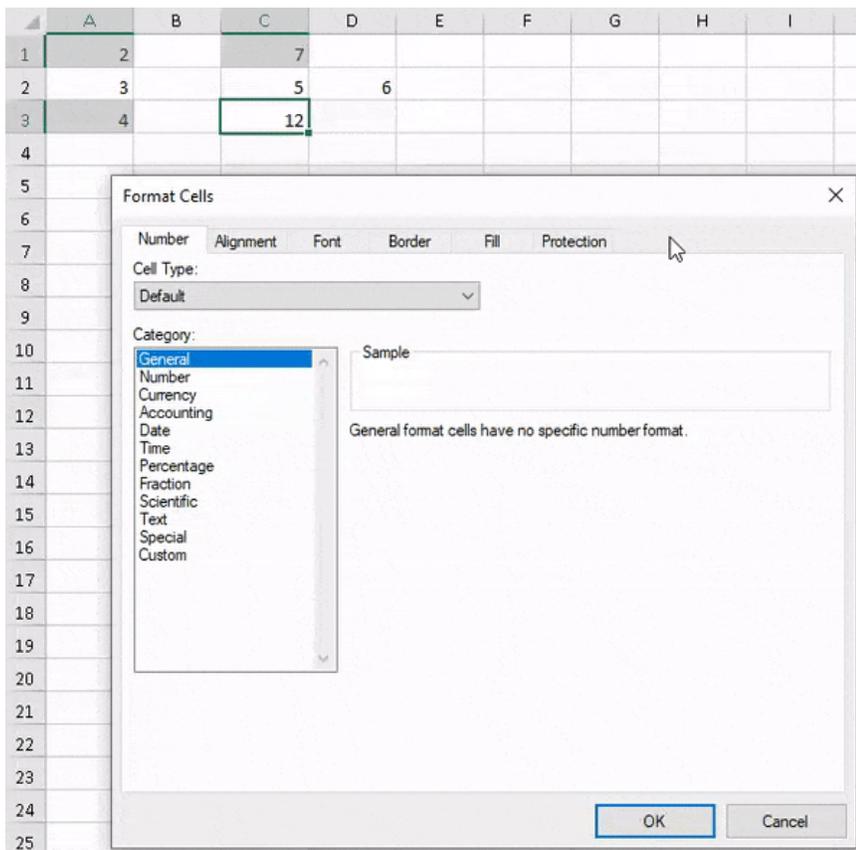
```
// Assign the SheetView to a sheet in the component.
fpSpread1.Sheets[0] = newsheet;
```

VB

```
Dim newsheet As New FarPoint.Win.Spread.SheetView()
' Add two selections.
newsheet.SelectionPolicy = FarPoint.Win.Spread.Model.SelectionPolicy.MultiRange
newsheet.AddSelection(2, 2, 2, 2)
newsheet.AddSelection(5, 5, 3, 3)
' Assign the SheetView to a sheet in the component.
fpSpread1.Sheets(0) = newsheet
```

Working with multi-range selections

Users can format multiple cells simultaneously using the Format Cells dialog at runtime after selecting a range of cells. The **BuiltInDialogs** class provides **FormatCells** method that allows you to customize the style settings.



Using Code

This example code allows you to select multi-range cells, and call the Format Cells dialog to apply the style effects.

C#

```
FarPoint.Win.Spread.Dialogs.BuiltInDialogs.FormatCells(TestActiveSheet.Selection).ShowDialog();
```

Hiding the Selection When Focus is Lost

By default, the Spread component displays the focus rectangle and highlights selections, regardless of whether the component has the focus. If you prefer, you can set the component to hide the focus rectangle and selection highlighting when the component does not have the focus. In this case, the component still displays the focus and highlight selections when the component regains the focus. (This does not apply when **EditModePermanent ('EditModePermanent Property' in the on-line documentation)** is set to True, in which case the focus rectangle and selection highlighting are not displayed.) For more information refer to the **RetainSelectionBlock ('RetainSelectionBlock Property' in**

the on-line documentation) property of the **FpSpread ('FpSpread Class' in the on-line documentation)** class.

You can specify that edit mode is always on in the spreadsheet. When edit mode is always on, the spreadsheet acts like a form, where the user can enter data in any data field and the focus rectangle is not displayed. Only a blinking I-bar is displayed in a cell to show the cursor. For more information refer to the **EditModePermanent ('EditModePermanent Property' in the on-line documentation)** property of the **FpSpread ('FpSpread Class' in the on-line documentation)** class.

For more information about the focus indicator, refer to **Customizing the Focus Indicator for a Cell**.

Working with Deselections

Spread for WinForms allows users to deselect the selected cells just like in Excel. In order to deselect a selection, users simply need to press and hold the ctrl key or cmd key on the keyboard and use the left, top, right or down arrow keys to deselect the cells that they want.

The deselection feature is useful especially when users mistakenly select extra cells within a spreadsheet. With large worksheets carrying huge amounts of data, it becomes cumbersome to deselect the unnecessary cells. Using the ctrl or cmd key, users can efficiently manage the entire process of deselection without having to start over; thus saving both time and efforts.

In order to demonstrate the working of deselection behavior, some example images are shared below:

1. The following image depicts how a user can work with four selection areas when the deselection operation is executed inside the original selection.

	A	B	C	D	E	F
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						

2. The following image depicts how a user can work with less than four selection areas when the deselection operation crosses the original selection.

	A	B	C	D	E	F
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						

3. The following image depicts how a user can work with "No selections" when the deselection operation is executed

within the original selection. In this scenario, after the deselection is performed, an active cell will be added at the beginning of the deselected selection.

	A	B	C	D	E	F
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						

After deselection, the new selection will be created as per the following rules:

- The new selections are specified in the row-major order.
- From left to right and from up to down, the cells are extended as much as possible.
- If the deselection operation contains the entire original selection, the deselection result is "none" selection.



Note: The deselect feature will be enabled only when the Default Sheet Selection Model is used, the **SelectionUnit ('SelectionUnit Enumeration' in the on-line documentation)** enumeration is set to "Cell" and the **SelectionPolicy ('SelectionPolicy Enumeration' in the on-line documentation)** enumeration is set to "MultiRange".

Example

This example code shows how to work with deselections in a spreadsheet.

C#

```
fpSpread1_Sheet1.SelectionUnit = FarPoint.Win.Spread.Model.SelectionUnit.Cell;
fpSpread1_Sheet1.SelectionPolicy =
FarPoint.Win.Spread.Model.SelectionPolicy.MultiRange;

// Adding Selection
fpSpread1.Sheets[0].AddSelection(0, 0, 10, 5);
var sels = fpSpread1.ActiveSheet.GetSelections();
Console.WriteLine(sels.Length);

// Removing selection from selected range of cells
fpSpread1.Sheets[0].RemoveSelection(2, 2, 2, 1);
fpSpread1.Sheets[0].RemoveSelection(5, 1, 2, 3);
fpSpread1.Sheets[0].RemoveSelection(8, 3, 1, 1);
```

VB

```
fpSpread1_Sheet1.SelectionUnit = FarPoint.Win.Spread.Model.SelectionUnit.Cell
fpSpread1_Sheet1.SelectionPolicy = FarPoint.Win.Spread.Model.SelectionPolicy.MultiRange

'Adding Selection

fpSpread1.Sheets(0).AddSelection(0, 0, 10, 5)
```

```
Dim sels = fpSpread1.ActiveSheet.GetSelections()  
Console.WriteLine(sels.Length)  
  
'Removing selection from selected range of cells  
  
fpSpread1.Sheets(0).RemoveSelection(2, 2, 2, 1)  
fpSpread1.Sheets(0).RemoveSelection(5, 1, 2, 3)  
fpSpread1.Sheets(0).RemoveSelection(8, 3, 1, 1)
```

Using Drag Operations to Fill Cells

You can fill cells with any of several operations that involve dragging over some cells. The drag operations, which can copy and move the content as well as the formatting, include:

- **Filling Cells with Drag and Drop**
- **Filling Cells with Drag and Fill**
- **Filling Cells with Drag and Move**

Filling Cells with Drag and Drop

You can allow the end user to drag-and-drop data from one range of cells to another. You can specify whether the user can select a cell or range of cells and drag and drop them to a new location in the same spreadsheet or another spreadsheet in the Spread component. When the mouse button is released and the range of cells is dropped, the **DragDropBlock ('DragDropBlock Event' in the on-line documentation)** event occurs.

For more information, refer to the **AllowDragDrop ('AllowDragDrop Property' in the on-line documentation)** property in the **FpSpread ('FpSpread Class' in the on-line documentation)** class. For more information on event arguments, refer to **DragDropBlock ('DragDropBlock Event' in the on-line documentation)** event, **DragDropBlockEventArgs ('DragDropBlockEventArgs Class' in the on-line documentation)**, and **DragDropBlockCompletedEventArgs ('DragDropBlockCompletedEventArgs Class' in the on-line documentation)**.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select (in the **Behavior** category) the **AllowDragDrop** property.
3. Select **True** from the drop-down list to allow drag-and-drop feature, or select **False** to prohibit it.

Using a Shortcut

Allow the drag-and-drop feature by setting the **AllowDragDrop ('AllowDragDrop Property' in the on-line documentation)** property for the **FpSpread ('FpSpread Class' in the on-line documentation)** component.

Example

This example code sets the component to allow the drag-drop feature.

C#

```
fpSpread1.AllowDragDrop = true;
```

VB

```
fpSpread1.AllowDragDrop = True
```

Using Code

Allow the drag-and-drop feature by setting the **AllowDragDrop** ('**AllowDragDrop Property**' in the **on-line documentation**) property of the **FpSpread** ('**FpSpread Class**' in the **on-line documentation**) component.

Example

This example code sets the child sheet to allow the drag-drop feature.

C#

```
FarPoint.Win.Spread.SpreadView sv = fpSpread1.GetRootWorkbook();  
sv.AllowDragDrop = true;
```

VB

```
Dim sv As FarPoint.Win.Spread.SpreadView = fpSpread1.GetRootWorkbook  
sv.AllowDragDrop = True
```

Using the Spread Designer

1. From the **Settings** menu, select **General** (**Spread Settings** section).
2. In the **General** tab, select the settings for the drag and drop properties as needed.
3. Click **OK**.
4. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

or

1. Select the Spread component (or select **Spread** from the pull-down menu).
2. In the property list for the component (in the **Behavior** category), select the **AllowDragDrop** property.
3. Click the drop-down arrow to display the choices and select **True**.
4. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Filling Cells with Drag and Fill

You can also allow the end user to drag-and-fill data from one cell or a range of cells to another cell or range of cells. With a cell or range of cells selected, you can fill other cells either in a row (or rows if more than one column is selected) or a column (or columns if more than one row is selected).

In order to fill cells with drag and fill, the **AllowDragFill** ('**AllowDragFill Property**' in the **on-line documentation**) property in the **FpSpread** ('**FpSpread Class**' in the **on-line documentation**) class must be set to true. In order to show the drag fill button and the drag fill context menu, the **EnableDragFillMenu** ('**EnableDragFillMenu Property**' in the **on-line documentation**) property must be set to true.

You can also specify the type of fill you want by using the **RangeDragFillMode** ('**RangeDragFillMode Property**' in the **on-line documentation**) property that allows you to choose from the following options in the drag fill drop down context menu icon that appears on your screen:

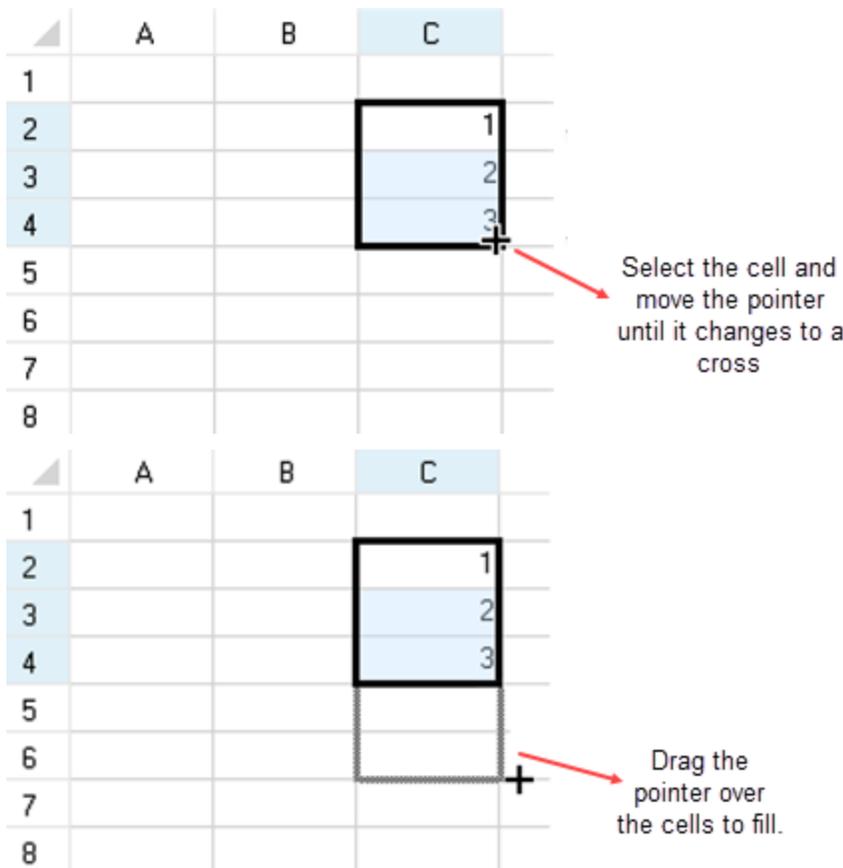
1. Copy Cells
2. Fill Series
3. Fill Formatting Only
4. Fill Without Formatting

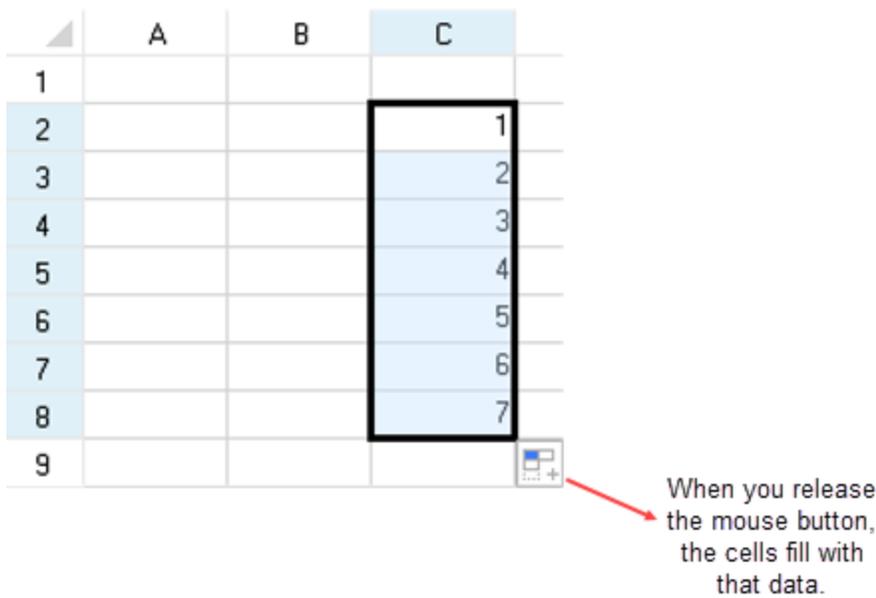
The Copy Cells option copies the data to the selected range. The Fill Series option increments or decrements the values

based on the series. Drag down or to the right to increment the values. Drag up or to the left to decrement the values. Spread uses the data model to determine whether the data can be incremented when using the series fill. The displayed values have a higher priority with the built-in cell types when using the series fill. The Fill Formatting Only option and the Fill Without Formatting option allows you to fill cells with or without the same formatting as in the originally selected cell.

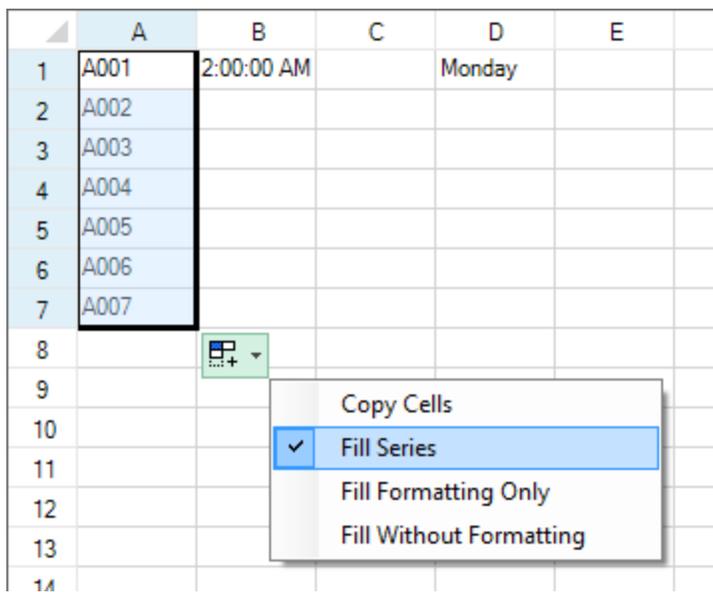
The AutoFill options that are available in the Drag Fill context menu depends on the type of selected data and the **DragFillDataOnly** ('**DragFillDataOnly Property**' in the on-line documentation) property.

The example shown below depicts the drag and fill operation that uses the copy option to fill several cells in a column.





The following example uses the fill series option.



You can customize the direction of the fill using the **FillDirection** ('**FillDirection Enumeration**' in the **on-line documentation**) enumeration.

For more information on event arguments, refer to **DragFillBlock** ('**DragFillBlock Event**' in the **on-line documentation**) event, **DragFillBlockEventArgs** ('**DragFillBlockEventArgs Class**' in the **on-line documentation**), and **DragFillBlockCompletedEventArgs** ('**DragFillBlockCompletedEventArgs Class**' in the **on-line documentation**).

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select (in the **Behavior** category) the **AllowDragFill** property.

3. Select **True** from the drop-down list to allow drag-and-fill feature, or select **False** to prohibit it.
4. Select the **EnableDragFillMenu** property.
5. Select **True** from the drop-down list to allow the drag fill button and drag fill context menu, or select **False** to prohibit it.

Using a Shortcut

Allow the drag-and-fill feature by setting the **AllowDragFill ('AllowDragFill Property' in the on-line documentation)** property in the **FpSpread ('FpSpread Class' in the on-line documentation)** component.

Allow the drag fill button and drag fill context menu by setting the **EnableDragFillMenu ('EnableDragFillMenu Property' in the on-line documentation)** property in the **FpSpread ('FpSpread Class' in the on-line documentation)** component.

Example

This example code sets the component to allow the drag-fill feature, drag fill button and the drag fill context menu.

C#

```
fpSpread1.AllowDragFill = true;  
fpSpread1.EnableDragFillMenu = true;
```

VB

```
fpSpread1.AllowDragFill = True  
fpSpread1.EnableDragFillMenu = True
```

Using Code

Allow the drag fill feature by setting the **AllowDragFill ('AllowDragFill Property' in the on-line documentation)** property in the **FpSpread ('FpSpread Class' in the on-line documentation)** component.

Allow the drag fill button and drag fill context menu by setting the **EnableDragFillMenu ('EnableDragFillMenu Property' in the on-line documentation)** property in the **FpSpread ('FpSpread Class' in the on-line documentation)** component.

Example

This example code sets the child sheet to allow the drag fill feature, drag fill button and the drag fill context menu.

C#

```
// To enable Drag Fill operation  
fpSpread1.ActiveSheet.Reset();  
fpSpread1.DragFillDataOnly = false;  
fpSpread1.AllowDragFill = true;  
fpSpread1.EnableDragFillMenu = true;  
this.fpSpread1.ActiveSheet.Cells[0, 0].Value = "A001";  
this.fpSpread1.ActiveSheet.Cells[0, 1].Value = DateTime.Today;  
this.fpSpread1.ActiveSheet.Cells[0, 3].Text = "Monday";
```

VB

```
' To enable Drag Fill operation  
fpSpread1.ActiveSheet.Reset()  
fpSpread1.DragFillDataOnly = False
```

```
fpSpread1.AllowDragFill = True
fpSpread1.EnableDragFillMenu = True
Me.fpSpread1.ActiveSheet.Cells(0, 0).Value = "A001"
Me.fpSpread1.ActiveSheet.Cells(0, 1).Value = DateTime.Today
Me.fpSpread1.ActiveSheet.Cells(0, 3).Text = "Monday"
```

Using the Spread Designer

1. Select the Spread component (or select **Spread** from the pull-down menu).
2. In the property list for the component (in the **Behavior** category), select the **AllowDragFill** property.
3. Click the drop-down arrow to display the choices and select **True**.
4. In the property list for the component, select the **EnableDragFillMenu** property.
5. Click the drop-down arrow to display the choices and select **True**.
6. From the **File** menu, choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.



Note:

If the `DragFillDataOnly` property is set to false and the selected data can be filled as series, all the options will appear in the context menu. If the `RangeDragFillMode` property is set to Series, then the context menu will show only the Fill Series option and if it is set to Copy, then the context menu will show only the Copy Cells option. If the selected data cannot be filled as series, the context menu will show only three options: Copy Cells, Fill Formatting Only, Fill Without Formatting.

If the `DragFillDataOnly` property is set to true and the selected data can be filled as series, the context menu will show only two options: Copy Cells and Fill Series. If the `RangeDragFillMode` property is set to Series, then the context menu will show only the Fill Series option and if it is set to Copy, then the context menu will show only the Copy Cells option. If the selected data cannot be filled as series, the context menu will show only one option: Copy Cells.

Filling Cells with Drag and Move

You can drag and move entire rows or columns of cells within a given sheet.

For more information, refer to the **Moving Rows or Columns**.

For more information on event arguments, refer to **DragMoveEventArgs** ('**DragMoveEventArgs Class**' in the **on-line documentation**) and **DragMoveCompletedEventArgs** ('**DragMoveCompletedEventArgs Class**' in the **on-line documentation**).

Using Validation

You can validate the contents of the cell in a number of ways. The following topics describe about how to validate the input from a user.

- **Using Validation in Cells**
 - **Using a Cell Comparison Validator**
 - **Using a Character Format Validator**
 - **Using a String Comparison Validator**
 - **Using a Value Comparison Validator**
 - **Using an Encoding Validator**
 - **Using the Exclude List Validator**
 - **Using the Include List Validator**
 - **Using a Pair Validator**

- **Using the Range Validator**
- **Using a Regular Expression Validator**
- **Using a Required Field Validator**
- **Using a Required Type Validator**
- **Using a Surrogate Character Validator**
- **Using a Text Length Validator**
- **Validating User Input**
- **Customizing the User Error Messages**
- **Displaying Error Icons in Cells or Rows**

You can validate the data in a cell using the cell validation. For more information, refer to **Using Validation in Cells**.

You can also combine cell types and events to validate the data in a cell. For more information, refer to **Validating User Input**.

Using Validation in Cells

You can use built-in validators and actions to validate data in cells. The validators allow you to set validation conditions such as minimum and maximum values. The actions allow you to notify the user when a validation fails (cell color change, sound, and so on).

The following validators are available:

- CharFormatValidator
- CompareCellValidator
- CompareStringValidator
- CompareValueValidator
- EncodingValidator
- ExcludeListValidator
- IncludeListValidator
- PairCharValidator
- RangeValidator
- RegularExpressionValidator
- RequiredFieldValidator
- RequiredTypeValidator
- SurrogateCharValidator
- TextLengthValidator

The following validation actions are available:

- Underline notifications (**LineNotify ('LineNotify Class' in the on-line documentation)**)
- Cell style notifications (**CellStyleNotify ('CellStyleNotify Class' in the on-line documentation)**)
- Icon notifications (**IconNotify ('IconNotify Class' in the on-line documentation)**)
- Sound notifications (**SoundNotify ('SoundNotify Class' in the on-line documentation)**)
- Tooltip notifications (**TipNotify ('TipNotify Class' in the on-line documentation)**)
- Focus change notifications (**FocusProcess ('FocusProcess Class' in the on-line documentation)**)
- Previous value notifications (**ValueProcess ('ValueProcess Class' in the on-line documentation)**)
- Message box notifications (**MessageBoxNotify ('MessageBoxNotify Class' in the on-line documentation)**)
- Three-state icon notifications (**ThreeStateIconNotify ('ThreeStateIconNotify Class' in the on-line documentation)**)

The following topics provide information about using validators in cells:

- Using a Cell Comparison Validator
- Using a Character Format Validator
- Using a String Comparison Validator
- Using a Value Comparison Validator
- Using an Encoding Validator
- Using the Exclude List Validator
- Using the Include List Validator
- Using a Pair Validator
- Using the Range Validator
- Using a Regular Expression Validator
- Using a Required Field Validator
- Using a Required Type Validator
- Using a Surrogate Character Validator
- Using a Text Length Validator

Using a Cell Comparison Validator

You can create a validator that compares the cell value to another cell's value. You can compare DateTime, TimeSpan, or Decimal type (Numeric) values.

A validation error occurs if the value is not valid. You can also create an action, such as adding an underline to the cell, that lets the user know the value is invalid.

Use the **CompareCellValidator** class to create the validation conditions. Specify a notification type such as **LineNotify** ('LineNotify Class' in the on-line documentation). Then use the **AddValidators** ('AddValidators Method' in the on-line documentation) method to add the validator to a cell range.

The following image displays a red underline for an invalid value.

	A	B	C
1	10		
2		3	
3			
4			

Using Code

The following example adds a red underline if you type a value less than or equal to 10 in cell 1,1.

CS

```
//Type a value equal to or less than 10 to see the red line
    FarPoint.Win.Spread.LineNotify linen = new
FarPoint.Win.Spread.LineNotify();
    linen.LineColor = Color.Red;
    linen.DoActionReason = FarPoint.Win.Spread.ValidateReasons.EndEdit;
    FarPoint.Win.Spread.CompareCellValidator compare = new
FarPoint.Win.Spread.CompareCellValidator();
    compare.ComparedOperator =
FarPoint.Win.Spread.ValidateComparisonOperator.GreaterThan;
    compare.Row = 0;
    compare.Column = 0;
    compare.Actions.Add(linen);
```

```

fpSpread1.Sheets[0].AddValidators(new
FarPoint.Win.Spread.Model.CellRange(1, 1, 1, 1), compare);
fpSpread1.Sheets[0].Cells[0, 0].Value = 10;

```

VB

```

'Type a value equal to Or less than 10 to see the red line
Dim linen As New FarPoint.Win.Spread.LineNotify()
linen.LineColor = Color.Red
linen.DoActionReason = FarPoint.Win.Spread.ValidateReasons.EndEdit
Dim compare As New FarPoint.Win.Spread.CompareCellValidator()
compare.ComparedOperator =
FarPoint.Win.Spread.ValidateComparisonOperator.GreaterThan
compare.Row = 0
compare.Column = 0
compare.Actions.Add(linen)
fpSpread1.Sheets(0).AddValidators(New
FarPoint.Win.Spread.Model.CellRange(1, 1, 1, 1), compare)
fpSpread1.Sheets(0).Cells(0, 0).Value = 10

```

Using a Character Format Validator

You can create a validator that checks for specific characters.

A validation error occurs if the value is not valid. You can also create an action, such as adding a backcolor to the cell, that lets the user know the value is invalid.

Use the **CharFormatValidator** class to create the validator. Specify a notification type such as **CellStyleNotify** (**'CellStyleNotify Class' in the on-line documentation**). Then use the **AddValidators** (**'AddValidators Method' in the on-line documentation**) method to add the validator to a cell range.

The following image displays an invalid backcolor for the cell.

	A	B	C
1			
2		Test	
3			
4			

Using Code

The following example displays an invalid backcolor if you type characters other than the format string.

CS

```

FarPoint.Win.Spread.CellStyleNotify cnotify = new
FarPoint.Win.Spread.CellStyleNotify();
cnotify.InvalidCellStyle.BackColor = Color.Aqua;
FarPoint.Win.Spread.CharFormatValidator cFormatValidator1 = new
FarPoint.Win.Spread.CharFormatValidator();
cFormatValidator1.Format = "A a ";
cFormatValidator1.Actions.Add(cnotify);
fpSpread1.Sheets[0].AddValidators(new FarPoint.Win.Spread.Model.CellRange(1, 1, 1, 1),
cFormatValidator1);

```

VB

```
Dim cnotify As New FarPoint.Win.Spread.CellStyleNotify()
cnotify.InvalidCellStyle.BackColor = Color.Aqua
Dim charFormatValidator1 As New FarPoint.Win.Spread.CharFormatValidator()
charFormatValidator1.Format = "A a "
charFormatValidator1.Actions.Add(cnotify)
fpSpread1.Sheets(0).AddValidators(New FarPoint.Win.Spread.Model.CellRange(1, 1, 1, 1),
charFormatValidator1)
```

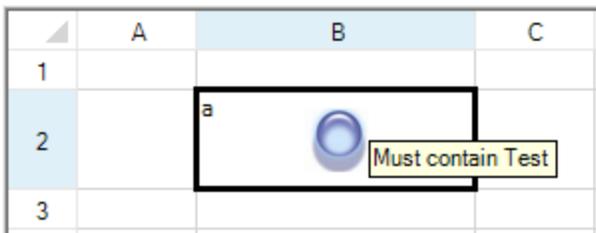
Using a String Comparison Validator

You can create a validator that compares string values.

A validation error occurs if the value does not match the specified conditions. You can also create an action, such as displaying an icon in the cell, that lets the user know the value is invalid.

Use the **CompareStringValidator** class to create the string validator. Specify a notification type such as **IconNotify** (**IconNotify Class' in the on-line documentation**). Then use the **AddValidators** (**AddValidators Method' in the on-line documentation**) method to add the validator to a cell range.

The following image displays an error icon with a text tip.

**Using Code**

The following example displays an icon if you type a string that does not contain "Test".

CS

```
//Type a value in cell 1,1
FarPoint.Win.Spread.IconNotify iconn = new FarPoint.Win.Spread.IconNotify();
iconn.Icon = new System.Drawing.Icon("C:\\Program Files (x86)\\Mescius\\spread.ico");
iconn.IconAlignment = ContentAlignment.MiddleCenter;
iconn.IconTip = "Must contain Test";
FarPoint.Win.Spread.CompareStringValidator svalid = new
FarPoint.Win.Spread.CompareStringValidator();
svalid.ComparedOperator = FarPoint.Win.Spread.CompareStringValidatorOperator.Contains;
svalid.ComparedString = "Test";
svalid.Actions.Add(iconn);
fpSpread1.Sheets[0].AddValidators(new FarPoint.Win.Spread.Model.CellRange(1, 1, 1, 1),
svalid);
fpSpread1.Sheets[0].Columns[1].Width = 140;
fpSpread1.Sheets[0].Rows[1].Height = 50;
```

VB

```
'Type a value in cell 1,1
Dim iconn As New FarPoint.Win.Spread.IconNotify()
iconn.Icon = New System.Drawing.Icon("C:\\Program Files (x86)\\Mescius\\spread.ico")
```

```

iconn.IconAlignment = ContentAlignment.MiddleCenter
iconn.IconTip = "Must contain Test"
Dim svalid As New FarPoint.Win.Spread.CompareStringValidator()
svalid.ComparedOperator = FarPoint.Win.Spread.CompareStringValidatorOperator.Contains
svalid.ComparedString = "Test"
svalid.Actions.Add(iconn)
fpSpread1.Sheets(0).AddValidators(New FarPoint.Win.Spread.Model.CellRange(1, 1, 1, 1),
svalid)
fpSpread1.Sheets(0).Columns(1).Width = 140
fpSpread1.Sheets(0).Rows(1).Height = 50

```

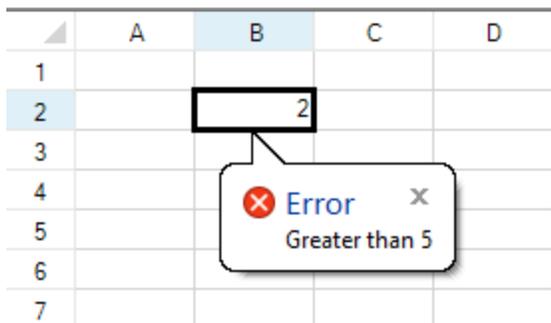
Using a Value Comparison Validator

You can create a validator that compares a cell value to other values. You can compare DateTime, TimeSpan, or Decimal type (Numeric) values.

A validation error occurs if the value is not valid. You can also create an action, such as adding a text tip to the cell, that lets the user know the value is invalid.

Use the **CompareValueValidator** class to create the validator. Specify a notification type such as **TipNotify** (**'TipNotify Class' in the on-line documentation**). Then use the **AddValidators** (**'AddValidators Method' in the on-line documentation**) method to add the validator to a cell range.

The following image displays the tip notification for an invalid value.



Using Code

The following example displays an icon if you type a value less than 5.

CS

```

//Type a value in cell 1,1
FarPoint.Win.Spread.TipNotify tnote = new FarPoint.Win.Spread.TipNotify();
tnote.ToolTipText = "Greater than 5";
tnote.ToolTipTitle = "Error";
tnote.ToolTipIcon = ToolTipIcon.Error;
FarPoint.Win.Spread.CompareValueValidator compare = new
FarPoint.Win.Spread.CompareValueValidator();
compare.ComparedOperator = FarPoint.Win.Spread.ValidateComparisonOperator.GreaterThan;
compare.ComparedValue = 5;
compare.Actions.Add(tnote);
fpSpread1.Sheets[0].AddValidators(new FarPoint.Win.Spread.Model.CellRange(1, 1, 1, 1),
compare);

```

VB

```
'Type a value in cell 1,1
Dim tnote As New FarPoint.Win.Spread.TipNotify()
tnote.ToolTipText = "Greater than 5"
tnote.ToolTipTitle = "Error"
tnote.ToolTipIcon = ToolTipIcon.Error
Dim compare As New FarPoint.Win.Spread.CompareValueValidator()
compare.ComparedOperator = FarPoint.Win.Spread.ValidateComparisonOperator.GreaterThan
compare.ComparedValue = 5
compare.Actions.Add(tnote)
fpSpread1.Sheets(0).AddValidators(New FarPoint.Win.Spread.Model.CellRange(1, 1, 1, 1),
compare)
```

Using an Encoding Validator

You can create a validator to determine whether a cell's value matches the specified encoding type.

A validation error occurs if the value is not valid. You can also create an action, such as adding an icon to the cell, that lets the user know the value is invalid.

Use the **EncodingValidator** class to create the validator. Specify a notification type such as **ThreeStateIconNotify** (**'ThreeStateIconNotify Class' in the on-line documentation**). Then use the **AddValidators** (**'AddValidators Method' in the on-line documentation**) method to add the validator to a cell range.

The **ThreeStateIconNotify** (**'ThreeStateIconNotify Class' in the on-line documentation**) class allows you to display an icon for a valid value, as shown in the following image.

	A	B	C
1			
2		valid	✔
3			
4			

Using Code

The following example displays a valid icon if you type an ASCII value or an invalid icon if you type a non-ASCII value.

CS

```
//Press Alt and type 0176 to enter a non-ASCII character in cell 1,1
FarPoint.Win.Spread.ThreeStateIconNotify three = new
FarPoint.Win.Spread.ThreeStateIconNotify();
FarPoint.Win.Spread.EncodingValidator compare = new
FarPoint.Win.Spread.EncodingValidator();
compare.Encoding = System.Text.Encoding.ASCII;
compare.Actions.Add(three);
fpSpread1.Sheets[0].AddValidators(new FarPoint.Win.Spread.Model.CellRange(1, 1, 1, 1),
compare);
```

VB

```
'Press Alt and type 0176 to enter a non-ASCII character in cell 1,1
Dim three As New FarPoint.Win.Spread.ThreeStateIconNotify()
Dim compare As New FarPoint.Win.Spread.EncodingValidator()
compare.Encoding = System.Text.Encoding.ASCII
compare.Actions.Add(three)
fpSpread1.Sheets(0).AddValidators(New FarPoint.Win.Spread.Model.CellRange(1, 1, 1, 1),
```

compare)

Using the Exclude List Validator

You can create a validator that has a list of invalid values. An error occurs if the user types a value from the candidate list.

You can also create an action, such as adding a bgcolor to the cell, that lets the user know the value is invalid.

Use the **ExcludeListValidator** class to create the validator. Specify a notification type such as **CellStyleNotify** ('CellStyleNotify Class' in the on-line documentation). Then use the **AddValidators** ('AddValidators Method' in the on-line documentation) method to add the validator to a cell range.

The following image displays a cell with an invalid value.

	A	B	C
1			
2		Deer	
3			
4			

Using Code

The following example displays an invalid bgcolor if you type an item from the list.

CS

```
//Type one of the candidate items in cell 1,1 to see the error notification
FarPoint.Win.Spread.CellStyleNotify cnotify = new
FarPoint.Win.Spread.CellStyleNotify();
cnotify.InvalidCellStyle.BackColor = Color.Bisque;
FarPoint.Win.Spread.ExcludeListValidator excludelist = new
FarPoint.Win.Spread.ExcludeListValidator();
excludelist.Candidates = new string[] { "Bird", "Deer", "Squirrel", "Lizard" };
excludelist.Actions.Add(cnotify);
fpSpread1.Sheets[0].AddValidators(new FarPoint.Win.Spread.Model.CellRange(1, 1, 1, 1),
excludelist);
```

VB

```
'Type one of the candidate items in cell 1,1 to see the error notification
Dim cnotify As New FarPoint.Win.Spread.CellStyleNotify()
cnotify.InvalidCellStyle.BackColor = Color.Bisque
Dim excludelist As New FarPoint.Win.Spread.ExcludeListValidator()
excludelist.Candidates = New String() {"Bird", "Deer", "Squirrel", "Lizard"}
excludelist.Actions.Add(cnotify)
fpSpread1.Sheets(0).AddValidators(New FarPoint.Win.Spread.Model.CellRange(1, 1, 1, 1),
excludelist)
```

Using the Include List Validator

You can create a validator that has a list of valid values.

A validation error occurs if the value is not valid. You can also create an action, such as adding a bgcolor to the cell, that lets the user know the value is invalid.

Use the **IncludeListValidator** class to create the validator. Specify a notification type such as **CellStyleNotify** ('CellStyleNotify Class' in the on-line documentation). Then use the **AddValidators** ('AddValidators Method' in the on-line documentation) method to add the validator to a cell range.

The following image displays an invalid bgcolor.

	A	B	C
1			
2		Item	
3			
4			

Using Code

The following example displays an invalid bgcolor if you type an item from the list.

CS

```
//Type an item in cell 1,1 that is not in the list to see the error notification
FarPoint.Win.Spread.CellStyleNotify cnotify = new
FarPoint.Win.Spread.CellStyleNotify();
cnotify.InvalidCellStyle.BackColor = Color.Yellow;
FarPoint.Win.Spread.IncludeListValidator ilst = new
FarPoint.Win.Spread.IncludeListValidator();
ilst.Candidates = new string[] { "Bird", "Deer", "Squirrel", "Lizard" };
ilst.Actions.Add(cnotify);
fpSpread1.Sheets[0].AddValidators(new FarPoint.Win.Spread.Model.CellRange(1, 1, 1, 1),
ilst);
```

VB

```
'Type an item in cell 1,1 that is not in the list to see the error notification
Dim cnotify As New FarPoint.Win.Spread.CellStyleNotify()
cnotify.InvalidCellStyle.BackColor = Color.Yellow
Dim IList As New FarPoint.Win.Spread.IncludeListValidator()
IList.Candidates = New String() {"Bird", "Deer", "Squirrel", "Lizard"}
IList.Actions.Add(cnotify)
fpSpread1.Sheets(0).AddValidators(New FarPoint.Win.Spread.Model.CellRange(1, 1, 1, 1),
IList)
```

Using a Pair Validator

You can create a validator that requires matching characters. This can be useful if you enter text that requires matching brackets.

A validation error occurs if the value is not valid. You can also create an action, such as adding a bgcolor to the cell, that lets the user know the value is invalid.

Use the **PairCharValidator** class to create the validator. Specify a notification type such as **CellStyleNotify** ('CellStyleNotify Class' in the on-line documentation). Then use the **AddValidators** ('AddValidators Method' in the on-line documentation) method to add the validator to a cell range.

The following image displays a cell with an invalid bgcolor.

	A	B	C
1			
2		(
3			

Using Code

The following example requires matching parentheses to be valid.

CS

```
//The error bgcolor is displayed if the text contains a single, non-matching
parenthesis
FarPoint.Win.Spread.CellStyleNotify cnotify = new
FarPoint.Win.Spread.CellStyleNotify();
cnotify.InvalidCellStyle.BackColor = Color.Yellow;
FarPoint.Win.Spread.PairChar pair = new FarPoint.Win.Spread.PairChar();
pair.Left = '(';
pair.Right = ')';
FarPoint.Win.Spread.PairCharValidator pairvalid = new
FarPoint.Win.Spread.PairCharValidator();
pairvalid.PairChars.Add(pair);
pairvalid.Actions.Add(cnotify);
fpSpread1.Sheets[0].AddValidators(new FarPoint.Win.Spread.Model.CellRange(1, 1, 1, 1),
pairvalid);
```

VB

```
'The error bgcolor is displayed if the text contains a single, non-matching
parenthesis
Dim cnotify As New FarPoint.Win.Spread.CellStyleNotify()
cnotify.InvalidCellStyle.BackColor = Color.Yellow
Dim pair As New FarPoint.Win.Spread.PairChar()
pair.Left = "("
pair.Right = ")"
Dim pairvalid As New FarPoint.Win.Spread.PairCharValidator()
pairvalid.PairChars.Add(pair)
pairvalid.Actions.Add(cnotify)
fpSpread1.Sheets(0).AddValidators(New FarPoint.Win.Spread.Model.CellRange(1, 1, 1, 1),
pairvalid)
```

Using the Range Validator

You can use the range validator to check to see if a value is within a specified range. A validation error occurs if the value is not within the specified range. You can also create an action, such as changing the cell bgcolor, that lets the user know the value is invalid.

Use the **RangeValidator** class to create the validation conditions. Specify a notification type such as **CellStyleNotify** (**'CellStyleNotify Class' in the on-line documentation**). Then use the **AddValidators** (**'AddValidators Method' in the on-line documentation**) method to add the validator to a cell range.

The following image uses the **CellStyleNotify** (**'CellStyleNotify Class' in the on-line documentation**) class to change the cell bgcolor when an invalid value is typed.

	A	B	C
1			
2		12	
3			

Using Code

The following example sets a minimum and maximum value for the cell. A different bgcolor is shown if you type an invalid value in the cell.

CS

```
FarPoint.Win.Spread.CellStyleNotify stylenotify = new
FarPoint.Win.Spread.CellStyleNotify();
stylenotify.InvalidCellStyle.BackColor = Color.Aquamarine;
FarPoint.Win.Spread.RangeValidator rvalid = new FarPoint.Win.Spread.RangeValidator();
rvalid.MaxValue = 10;
rvalid.MinValue = 0;
rvalid.Actions.Add(stylenotify);
fpSpread1.Sheets[0].AddValidators(new FarPoint.Win.Spread.Model.CellRange(1, 1, 1, 1),
rvalid);
```

VB

```
Dim stylenotify As New FarPoint.Win.Spread.CellStyleNotify()
stylenotify.InvalidCellStyle.BackColor = Color.Aquamarine
Dim rvalid As New FarPoint.Win.Spread.RangeValidator()
rvalid.MaxValue = 10
rvalid.MinValue = 0
rvalid.Actions.Add(stylenotify)
fpSpread1.Sheets(0).AddValidators(New FarPoint.Win.Spread.Model.CellRange(1, 1, 1, 1),
rvalid)
```

Using a Regular Expression Validator

You can create a regular expression validator that checks to see if a cell's value matches the specified regular expression.

A validation error occurs if the value is not valid. You can also create an action, such as adding a bgcolor to the cell, that lets the user know the value is invalid.

Use the **RegularExpressionValidator** class to create the validator. Specify a notification type such as **CellStyleNotify** ('**CellStyleNotify Class**' in the on-line documentation). Then use the **AddValidators** ('**AddValidators Method**' in the on-line documentation) method to add the validator to a cell range.

	A	B	C
1			
2		test	
3			
4			

Using Code

The following example displays the invalid color if the value does not match the expression.

CS

```
//The error bgcolor is displayed if the value is invalid
FarPoint.Win.Spread.CellStyleNotify cnotify = new
FarPoint.Win.Spread.CellStyleNotify();
cnotify.InvalidCellStyle.BackColor = Color.Lime;
FarPoint.Win.Spread.RegularExpressionValidator rvalidator = new
FarPoint.Win.Spread.RegularExpressionValidator();
rvalidator.Expression = "AA";
rvalidator.RegexOptions = System.Text.RegularExpressions.RegexOptions.IgnoreCase;
rvalidator.NullIsValid = true;
rvalidator.Actions.Add(cnotify);
fpSpread1.Sheets[0].AddValidators(new FarPoint.Win.Spread.Model.CellRange(1, 1, 1, 1),
rvalidator);
```

VB

```
'The error bgcolor is displayed if the value is invalid
Dim cnotify As New FarPoint.Win.Spread.CellStyleNotify()
cnotify.InvalidCellStyle.BackColor = Color.Lime
Dim rvalidator As New FarPoint.Win.Spread.RegularExpressionValidator()
rvalidator.Expression = "AA"
rvalidator.RegexOptions = System.Text.RegularExpressions.RegexOptions.IgnoreCase
rvalidator.NullIsValid = True
rvalidator.Actions.Add(cnotify)
fpSpread1.Sheets(0).AddValidators(New FarPoint.Win.Spread.Model.CellRange(1, 1, 1, 1),
rvalidator)
```

Using a Required Field Validator

You can use the required field validator to specify that the cell requires a value.

A validation error occurs if the value is not valid. You can also create an action, such as adding a bgcolor to the cell, that lets the user know the value is invalid.

Use the **RequiredFieldValidator** class to create the validator. Specify a notification type such as **CellStyleNotify** ('**CellStyleNotify Class**' in the on-line documentation). Then use the **AddValidators** ('**AddValidators Method**' in the on-line documentation) method to add the validator to a cell range.

The following image displays a cell with an invalid, empty value.

	A	B	C
1			
2			
3			

Using Code

The following example displays the invalid color if the value is deleted from the cell.

CS

```
//Cell 1,1 requires a value
FarPoint.Win.Spread.CellStyleNotify cnotify = new
FarPoint.Win.Spread.CellStyleNotify();
```

```

cnotify.InvalidCellStyle.BackColor = Color.Fuchsia;
FarPoint.Win.Spread.RequiredFieldValidator requiredv = new
FarPoint.Win.Spread.RequiredFieldValidator();
requiredv.Actions.Add(cnotify);
fpSpread1.Sheets[0].AddValidators(new FarPoint.Win.Spread.Model.CellRange(1, 1, 1, 1),
requiredv);
fpSpread1.Sheets[0].Cells[1, 1].Text = "Value";

```

VB

```

'Cell 1,1 requires a value
Dim cnotify As New FarPoint.Win.Spread.CellStyleNotify()
cnotify.InvalidCellStyle.BackColor = Color.Fuchsia
Dim requiredv As New FarPoint.Win.Spread.RequiredFieldValidator()
requiredv.Actions.Add(cnotify)
fpSpread1.Sheets(0).AddValidators(New FarPoint.Win.Spread.Model.CellRange(1, 1, 1, 1),
requiredv)
fpSpread1.Sheets(0).Cells(1, 1).Text = "Value"

```

Using a Required Type Validator

You can create a validator that requires a valid type. The required type validator supports DateTime, TimeSpan, and Decimal (Numeric).

A validation error occurs if the value is not valid. You can also create an action, such as adding a bgcolor to the cell, that lets the user know the value is invalid.

Use the **RequiredTypeValidator** class to create the validator. Specify a notification type such as **CellStyleNotify** ('**CellStyleNotify Class**' in the on-line documentation). Then use the **AddValidators** ('**AddValidators Method**' in the on-line documentation) method to add the validator to a cell range.

The following image displays an invalid bgcolor in the cell.

	A	B	C
1			
2		Test	
3			

Using Code

The following example displays the invalid color if the value is not decimal or null.

CS

```

//Type a text string in cell 1,1 to see the error notification
FarPoint.Win.Spread.CellStyleNotify cnotify = new
FarPoint.Win.Spread.CellStyleNotify();
cnotify.InvalidCellStyle.BackColor = Color.Aqua;
FarPoint.Win.Spread.RequiredTypeValidator requiredt = new
FarPoint.Win.Spread.RequiredTypeValidator();
requiredt.RequiredType = typeof(decimal);
requiredt.NullIsValid = true;
requiredt.Actions.Add(cnotify);
fpSpread1.Sheets[0].AddValidators(new FarPoint.Win.Spread.Model.CellRange(1, 1, 1, 1),
requiredt);

```

```
fpSpread1.Sheets[0].Cells[1, 1].Value = 5;
```

VB

```
'Type a text string in cell 1,1 to see the error notification
Dim cnotify As New FarPoint.Win.Spread.CellStyleNotify()
cnotify.InvalidCellStyle.BackColor = Color.Aqua
Dim requiredt As New FarPoint.Win.Spread.RequiredTypeValidator()
requiredt.RequiredType = GetType(Decimal)
requiredt.NullIsValid = True
requiredt.Actions.Add(cnotify)
fpSpread1.Sheets(0).AddValidators(New FarPoint.Win.Spread.Model.CellRange(1, 1, 1, 1),
requiredt)
fpSpread1.Sheets(0).Cells(1, 1).Value = 5
```

Using a Surrogate Character Validator

You can create a validator that checks if surrogate characters have been entered in the cell

A validation error occurs if the value is not valid. You can also create an action, such as adding a backcolor to the cell, that lets the user know the value is invalid.

Use the **SurrogateCharValidator** class to create the validator. Specify a notification type such as **CellStyleNotify** ('**CellStyleNotify Class**' in the on-line documentation). Then use the **AddValidators** ('**AddValidators Method**' in the on-line documentation) method to add the validator to a cell range.

Using Code

The following example uses the surrogate character validator.

CS

```
FarPoint.Win.Spread.CellStyleNotify cnotify = new
FarPoint.Win.Spread.CellStyleNotify();
    cnotify.InvalidCellStyle.BackColor = Color.Aqua;
    FarPoint.Win.Spread.SurrogateCharValidator surchar = new
FarPoint.Win.Spread.SurrogateCharValidator();
    surchar.Actions.Add(cnotify);
    fpSpread1.Sheets[0].AddValidators(new
FarPoint.Win.Spread.Model.CellRange(1, 1, 1, 1), surchar);
    fpSpread1.Sheets[0].Cells[1, 1].Value = "In the game of mahjong
\U0001F01C denotes the Four of circles";
```

VB

```
Dim cnotify As New FarPoint.Win.Spread.CellStyleNotify()
    cnotify.InvalidCellStyle.BackColor = Color.Aqua
    Dim surchar As New FarPoint.Win.Spread.SurrogateCharValidator()
    surchar.Actions.Add(cnotify)
    fpSpread1.Sheets(0).AddValidators(New
FarPoint.Win.Spread.Model.CellRange(1, 1, 1, 1), surchar)
    fpSpread1.Sheets(0).Cells(1, 1).Text = "In the game of mahjong" +
Char.ConvertFromUtf32(&H1F01C) + "denotes the Four Of circles"
```

Using a Text Length Validator

You can create a validator that checks to see if the text length is within a specified range.

A validation error occurs if the value is not valid. You can also create an action, such as adding a bgcolor to the cell, that lets the user know the value is invalid.

Use the **TextLengthValidator** class to create the validator. Specify a notification type such as **CellStyleNotify** ('CellStyleNotify Class' in the on-line documentation). Then use the **AddValidators** ('AddValidators Method' in the on-line documentation) method to add the validator to a cell range.

The following image displays an invalid bgcolor in the cell.

	A	B	C
1			
2		Testing	
3			

Using Code

The following example displays an invalid bgcolor if the cell value contains more than six characters.

CS

```
//Type a text string that contains more than 6 characters in cell 1,1 to see the error
notification
FarPoint.Win.Spread.CellStyleNotify cnotify = new
FarPoint.Win.Spread.CellStyleNotify();
cnotify.InvalidCellStyle.BackColor = Color.Aqua;
FarPoint.Win.Spread.TextLengthValidator tvalid = new
FarPoint.Win.Spread.TextLengthValidator();
tvalid.LengthUnit = FarPoint.Win.Spread.LengthUnit.Char;
tvalid.MaximumLength = 6;
tvalid.MinimumLength = 0;
tvalid.NullIsValid = true;
tvalid.Actions.Add(cnotify);
fpSpread1.Sheets[0].AddValidators(new FarPoint.Win.Spread.Model.CellRange(1, 1, 1, 1),
tvalid);
```

VB

```
'Type a text string that contains more than 6 characters in cell 1,1 to see the error
notification
Dim cnotify As New FarPoint.Win.Spread.CellStyleNotify()
notify.InvalidCellStyle.BackColor = Color.Aqua
Dim tvalid As New FarPoint.Win.Spread.TextLengthValidator()
tvalid.LengthUnit = FarPoint.Win.Spread.LengthUnit.Char
tvalid.MaximumLength = 6
tvalid.MinimumLength = 0
tvalid.NullIsValid = True
tvalid.Actions.Add(cnotify)
fpSpread1.Sheets(0).AddValidators(New FarPoint.Win.Spread.Model.CellRange(1, 1, 1, 1),
tvalid)
```

Validating User Input

You can validate the contents of the cell in a number of ways. Some validation is performed by the Spread component

automatically, based on the type of cell.

You can also use the validator classes to assign a validator to a cell. For more information, refer to **Using Validation in Cells**.

Beyond this, to validate the input from a user, you can look for an event and run a validation routine based on the occurrence of that event. Another simple way to check whether the user enters data that is valid based on the cell type is by using the **IsValid** method, which is available in all the cell type classes.

Cell Type Validation

The cell validates user input and verifies that it fits the requirements of the cell type's format and settings. At run time, the component checks data either when it is entered by the user or code, when the component loses the focus, or both. The component validates data as it is provided, for example, as the user types the data, and when the component loses focus. Users can enter data by typing or pasting; data from code can come from property settings or a database. In general, all these methods of entering data are handled in the same way by the component, which checks the data to determine if it is valid.

Values in the component that are less than the setting of the minimum value (**MinimumValue** or **MinimumDate**, or **MinimumTime** property) are allowed in the component. Components must allow values less than the minimum to let users provide a partial value that is later changed to a value greater than the minimum value. For example, if the **MinimumValue** property is set to 100, and the user tries to change the value to 124 by selecting the existing value and typing, as the user types the value changes to "1", which is less than the allowed minimum value. However, as the user types the value becomes "12" and then "124". The final value is above the allowed minimum value. The value provided by the user is checked to see if it is less than the minimum value when the control loses the focus.

When the control is validating data as it comes into the component, if the user tries to provide invalid data, or invalid data is coming from code or a database, the **UserError** event occurs.

Each cell type has some default restrictions to determine what is valid data. For example, the currency cell type regards a text string of characters such as "abcd" as an invalid value because it expects numeric data. In addition, you can set properties for each cell type that specify valid data settings.

The following table lists the default data allowed in each of the editable cell types.

Cell Type	Default	Valid Data Examples
Currency	Numeric data, which can include characters for the currency symbol, a separator, and a decimal symbol	\$3.45 \$1,234.56 £45
DateTime	Date and time data, which can include separator characters	8/16/2002, Monday, August 05, 2002 4:40 PM
GcDateTime	Date and time data, which can include separator characters	8/16/2002, Monday, August 05, 2002 4:40 PM
GcCharMask or GcMask	Any character is accepted that fits the mask string criteria	
GcComboBox or GcTextBox	Any character is accepted	
GcNumber	Numeric data, which can include characters for a separator and a	3.45

	decimal symbol	1,234.56
GcTimeSpan	TimeSpan data, which can include separator characters	
Hyperlink	Any character is accepted.	
General	Any character is accepted.	
Mask	Any character is accepted that fits the mask string criteria	
Number	Numeric data, which can include characters for a separator and a decimal symbol	3.45 1,234.56
Percent	Numeric data, which can include characters for the percent symbol, a separator, and a decimal symbol	0.5 1,234%
Text	Any character is accepted.	

For more information about differences between these cell types, refer to the **Cell Types**.

The following table lists the additional properties you can set for each cell type that specify valid data settings.

Cell Type Cell Type Properties for Defining Valid Data

Currency	MaximumValue ('MaximumValue Property' in the on-line documentation), MinimumValue ('MinimumValue Property' in the on-line documentation)
DateTime	MaximumDate ('MaximumDate Property' in the on-line documentation), MinimumDate ('MinimumDate Property' in the on-line documentation), MaximumTime ('MaximumTime Property' in the on-line documentation), MinimumTime ('MinimumTime Property' in the on-line documentation)
GcCharMask	FormatString ('FormatString Property' in the on-line documentation)
GcCharMask or GcMask	Pattern ('Pattern Property' in the on-line documentation), MaxLength ('MaxLength Property' in the on-line documentation), MinLength ('MinLength Property' in the on-line documentation)
GcComboBox	MaxLength ('MaxLength Property' in the on-line documentation), MaxLengthUnit ('MaxLengthUnit Property' in the on-line documentation), FormatString ('FormatString Property' in the on-line documentation)
GcDateTime	MaxDate ('MaxDate Property' in the on-line documentation), MinDate ('MinDate Property' in the on-line documentation), MaxMinBehavior ('MaxMinBehavior Property' in the on-line documentation)
GcNumber	MaxValue ('MaxValue Property' in the on-line documentation), MinValue ('MinValue Property' in the on-line documentation), MaxMinBehavior ('MaxMinBehavior Property' in the on-line documentation), ValueSign ('ValueSign Property' in the on-line documentation)
GcTextBox	MaxLength ('MaxLength Property' in the on-line documentation), MaxLengthUnit ('MaxLengthUnit Property' in the on-line documentation), MaxLengthCodePage ('MaxLengthCodePage Property' in the on-line documentation), FormatString ('FormatString Property' in the on-line documentation)
GcTimeSpan	MaxValue ('MaxValue Property' in the on-line documentation), MinValue ('MinValue Property' in the on-line documentation), MaxMinBehavior ('MaxMinBehavior Property' in the on-line documentation), ValueSign ('ValueSign Property' in the on-line documentation)

	documentation)
Number	MaximumValue ('MaximumValue Property' in the on-line documentation), MinimumValue ('MinimumValue Property' in the on-line documentation)
Mask	Mask ('Mask Property' in the on-line documentation), MaskChar ('MaskChar Property' in the on-line documentation)
Percent	MaximumValue ('MaximumValue Property' in the on-line documentation), MinimumValue ('MinimumValue Property' in the on-line documentation)
Text	MaxLength ('MaxLength Property' in the on-line documentation)

The following table lists how invalid data is handled by the number and text cell types. The Column and Cell headings in the table refer to entering the data at the column or cell level.

Action	Text Cell		Number Cell	
	Column	Cell	Column	Cell
Input invalid cell text while cell is in edit mode	+	*	+	*
Paste invalid cell text while cell is in edit mode	#	*	+	*
Paste invalid cell text while cell is not in editmode	#	*	+	*
Paste copied cell with cell type and invalid value while not in edit mode	##	**	##	**
Paste copied cell with invalid value and cell type not set while cell is not in edit mode	#	*	+	*
Input invalid value with cell Value property (or ISheetDataModel SetValue method)	++	**	++	**
Input invalid string value with cell Text property (or SheetView SetText method)	#	**	+	**
Input invalid cell value with SheetView SetValue method (validate = false)	++	**	++	**
Input invalid cell value with SheetView SetValue method (validate = true)	+	**	+	**
Set invalid cell type after binding	++	**	++	**
Use the ClipboardPasteValues field to paste invalid cell value while cell is not in edit mode	#	*	+	*
DragFillMode.Copy (the operation is canceled if the cell is locked)	#	*	+	*
DragFillMode.Series (the operation is canceled if the cell is locked)	#	*	+	*

The following list defines the conditions in the above table:

- + Reject the value
- ++ Allow (paint) the existing invalid value and make the value valid (truncate the value or change it to be within the minimum/maximum setting) when editing
- # Truncate the value

- ## Paste the value and cell type
- * Fire the Error/EditError events
- ** Do not fire the Error/EditError events

Event-based Validation

You can check for an event and run a validation routine based on the occurrence of that event. For instance, the **Changed** ('**Changed Event**' in the on-line documentation) event in the **SheetView** class notifies your application that the user has left edit mode and the contents of the cell has changed. For more thorough validation, to handle the case where a user pastes a value from the Clipboard as opposed to typing in a value, use the **Changed** event on the data model (**DefaultSheetDataModel** class). This is a good way to evaluate the contents of a cell after it has been edited, and throw an error message or revert to original value if the data in the cell is not valid. For more information about using events, refer to **Events from User Actions**.

IsValid Method Validation

The **IsValid** method for the cell type classes checks whether a value is valid for the cell editor. Spread uses that method internally to check values coming out of the cell editor to ensure that they are valid. In most cases, it will return True if the **Format** method is able to format the specified non-string value into a string to display in the editor, or whether the **Parse** method is able to parse the specified string value into a value of the appropriate type for the cell.

For advanced users, you can evaluate the contents of a cell after it has been edited, and throw an error message and revert to the original value if the data in the cell is not valid. To do this requires handling the **EditModeOn** event and setting properties in **SuperEditBase** and thus **GeneralEditor** on the editor control each time edit mode is turned on. These properties include **InvalidOption**, **InvalidColor**, **CanValidate**, and **UserEntry**, but this requires a more involved process. At the time the **EditModeOff** event is raised, the value in the data model has already been changed. Handle **EditModeOn** and store the cell's value and then in the **EditModeOff** event if the current value of the cell fails your validation, reset the value to the stored value.

Customizing the User Error Messages

You can set the error messages that the component displays when the user performs invalid actions. To determine the display of error messages, use the **EditError** ('**EditError Event**' in the on-line documentation) event and the **EditError** ('**EditError Enumeration**' in the on-line documentation) enumeration.

Displaying Error Icons in Cells or Rows

You can display error icons in cells or rows.

Use the **ShowCellErrors** ('**ShowCellErrors Property**' in the on-line documentation) or **ShowRowErrors** ('**ShowRowErrors Property**' in the on-line documentation) property to specify whether you wish to display an error icon or a cell note indicator. Set the **ErrorText** ('**ErrorText Property**' in the on-line documentation) property for the **Cell** ('**Cell Class**' in the on-line documentation) class or **Row** ('**Row Class**' in the on-line documentation) class to specify the cell or row to display the error icon in. Set the value of the **ErrorText** property to "CellError" to display the error icon for a cell or "RowError" to specify the error icon for a row. The following image displays the error icon.

	A	B	C
1			
2		❗	
3			

You can display a cell note indicator instead of the error icon by setting **ShowCellErrors** or **ShowRowErrors** to false. The text you specify with the **ErrorText** property is then displayed in the cell note.

Using Code

1. Set the **ShowCellErrors** ('**ShowCellErrors Property**' in the on-line documentation) or **ShowRowErrors** ('**ShowRowErrors Property**' in the on-line documentation) property.
2. Set the **ErrorText** ('**ErrorText Property**' in the on-line documentation) property for the **Cell** ('**Cell Class**' in the on-line documentation) or **Row** ('**Row Class**' in the on-line documentation) object.

Example

This example displays an error icon in the locked cell.

CS

```
fpSpread1.ShowCellErrors = true;
fpSpread1.Sheets[0].Cells[1, 1].ErrorText = "CellError";
fpSpread1.Sheets[0].Cells[1, 1].Locked = true;
```

;

VB

```
fpSpread1.ShowCellErrors = True
fpSpread1.Sheets(0).Cells(1, 1).ErrorText = "CellError"
fpSpread1.Sheets(0).Cells(1, 1).Locked = True
```

Using Visible Indicators in the Cell

You can customize the user interaction with individual cells (or a range of cells). To customize this aspect of user interaction, you can perform the following tasks:

- **Locating the Pointer Using HitTest**
- **Returning Information for a Clicked Cell**
- **Preventing a Cell from Receiving Focus**
- **Customizing the Focus Indicator for a Cell**

As with most spreadsheets, Spread does not allow in-cell editing of the cells in the row and column headers.

For more information about cells and sheet interactions, refer to the following topics:

Description

For information on adding a formula to a cell
For information on allowing a user to enter a formula in a cell
For information on how to change the appearance of cells
For information on setting the cell type
For information about customizing focus indicators

Refer to

Placing a Formula in Cells
Allowing the User to Enter Formulas
Customizing the Appearance of a Cell
Cell Types
Customizing the Focus Indicator for a Cell

Locating the Pointer Using HitTest

You can locate the pointer at any time in code using the **HitTest** method of the Spread component. Whether you are meeting accessibility standards and displaying information for the user based on pointer location, or want to provide additional support based on pointer location, you can use this capability to customize the display and user interaction.

The list of members corresponding to this capability are listed here:

Component Area	Class Name
Spread component	FpSpread.HitTest ('HitTest Method' in the on-line documentation) method
Spread component	HitTestType ('HitTestType Enumeration' in the on-line documentation) enumeration
Spread component	HitTestInformation ('HitTestInformation Class' in the on-line documentation) class
Row or column header	HeaderHitTestInformation ('HeaderHitTestInformation Class' in the on-line documentation) class
Outline (range group) area	RangeGroupHitTestInformation ('RangeGroupHitTestInformation Class' in the on-line documentation) class
Tab strip	TabStripHitTestInformation ('TabStripHitTestInformation Class' in the on-line documentation) class
Viewport	ViewportHitTestInformation ('ViewportHitTestInformation Class' in the on-line documentation) class

Returning Information for a Clicked Cell

You can get row and column index information for cells that are clicked by accessing the **CellClick ('CellClick Event' in the on-line documentation)** event parameter *e* in the **CellClickEventArgs ('CellClickEventArgs Class' in the on-line documentation)** class. You can get *x*- and *y*-coordinates as well from this parameter. You can implement a **MouseDown** event and from the *x*- and *y*-coordinates you can obtain row and column index information of the clicked cell. With the **GetCellFromPixel ('GetCellFromPixel Method' in the on-line documentation)** method in the **FpSpread ('FpSpread Class' in the on-line documentation)** class, you can get target cell information in the **CellRange ('CellRange Class' in the on-line documentation)** class format. You can obtain row and column information from respective members.

You can obtain cell information such as positions and sizes that have been specified by row and column indexes. When the **GetCellRectangle ('GetCellRectangle Method' in the on-line documentation)** method in the **FpSpread** class is called, specify the target row and column indexes. The cell coordinate information is returned in the .NET framework **Rectangle** format.

For headers, you can get row and column index information of clicked header cells by accessing **CellClick** event parameter *e* in the **CellClickEventArgs** class. You can detect whether the headers have been clicked. You can get *x*- and *y*-coordinates as well from this parameter. You can implement a **MouseDown** event and from the *x*- and *y*-coordinates you can obtain row and column index information of the clicked header cell.

With the **GetColumnHeaderCellFromPixel ('GetColumnHeaderCellFromPixel Method' in the on-line documentation)** method in the **SpreadView ('SpreadView Class' in the on-line documentation)** class, you can get target cell information in the **CellRange** class format for column cells. You can obtain row and column information in column headers from respective members. In the case of row header cells, call the **GetRowHeaderCellFromPixel ('GetRowHeaderCellFromPixel Method' in the on-line documentation)** method.

Preventing a Cell from Receiving Focus

You can prevent cells from receiving focus, and thus not allow the end user to click in that cell. You control focus settings that are defined by keyboard input and mouse operation by setting the **CanFocus ('CanFocus Property' in the on-line documentation)** property in the cell. You can also set this property in the columns and rows shortcut objects and

you can set it in a style that is applied to a group of cells.

Customizing the Focus Indicator for a Cell

The focus rectangle indicates to the end user the selected and active cell. By default, when a cell is selected the cell has a solid focus rectangle as shown in the figure below. If an entire column (or row) is selected, the column (or row) is highlighted, also as shown. The column and row header of the active cell also have a different background color.

	Title	Year	ISBN
1			
2			
3			
4			

Focus indicator for selected individual cell

	Title	Year	ISBN
1			
2			
3			
4			

Focus indicator for selected column

You can customize the focus indicator for the active cell by using the **FocusRenderer** ('**FocusRenderer Property**' in the on-line documentation) property of the Spread component (which uses the **IFocusIndicatorRenderer** ('**IFocusIndicatorRenderer Interface**' in the on-line documentation) interface). For animated indicators, you need the **IAnimatedFocusIndicatorRenderer** ('**IAnimatedFocusIndicatorRenderer Interface**' in the on-line documentation) interface. You can also change the selected background color of the active cell headers.

This table summarized the types of focus indicators and the classes that correspond to them.

Type	Class
Default	DefaultFocusIndicatorRenderer (' DefaultFocusIndicatorRenderer Class ' in the on-line documentation)
Animated	AnimatedDefaultFocusIndicatorRenderer (' AnimatedDefaultFocusIndicatorRenderer Class ' in the on-line documentation)
Custom Line	CustomFocusIndicatorRenderer (' CustomFocusIndicatorRenderer Class ' in the on-line documentation)
Editing	EditingFocusIndicatorRenderer (' EditingFocusIndicatorRenderer Class ' in the on-line documentation)
Enhanced	EnhancedFocusIndicatorRenderer (' EnhancedFocusIndicatorRenderer Class ' in the on-line documentation)
Image	ImageFocusIndicatorRenderer (' ImageFocusIndicatorRenderer Class ' in the on-line documentation)
Marquee Line	MarqueeFocusIndicatorRenderer (' MarqueeFocusIndicatorRenderer Class ' in the on-line documentation)
Solid Line	SolidFocusIndicatorRenderer (' SolidFocusIndicatorRenderer Class ' in the on-line documentation)
Flat	FlatFocusIndicatorRenderer (' FlatFocusIndicatorRenderer Class ' in the on-line documentation)

The **DefaultFocusIndicatorRenderer** ('**DefaultFocusIndicatorRenderer Class**' in the on-line documentation) is the base class for the others. The **ImageFocusIndicatorRenderer** ('**ImageFocusIndicatorRenderer Class**' in the on-line documentation) allows you to use an image as the focus indicator. The **SolidFocusIndicatorRenderer** ('**SolidFocusIndicatorRenderer Class**' in the on-line documentation) allows you to customize a solid border around the selected cell as a focus indicator.

In the Spread Designer, you can customize the focus indicator with the **Focus Indicator Editor**. For more information about using the Spread Designer, refer to the **Focus Indicator Editor (on-line documentation)** topic in the Spread Designer Guide.

Using Code

Use the **IFocusIndicatorRenderer** (**'IFocusIndicatorRenderer Interface' in the on-line documentation**) interface to create custom indicators for the active cell.

Example

This example creates a custom focus indicator for the active cell.

C#

```
FarPoint.Win.Spread.SolidFocusIndicatorRenderer sfir =new
FarPoint.Win.Spread.SolidFocusIndicatorRenderer(Color.Blue, 2);
fpSpread1.FocusRenderer = sfir;
// Create a custom indicator.
public class MyIndicator : FarPoint.Win.Spread.IFocusIndicatorRenderer
{
public void Paint(System.Drawing.Graphics g, int x, int y, int width, int height, bool
left, bool top, bool right, bool bottom)
{
SolidBrush r = new SolidBrush(System.Drawing.Color.Red);
SolidBrush b = new SolidBrush(System.Drawing.Color.Blue);
SolidBrush gr = new SolidBrush(System.Drawing.Color.DarkGreen);
g.FillRectangle(r, x, y, 1, height);
g.FillRectangle(gr, x, y, width, 1);
g.FillRectangle(r, x + width - 1, y, 1, height);
g.FillRectangle(b, x, y + height - 1, width, 1);
}
}

fpSpread1.FocusRenderer = new MyIndicator();
```

VB

```
Dim sfir As New FarPoint.Win.Spread.SolidFocusIndicatorRenderer(Color.Blue, 2)
fpSpread1.FocusRenderer = sfir
' Create a custom indicator
Public Class MyIndicator
Implements FarPoint.Win.Spread.IFocusIndicatorRenderer
Public Sub Paint(ByVal g As System.Drawing.Graphics, ByVal x As Integer, ByVal y As
Integer, ByVal width As Integer, ByVal height As Integer, ByVal left As Boolean, ByVal
top As Boolean, ByVal right As Boolean, ByVal bottom As Boolean) Implements
FarPoint.Win.Spread.IFocusIndicatorRenderer.Paint
Dim r As New SolidBrush(Color.Red)
Dim b As New SolidBrush(Color.Blue)
Dim gr As New SolidBrush(Color.DarkGreen)
g.FillRectangle(r, x, y, 1, height)
g.FillRectangle(gr, x, y, width, 1)
g.FillRectangle(r, x + width - 1, y, 1, height)
g.FillRectangle(b, x, y + height - 1, width, 1)
End Sub
End Class
```

```
fpSpread1.FocusRenderer = New MyIndicator()  
  
)
```

Using the Spread Designer

1. Select the **Format** menu option.
2. Choose the **Focus Indicator Editor** menu.
3. Make changes to the dialog and select **OK**.

Using Code

Use the **SelectedBackgroundColor** (**'SelectedBackgroundColor Property' in the on-line documentation**) property of the **EnhancedColumnHeaderRenderer** (or row) to set the header backcolor of the active cell.

Example

This example changes the header background color for the active cell.

C#

```
FarPoint.Win.Spread.CellType.EnhancedColumnHeaderRenderer testing = new  
FarPoint.Win.Spread.CellType.EnhancedColumnHeaderRenderer();  
testing.SelectedBackgroundColor = Color.MediumTurquoise;  
FarPoint.Win.Spread.CellType.EnhancedRowHeaderRenderer testing1 = new  
FarPoint.Win.Spread.CellType.EnhancedRowHeaderRenderer();  
testing1.SelectedBackgroundColor = Color.MediumTurquoise;  
fpSpread1.Sheets[0].ColumnHeader.DefaultStyle.Renderer = testing;  
fpSpread1.Sheets[0].RowHeader.DefaultStyle.Renderer = testing1;
```

VB

```
Dim testing as New FarPoint.Win.Spread.CellType.EnhancedColumnHeaderRenderer  
testing.SelectedBackgroundColor = Color.MediumTurquoise  
Dim testing1 as New FarPoint.Win.Spread.CellType.EnhancedRowHeaderRenderer  
testing1.SelectedBackgroundColor = Color.MediumTurquoise  
fpSpread1.Sheets(0).ColumnHeader.DefaultStyle.Renderer = testing  
fpSpread1.Sheets(0).RowHeader.DefaultStyle.Renderer = testing1
```

Customizing Undo and Redo Actions

With the undo/redo feature, you can add capability to your application to undo various actions in the spreadsheet performed by your end user. You can use the **UndoAction** (**'UndoAction Class' in the on-line documentation**) class and several specific classes that correspond with various user actions. There is also a manager class that keeps track of the end user actions that can be undone and re-done.

The **SpreadView** (**'SpreadView Class' in the on-line documentation**) class and **FpSpread** (**'FpSpread Class' in the on-line documentation**) class have properties, **AllowUndo** and **UndoManager**, which turn on and off the undo/redo feature and return the **UndoManager** for that **SpreadView** instance, respectively. Each **SpreadView** has its own **UndoManager**.

Assigning the Actions

The **UndoAction** (**'UndoAction Class' in the on-line documentation**) class is an abstract class that inherits from

Action and adds new methods to the class: **Undo** and **SaveUndoState**. It also inherits the **PerformAction** method from **Action**.

SaveUndoState is used to save undo state information (in fields of the class). **PerformAction** is used to perform the action. **Undo** is used to reverse the action (using the undo state information in the fields).

Each of the classes inheriting from **UndoAction** is designed to do one specific action (for example, edit a cell, resize a column, move a range, and so on), and to undo that action. All relevant information to do that action should be passed into the constructor for the object, and all relevant information to undo that action should be stored in the **SaveUndoState** implementation. Once the **UndoAction** object is created, the variables of that specific action are fixed (specified by the values passed to the constructor). For example, edit cell A1 in sheet1 and change the value to "test", resize column B to 24 pixels, and move the range C4:F6 to A1:D:3. The action can only do that specific action in that specific way.

Managing the Actions

The **UndoManager** ('**UndoManager Class**' in the on-line documentation) class manages the undo and redo stacks. It keeps track of which actions have been done and undone, and in what order. An **UndoAction** must be passed into the **PerformUndoAction** method of **UndoManager** to do the action in order for it to be undoable by the **UndoManager**. When that happens, the **UndoManager** pushes the **UndoAction** onto the undo stack and calls **PerformAction** on the **UndoAction**, and then the **CanUndo** method returns true (indicating there is something to undo). When **CanUndo** returns false, that means the undo stack is empty, and there is no action ready to undo. You might want to use this to disable the **Undo** menu item in the **Edit** menu, for example, if your application has an **Edit** menu.

When an action is ready to undo, you can call **Undo** on the **UndoManager**, and it moves the last action performed from the undo stack to the redo stack, and calls **Undo** on the action, and then the **CanRedo** method returns True (indicating there is something to redo).

When **CanRedo** returns False, that means the redo stack is empty, and there is no action ready to redo. You might want to use this to disable the Redo menu item in the Edit menu, for example, if your application has Edit menu.

When an action is ready to redo, you can call **Redo** on the **UndoManager**, and it moves the last action undone from the redo stack to the undo stack, and calls **PerformAction** on the action, and the **CanUndo** method returns true.

You can call **PerformAction** on the **UndoManager** with a sequence of **UndoAction** objects, and it performs each action in sequence, and remembers each action and the order in which they are done. Then you can call **Undo** to undo some of those actions, and each can be re-done with **Redo** (and then un-done again with **Undo**).

But, when you call **PerformAction** to perform a new action, if there are any actions pending in the redo stack, those actions are cleared, and **CanRedo** returns False (that is, once you perform a new action, you will not be able to redo any actions that you have undone with **Undo**). That is why the **PushUndo** method in the **UndoManager** class has a flag to indicate whether the redo stack should be cleared when the action is pushed onto the undo stack.

Some of the **UndoAction** classes will be replacing **Action** objects in the action maps, so that those actions are routed through the **UndoManager** and become undoable. Other **UndoAction** classes will not be part of the action maps, but instead are used in the **SheetView** or **SpreadView** code to make the action undoable.

Other API Updates

The input maps include new items to map the Ctrl+Z and Ctrl+Y keys to the new **UndoAction** and **RedoAction** action objects, respectively. These actions make calls into the **UndoManager** to the **Undo** and **Redo** methods, respectively.

Customizing Interaction Based on Events

You can customize how the Spread component responds to user-initiated events. In the **FpSpread** ('**FpSpread Class**' in the on-line documentation) class there are several events, from **ButtonClicked** ('**ButtonClicked Event**' in the on-line documentation) to **LeaveCell** ('**LeaveCell Event**' in the on-line documentation) to **SelectionChanged** ('**SelectionChanged Event**' in the on-line documentation). Use events that correspond to

user actions to initiate responses. For a list of events in the component, refer to the **FpSpread ('FpSpread Class' in the on-line documentation)** class members. For events available for the sheet, refer to the **SheetView ('SheetView Class' in the on-line documentation)** class members. For a list of events that can be used while in edit mode, refer to the **FarPoint.Win.SuperEditBase ('FarPoint.Win.SuperEdit Namespace' in the on-line documentation)** class.

The **FpSpread** class is derived from the **Control** class that has the following properties and events that are relevant to our understanding of events in Spread:

- **Text** property and **TextChanged** event
- **Click** event
- **Enter** event

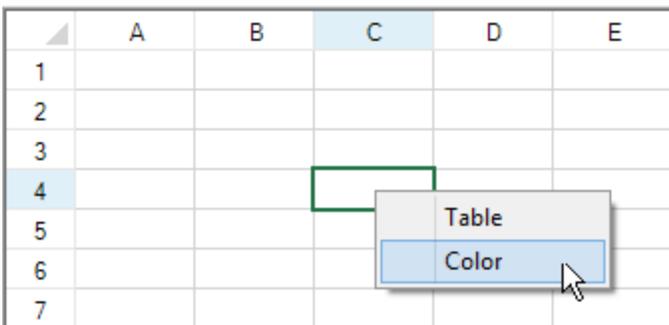
The **Text** property and **TextChanged** event are used by simple controls that have a single **Text** attribute (for example, the **TextBox** control). The Spread component is a more complex control that consists of rows and columns of cells. Each cell has its own **Text** property. The **Text** property of the cell is separate from the **Text** property of the Spread component. Since the Spread component does not use the component's **Text** property, the **TextChange** event is never raised.

The **Click** event is used by simple controls that have a single area (for example, the **Button** control). The Spread component is a more complex control that consists of rows and columns of cells. The Spread component raises a **CellClick** event instead of a **Click** event. The **CellClick ('CellClick Event' in the on-line documentation)** event contains more detailed information than the **Click** event.

The **Enter** event is raised when keyboard focus is moved from another control on the form to the Spread component.

Adding a Context Menu to a Component

You can create a context menu and add it to the **ContextMenu** property of the **FpSpread ('FpSpread Class' in the on-line documentation)** component (which is inherited from the **System.Windows.Forms.Control**). The component automatically displays this menu of context-specific menu options when you right click on the component. A context menu is also known as a shortcut menu. For more information, refer to the Microsoft .NET documentation about context menu (or shortcut menu). The figure shows a context menu with two choices. The code for this figure is shown in the example.



The scroll bars have, by default, a context menu of their own.

Using Code

1. Add a context menu using the **ContextMenu** property.
2. Define the menu items.

 At design time, you could also drop in a Context Menu from the Toolbox and look at the code generated by that to learn more.

Example

This example creates a context menu.

C#

```
ContextMenu custommenu = new ContextMenu();
custommenu.MenuItems.Add("&Table");
custommenu.MenuItems.Add("&Color", new EventHandler(ContextMenu_Color));
fpSpread1.ContextMenu = custommenu;

private void ContextMenu_Color(object sender, System.EventArgs e)
{
    MessageBox.Show("You chose color.");
}
```

VB

```
Dim custommenu As New ContextMenu
custommenu.MenuItems.Add("&Table")
custommenu.MenuItems.Add("&Color", New EventHandler(AddressOf ContextMenu_Color))
fpSpread1.ContextMenu = custommenu

Private Sub ContextMenu_Color(ByVal sender As Object, ByVal e As System.EventArgs)
    MsgBox("You chose color.")
End Sub
```

Tables

You can create a table from a range of cells to make managing and analyzing a group of related data easier. A table typically contains related data in rows and columns. You can manage the data in the table rows and columns independently from the data in other rows and columns on the sheet.

A table can contain a header row, banded rows, calculated columns, a total row, and a sizing handle. The header row contains icons that allow you to filter or sort the table data quickly. Banded rows are alternate rows that have shading applied so that the data is easier to view. You can create a calculated column by entering a formula in one cell in a table column. This creates a calculated column in which that formula is instantly applied to all other cells in that table column. You can add a total row to your table that provides access to summary functions (such as the AVERAGE, COUNT, or SUM function). A drop-down list appears in each total row cell so that you can quickly calculate the totals that you want. A sizing handle in the lower-right corner of the table allows you to change the table size.

The following image illustrates the main table elements.

The diagram shows a table with the following structure:

Last Name	Value
Smith	50
Vil	10
Press	78
Total	138

Labels in the diagram point to the following elements:

- Column with data:** Points to the 'Value' column.
- Header row:** Points to the first row containing 'Last Name' and 'Value'.
- Banded rows:** Points to the rows containing 'Smith', 'Vil', and 'Press'.
- Total row:** Points to the bottom row containing 'Total' and '138'.
- Sizing handle:** Points to the bottom-right corner of the table.

You can also move tables and create structured references in tables.

For more information, see the following topics:

- **Adding a Table**
- **Using Table Filters**
- **Resizing a Table**
- **Sorting a Table**
- **Setting Table Styles**
- **Binding a Table**
- **Adding a Table Formula**
- **Understanding Structured References**

Adding a Table

You can add a table to a sheet using code or the designer. You can type data in the table cells or add text to the cells with the **Text** ('**Text Property**' in the on-line documentation) or **Value** ('**Value Property**' in the on-line documentation) property.

	A	B	C	D
1				
2		Last Name	Value	
3		Smith	50	
4		Vil	10	
5		Press	78	
6				

Using Code

Use the **AddTable** ('AddTable Method' in the on-line documentation) method to add a table to a sheet.

Example

This example code adds a table using cell data.

C#

```
fpSpread1.Sheets[0].Cells[1, 1].Text = "Last Name";
fpSpread1.Sheets[0].Cells[1, 2].Text = "Value";
fpSpread1.Sheets[0].Cells[2, 1].Text = "Smith";
fpSpread1.Sheets[0].Cells[2, 2].Value = 50;
fpSpread1.Sheets[0].Cells[3, 1].Text = "Vil";
fpSpread1.Sheets[0].Cells[3, 2].Value = 10;
fpSpread1.Sheets[0].Cells[4, 1].Text = "Press";
fpSpread1.Sheets[0].Cells[4, 2].Value = 78;
fpSpread1.Sheets[0].AddTable("table", 1, 1, 5, 2);
```

VB

```
fpSpread1.Sheets(0).Cells(1, 1).Text = "Last Name"
fpSpread1.Sheets(0).Cells(1, 2).Text = "Value"
fpSpread1.Sheets(0).Cells(2, 1).Text = "Smith"
fpSpread1.Sheets(0).Cells(2, 2).Value = 50
fpSpread1.Sheets(0).Cells(3, 1).Text = "Vil"
fpSpread1.Sheets(0).Cells(3, 2).Value = 10
fpSpread1.Sheets(0).Cells(4, 1).Text = "Press"
fpSpread1.Sheets(0).Cells(4, 2).Value = 78
fpSpread1.Sheets(0).AddTable("table", 1, 1, 5, 2)
```

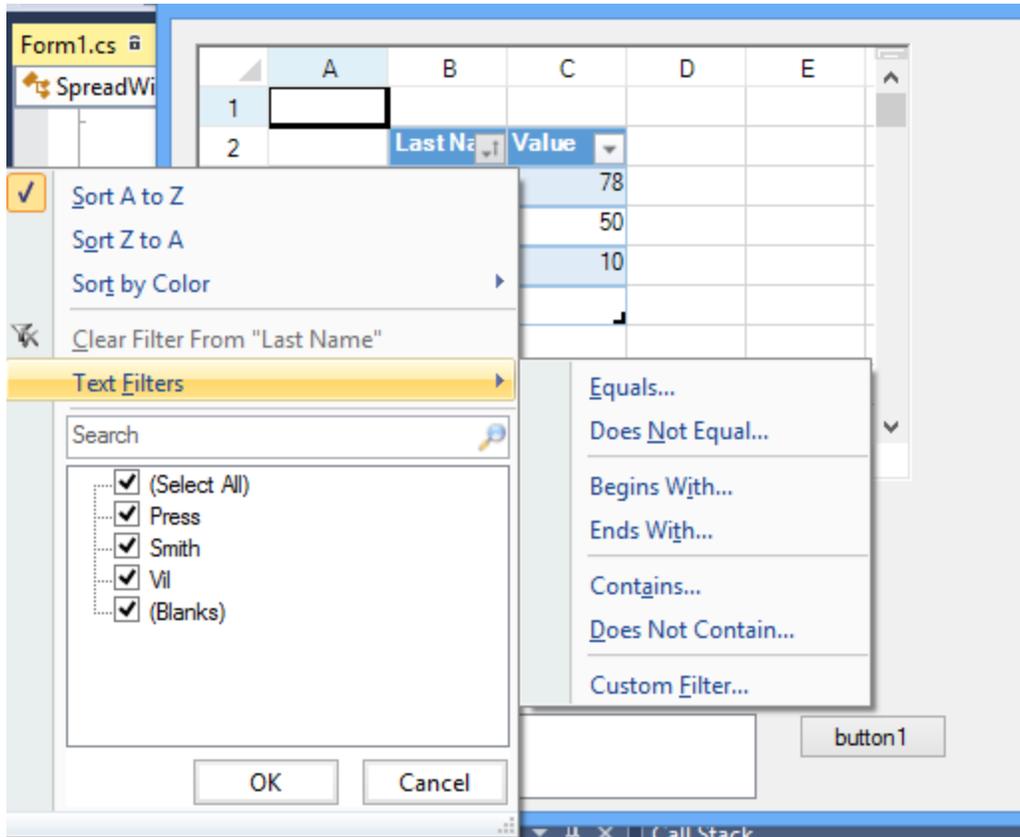
Using the Spread Designer

1. In the work area, select the cell range where you want to add the table.
2. From the **Insert** menu, select **Table**.
3. Provide the cell range for the table and select **OK**.
4. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

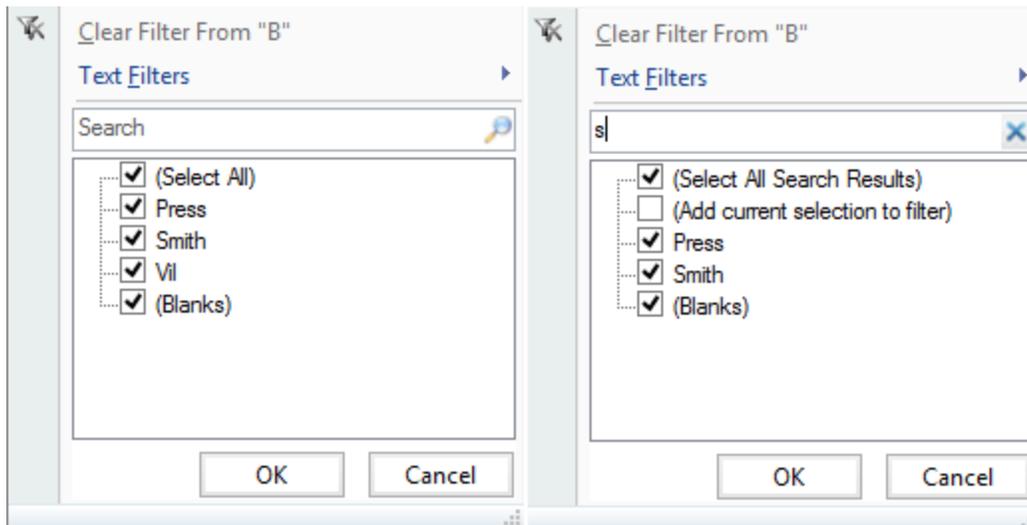
Using Table Filters

You can use enhanced filtering with tables.

The default filter that is displayed depends on the data in the column. The filter can be a number, text, date, or color filter. The following image displays the menus for setting up a text filter.



You can also type in the search box in the filter dialog to change the list of filter options. The following image displays the search box and the filter choices after typing characters in the search box.



The filters are described in the following table.

Type of Filters	Description
Number Filters	
Equals	Values in rows are equal to condition

Does Not Equal	Values in rows do not equal condition
Greater Than	Values in rows are greater than condition
Greater Than Or Equal To	Values in rows are greater than or equal to condition
Less Than	Values in rows are less than condition
Less Than Or Equal To	Values in rows are less than or equal to condition
Between	Values in rows are greater than one condition and less than another condition
Top 10	Values in the rows with the ten highest values
Above Average	Values in the rows that are above the average of the values in all the rows
Below Average	Values in the rows that are below the average of the values in all the rows
Custom Filter	Values in rows that meet the conditions of a custom filter

Text Filters

Equals	Values in rows equal the condition
Does Not Equal	Values in rows do not equal the condition
Begins With	Values in rows begin with the specified characters
Ends With	Values in rows end with the specified characters
Contains	Values in rows contain the specified characters
Does Not Contain	Values in rows do not contain the specified characters
Custom Filter	Values in rows that meet the conditions of a custom filter

Date Filters

Equals	Values in rows equal the condition
Before	Values in rows are dates before the condition
After	Values in rows are dates after the condition
Between	Values in rows are dates between two specified dates for the condition
Tomorrow	Values in rows are tomorrow's date
Today	Values in rows are today's date
Yesterday	Values in rows are yesterday's date
Next Week	Values in rows are during next week
This Week	Values in rows are during current week
Last Week	Values in rows are during last week
Next Month	Values in rows are during next month
This Month	Values in rows are during current month
Last Month	Values in rows are during last month
Next Quarter	Values in rows are during next quarter
This Quarter	Values in rows are during current quarter
Last Quarter	Values in rows are during last quarter
Next Year	Values in rows are during next year
This Year	Values in rows are during current year

Last Year	Values in rows are during last year
Year to Date	Values in rows are during current year to present date
All Dates in the Period	Values in rows are within a specified period
Custom Filter	Values in rows that meet the conditions of a custom filter

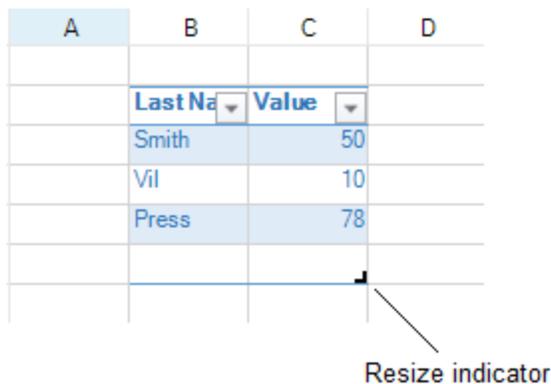
Users can specify wildcards in conditions. The "?" character represents any single character. The "*" character represents any series of characters.

When the user chooses a filter, the table filters the data to display only the rows that match the filter criteria.

You can use the **Filter ('Filter Method' in the on-line documentation)** method to filter a table using code. You can reset a filter by setting null in the **Filter ('Filter Method' in the on-line documentation)** method (for example, `table.Filter(3, null)`; resets the filter in the third column).

Resizing a Table

You can resize the table with the resize indicator in the bottom, right corner of the table or you can use code to do so.



A	B	C	D
	Last Name	Value	
	Smith	50	
	Vil	10	
	Press	78	

Resize indicator

Select the indicator and drag to the right to add columns or down to add rows.

Using Code

You can use the **Resize ('Resize Method' in the on-line documentation)** method to add columns or rows to a table. The below example code adds rows and columns to the table.

C#

```
FarPoint.Win.Spread.TableStyle tstyle = fpSpread1.CreateTableStyle("Style1",
FarPoint.Win.Spread.TableStyle.TableStyleLight2);

fpSpread1.Sheets[0].Cells[1, 1].Text = "Last Name";
fpSpread1.Sheets[0].Cells[1, 2].Text = "Value";
fpSpread1.Sheets[0].Cells[2, 1].Text = "Smith";
fpSpread1.Sheets[0].Cells[2, 2].Value = 50;
fpSpread1.Sheets[0].Cells[3, 1].Text = "Vil";
fpSpread1.Sheets[0].Cells[3, 2].Value = 10;
fpSpread1.Sheets[0].Cells[4, 1].Text = "Press";
fpSpread1.Sheets[0].Cells[4, 2].Value = 78;
fpSpread1.TableStyleCollection.Add(tstyle);

FarPoint.Win.Spread.TableView table = fpSpread1.Sheets[0].AddTable("table", 1, 1, 5, 2,
"Style1");
```

```
table.Resize(6, 2);
```

Visual Basic

```
Dim tstyle As FarPoint.Win.Spread.TableStyle
tstyle = FpSpread1.CreateTableStyle("Style1",
FarPoint.Win.Spread.TableStyle.TableStyleLight2)
FpSpread1.Sheets(0).Cells(1, 1).Text = "Last Name"
FpSpread1.Sheets(0).Cells(1, 2).Text = "Value"
FpSpread1.Sheets(0).Cells(2, 1).Text = "Smith"
FpSpread1.Sheets(0).Cells(2, 2).Value = 50
FpSpread1.Sheets(0).Cells(3, 1).Text = "Vil"
FpSpread1.Sheets(0).Cells(3, 2).Value = 10
FpSpread1.Sheets(0).Cells(4, 1).Text = "Press"
FpSpread1.Sheets(0).Cells(4, 2).Value = 78
FpSpread1.TableStyleCollection.Add(tstyle)
```

```
Dim table As FarPoint.Win.Spread.TableView = FpSpread1.Sheets(0).AddTable("table", 1,
1, 5, 2, "Style1")
```

```
table.Resize(6, 2)
```

Automatically Extend Table Rows or Columns

You can expand the rows or columns of a table automatically using the **AutoExpandTable** ('AutoExpandTable Property' in the on-line documentation) property.

A new row or column is added after editing an empty cell located below the row or right of the column.

A	B	C	D
	Last Na	Value	
	Smith	50	
	Vil	10	
	Press	78	
	Dias	35	

The **AutoExpandTable** property will not expand table rows if the table has a **Total row**, that is, consists a formula cell in the last row.

C#

```
fpSpread1.Features.AutoExpandTable = true;
```

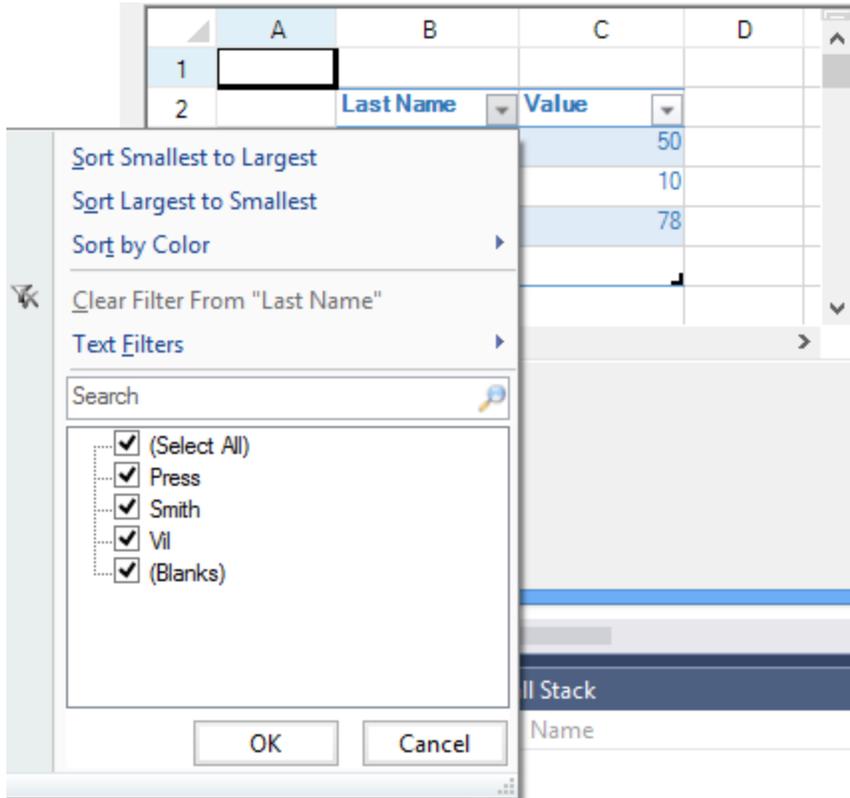
Visual Basic

```
FpSpread1.Features.AutoExpandTable = True
```

Sorting a Table

You can sort a table by selecting the drop-down icon and selecting a sort option, as shown in the following figure, or by

using code.



Using Code

Use the **Sort ('Sort Method' in the on-line documentation)** method to sort columns in a table.

Example

This example code sorts the column.

C#

```
FarPoint.Win.Spread.TableStyle tstyle = fpSpread1.CreateTableStyle("Style1",
FarPoint.Win.Spread.TableStyle.TableStyleLight2);
fpSpread1.Sheets[0].Cells[1, 1].Text = "Last Name";
fpSpread1.Sheets[0].Cells[1, 2].Text = "Value";
fpSpread1.Sheets[0].Cells[2, 1].Text = "Smith";
fpSpread1.Sheets[0].Cells[2, 2].Value = 50;
fpSpread1.Sheets[0].Cells[3, 1].Text = "Vil";
fpSpread1.Sheets[0].Cells[3, 2].Value = 10;
fpSpread1.Sheets[0].Cells[4, 1].Text = "Press";
fpSpread1.Sheets[0].Cells[4, 2].Value = 78;
fpSpread1.TableStyleCollection.Add(tstyle);
FarPoint.Win.Spread.TableView table = fpSpread1.Sheets[0].AddTable("table", 1, 1, 5, 2,
"Style1");
FarPoint.Win.Spread.ComplexSortInfo[] sort = new
FarPoint.Win.Spread.ComplexSortInfo[1];
sort[0] = new FarPoint.Win.Spread.ComplexSortInfo(1, true);
table.Sort(sort);
```

VB

```
Dim tstyle As FarPoint.Win.Spread.TableStyle
tstyle = fpSpread1.CreateTableStyle("Style1",
FarPoint.Win.Spread.TableStyle.TableStyleLight2)
fpSpread1.Sheets(0).Cells(1, 1).Text = "Last Name"
fpSpread1.Sheets(0).Cells(1, 2).Text = "Value"
fpSpread1.Sheets(0).Cells(2, 1).Text = "Smith"
fpSpread1.Sheets(0).Cells(2, 2).Value = 50
fpSpread1.Sheets(0).Cells(3, 1).Text = "Vil"
fpSpread1.Sheets(0).Cells(3, 2).Value = 10
fpSpread1.Sheets(0).Cells(4, 1).Text = "Press"
fpSpread1.Sheets(0).Cells(4, 2).Value = 78
fpSpread1.TableStyleCollection.Add(tstyle)
Dim table As FarPoint.Win.Spread.TableView = fpSpread1.Sheets(0).AddTable("table", 1,
1, 5, 2, "Style1")
Dim sort As FarPoint.Win.Spread.ComplexSortInfo() = New
FarPoint.Win.Spread.ComplexSortInfo(0) {}
sort(0) = New FarPoint.Win.Spread.ComplexSortInfo(1, True)
table.Sort(sort)
```

Setting Table Styles

You can add custom or built-in styles to a table.

	A	B	C	D
1				
2		Last Name	Value	
3		Smith	50	
4		Vil	10	
5		Press	78	
6				
7				

You can specify custom styles for the first, second, or last column as well as other areas of the table. For a complete list, see the **TableStyle ('TableStyle Class' in the on-line documentation)** properties. You can specify a built-in style with the **TableStyle** fields.

Table styles have a priority order when the styles overlap. The priority from highest to lowest is cell, row, column, and table.

Some style properties apply to areas that are not visible or do not have a style setting by default. For example, the **FirstRowStripe** style is not displayed unless the **BandedRows ('BandedRows Property' in the on-line documentation)** property is true. The following table lists the **TableView** setting that must be true before the associated table style is displayed in the table.

TableView property	TableStyle property
BandedColumns ('BandedColumns Property' in the on-line documentation)	FirstColumnStripe ('FirstColumnStripe Property' in the on-line documentation), FirstColumnStripSize ('FirstColumnStripSize Property' in the on-line documentation), SecondColumnStripe ('SecondColumnStripe Property' in the on-line documentation), SecondColumnStripSize ('SecondColumnStripSize Property' in the on-line documentation)

BandedRows ('BandedRows Property' in the on-line documentation)	FirstRowStripe ('FirstRowStripe Property' in the on-line documentation), FirstRowStripSize ('FirstRowStripSize Property' in the on-line documentation), SecondRowStripe ('SecondRowStripe Property' in the on-line documentation), SecondRowStripSize ('SecondRowStripSize Property' in the on-line documentation)
FirstColumn ('FirstColumn Property' in the on-line documentation)	FirstColumn ('FirstColumn Property' in the on-line documentation)
HeaderRowVisible ('HeaderRowVisible Property' in the on-line documentation)	HeaderRow ('HeaderRow Property' in the on-line documentation)
LastColumn ('LastColumn Property' in the on-line documentation)	LastColumn ('LastColumn Property' in the on-line documentation)

Using Code

1. Create a style using **TableBorder** (**'TableBorder Class'** in the on-line documentation) and **TableElementStyle** (**'TableElementStyle Class'** in the on-line documentation).
2. Use the **TableStyle** (**'TableStyle Constructor'** in the on-line documentation) constructor and the **CreateTableStyle** (**'CreateTableStyle Method'** in the on-line documentation) method to assign the style.
3. Set the **TableStyle FirstColumn** (**'FirstColumn Property'** in the on-line documentation) property to assign the style to the column or set any of the **TableStyle** properties.
4. Set the **TableView FirstColumn** (**'FirstColumn Property'** in the on-line documentation) property to True to display the column style or set the appropriate **TableView** property.

Example

This example code adds a custom style to the first column.

C#

```
FarPoint.Win.ComplexBorderSide bside = new
FarPoint.Win.ComplexBorderSide(Color.Yellow);
FarPoint.Win.Spread.TableBorder tborder = new FarPoint.Win.Spread.TableBorder(bside);
FarPoint.Win.Spread.TableElementStyle testyle = new
FarPoint.Win.Spread.TableElementStyle(tborder, Color.Red, Color.Blue,
FarPoint.Win.Spread.RegularBoldItalicFontStyle.Bold);
FarPoint.Win.Spread.TableStyle tstyle = fpSpread1.CreateTableStyle("Style1",
FarPoint.Win.Spread.TableStyle.TableStyleLight2);
tstyle.FirstColumn = testyle;
fpSpread1.Sheets[0].Cells[1, 1].Text = "Last Name";
fpSpread1.Sheets[0].Cells[1, 2].Text = "Value";
fpSpread1.Sheets[0].Cells[2, 1].Text = "Smith";
fpSpread1.Sheets[0].Cells[2, 2].Value = 50;
fpSpread1.Sheets[0].Cells[3, 1].Text = "Vil";
```

```
fpSpread1.Sheets[0].Cells[3, 2].Value = 10;
fpSpread1.Sheets[0].Cells[4, 1].Text = "Press";
fpSpread1.Sheets[0].Cells[4, 2].Value = 78;
fpSpread1.TableStyleCollection.Add(tstyle);
FarPoint.Win.Spread.TableView table = fpSpread1.Sheets[0].AddTable("table", 1, 1, 5, 2,
"Style1");
table.FirstColumn = true;
```

VB

```
Dim bside As New FarPoint.Win.ComplexBorderSide(Color.Yellow)
Dim tborder As New FarPoint.Win.Spread.TableBorder(bsite)
Dim testyle As New FarPoint.Win.Spread.TableElementStyle(tborder, Color.Red,
Color.Blue, FarPoint.Win.Spread.RegularBoldItalicFontStyle.Bold)
Dim tstyle As FarPoint.Win.Spread.TableStyle
tstyle = fpSpread1.CreateTableStyle("Style1",
FarPoint.Win.Spread.TableStyle.TableStyleLight2)
tstyle.FirstColumn = testyle
pSpread1.Sheets(0).Cells(1, 1).Text = "Last Name"
fpSpread1.Sheets(0).Cells(1, 2).Text = "Value"
fpSpread1.Sheets(0).Cells(2, 1).Text = "Smith"
fpSpread1.Sheets(0).Cells(2, 2).Value = 50
fpSpread1.Sheets(0).Cells(3, 1).Text = "Vil"
fpSpread1.Sheets(0).Cells(3, 2).Value = 10
fpSpread1.Sheets(0).Cells(4, 1).Text = "Press"
fpSpread1.Sheets(0).Cells(4, 2).Value = 78
fpSpread1.TableStyleCollection.Add(tstyle)
Dim table As FarPoint.Win.Spread.TableView = fpSpread1.Sheets(0).AddTable("table", 1,
1, 5, 2, "Style1")
table.FirstColumn = True
```

Binding a Table

Spread for Winforms allows you to bind a table to a data source using cell-level binding.

Table binding can be done in two ways:

- Automatically generate columns for a table using **ITable Interface (on-line documentation)**.
- Manually set data fields in a table using **ITableColumn Interface (on-line documentation)**.

A data-bound table shows the following behavior:

- You can resize the table by columns but not by rows.
- Setting a value changes the data source.
- Adding or deleting rows changes the data source.
- Adding or deleting columns does not change the data source.
- Removing, moving, or resizing the table does not change the data source.
- Formulas are not saved to the data source.
- The value of existing cells will be cleared if they belong to the binding data range (table range).
- If **ITable.AutoGenerateColumns ('AutoGenerateColumns Property' in the on-line documentation)** is false, binding adjusts the table row count automatically (but column count stays the same). Otherwise, all table columns are regenerated.

Binding Tables Automatically

You can bind a table to a data source automatically by using the **ITable Interface (on-line documentation)** and its

members.

The **AutoGenerateColumns** ('**AutoGenerateColumns Property**' in the on-line documentation) property, as the name suggests, automatically generates columns. The **DataSource** ('**DataSource Property**' in the on-line documentation) property is used to set the data source.

 Note: **AutoGenerateColumns** ('**AutoGenerateColumns Property**' in the on-line documentation) property must be assigned before the **DataSource** ('**DataSource Property**' in the on-line documentation) property. Otherwise, current table columns will be kept.

Consider the following example where the finance department of a company maintains a database of its employees' work-related travel details such as flight ID, flight date, source and destination. The database can be loaded in a table automatically to display travel details.

	A	B	C	D	E	F	G
1	For Finance Only:						
2	Passenger Name	Department	Ticket Type	Flight ID	Flight Date	Flight Src	Flight Dest
3	Mark	Sales	Economy	7855	11-10-2021	New York	Tokyo
4	Sophie	Services	Economy	7426	12-10-2021	London	Venice
5	Oliver	Finance	Business	7641	15-10-2021	New Delhi	Moscow
6	James	R&D	Economy	7293	18-10-2021	Beijing	Dubai
7	Emma	Marketing	Business	7117	20-10-2021	Paris	Seoul

C#

```
GrapeCity.Spreadsheet.IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;
```

```
// Create table data
DataSet ds = new DataSet();
DataTable flightDetails = new DataTable("FlightDetails");
flightDetails.Columns.Add("Passenger Name");
flightDetails.Columns.Add("Department");
flightDetails.Columns.Add("Ticket Type");
flightDetails.Columns.Add("Flight ID");
flightDetails.Columns.Add("Flight Date");
flightDetails.Columns.Add("Flight Src");
flightDetails.Columns.Add("Flight Dest");
flightDetails.Rows.Add("Mark", "Sales", "Economy", 7855, new DateTime(2021, 10,
11).ToShortDateString(), "New York", "Tokyo");
flightDetails.Rows.Add("Sophie", "Services", "Economy", 7426, new DateTime(2021, 10,
12).ToShortDateString(), "London", "Venice");
flightDetails.Rows.Add("Oliver", "Finance", "Business", 7641, new DateTime(2021, 10,
15).ToShortDateString(), "New Delhi", "Moscow");
flightDetails.Rows.Add("James", "R&D", "Economy", 7293, new DateTime(2021, 10,
18).ToShortDateString(), "Beijing", "Dubai");
flightDetails.Rows.Add("Emma", "Marketing", "Business", 7117, new DateTime(2021, 10,
20).ToShortDateString(), "Paris", "Seoul");

ds.Tables.Add(flightDetails);

// Create table
TestActiveSheet.Cells["A1"].Value = "For Finance Only:";
GrapeCity.Spreadsheet.ITable table = TestActiveSheet.Range("A2:G7").CreateTable(true);

// Set auto generate columns
table.AutoGenerateColumns = true;
```

```
// Bind table to datasource
table.DataSource = ds;
```

Visual Basic

```
Dim TestActiveSheet As GrapeCity.Spreadsheet.IWorksheet =
FpSpread1.AsWorkbook().ActiveSheet

'Create table data
Dim ds As DataSet = New DataSet()
Dim flightDetails As DataTable = New DataTable("FlightDetails")
flightDetails.Columns.Add("Passenger Name")
flightDetails.Columns.Add("Department")
flightDetails.Columns.Add("Ticket Type")
flightDetails.Columns.Add("Flight ID")
flightDetails.Columns.Add("Flight Date")
flightDetails.Columns.Add("Flight Src")
flightDetails.Columns.Add("Flight Dest")
flightDetails.Rows.Add("Mark", "Sales", "Economy", 7855, New DateTime(2021, 10,
11).ToShortDateString(), "New York", "Tokyo")
flightDetails.Rows.Add("Sophie", "Services", "Economy", 7426, New DateTime(2021, 10,
12).ToShortDateString(), "London", "Venice")
flightDetails.Rows.Add("Oliver", "Finance", "Business", 7641, New DateTime(2021, 10,
15).ToShortDateString(), "New Delhi", "Moscow")
flightDetails.Rows.Add("James", "R&D", "Economy", 7293, New DateTime(2021, 10,
18).ToShortDateString(), "Beijing", "Dubai")
flightDetails.Rows.Add("Emma", "Marketing", "Business", 7117, New DateTime(2021, 10,
20).ToShortDateString(), "Paris", "Seoul")

ds.Tables.Add(flightDetails)

'Create table
TestActiveSheet.Cells("A1").Value = "For Finance Only:"
Dim table As GrapeCity.Spreadsheet.ITable =
TestActiveSheet.Range("A2:G7").CreateTable(True)

'Set auto generate columns
table.AutoGenerateColumns = True

'Bind table to datasource
table.DataSource = ds
```

Binding Tables Manually

You can bind a table to a data source and manually set its columns by using the **ITableColumn Interface (on-line documentation)**. The **AutoGenerateColumns (AutoGenerateColumns Property in the on-line documentation)** property must be set to false when manually binding a table.

The **DataField (DataField Property in the on-line documentation)** property helps to assign columns to a data field in the data source. You can also set the columns to a specific cell type by using the **CellType (CellType Property in the on-line documentation)** property.

Following up on the example from the previous section, the company can also choose to manually create a table using the existing data source. This table keeps a record of unavailability of employees on their flying date to keep their respective managers informed.

	A	B	C	D	E	F	G
1	For Finance Only:						
2	Passenger Name	Department	Ticket Type	Flight ID	Flight Date	Flight Src	Flight Dest
3	Mark	Sales	Economy	7855	11-10-2021	New York	Tokyo
4	Sophie	Services	Economy	7426	12-10-2021	London	Venice
5	Oliver	Finance	Business	7641	15-10-2021	New Delhi	Moscow
6	James	R&D	Economy	7293	18-10-2021	Beijing	Dubai
7	Emma	Marketing	Business	7117	20-10-2021	Paris	Seoul
8							
9	Unavailable Employees:						
10	Passenger Name	Department	Flight Dest	Flight Date			
11	Mark	Sales	Tokyo	11-10-2021			
12	Sophie	Services	Venice	12-10-2021			
13	Oliver	Finance	Moscow	15-10-2021			
14	James	R&D	Dubai	18-10-2021			
15	Emma	Marketing	Seoul	20-10-2021			

C#

```
// Create table
// GrapeCity.Spreadsheet.IWorksheet TestActiveSheet =
fpSpread1.AsWorkbook().ActiveSheet;
TestActiveSheet.Cells["A9"].Value = "Unavailable Employees:";
GrapeCity.Spreadsheet.ITable table2 =
TestActiveSheet.Range("A10:D12").CreateTable(true);

// Disable auto generate columns
table2.AutoGenerateColumns = false;

// Set data fields in columns
table2.TableColumns[0].DataField = "Passenger Name";
table2.TableColumns[1].DataField = "Department";
table2.TableColumns[2].DataField = "Flight Dest";
table2.TableColumns[3].DataField = "Flight Date";

// Bind table to the existing datasource 'ds' defined in the previous section
table2.DataSource = ds;
```

Visual Basic

```
'Create table
'Dim TestActiveSheet As GrapeCity.Spreadsheet.IWorksheet =
FpSpread1.AsWorkbook().ActiveSheet
TestActiveSheet.Cells("A9").Value = "Unavailable Employees:"
Dim table2 As GrapeCity.Spreadsheet.ITable =
TestActiveSheet.Range("A10:D12").CreateTable(True)

'Disable auto generate columns
table2.AutoGenerateColumns = False

'Set data fields in columns
table2.TableColumns(0).DataField = "Passenger Name"
table2.TableColumns(1).DataField = "Department"
table2.TableColumns(2).DataField = "Flight Dest"
```

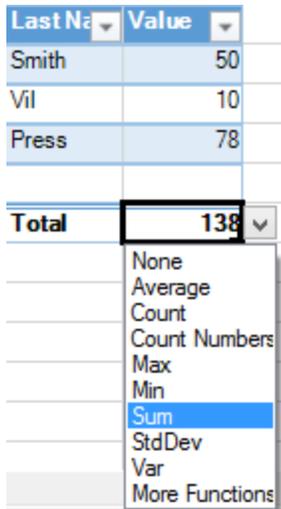
```
table2.TableColumns(3).DataField = "Flight Date"
```

```
'Bind table to the existing datasource 'ds' defined in the previous section
table2.DataSource = ds
```

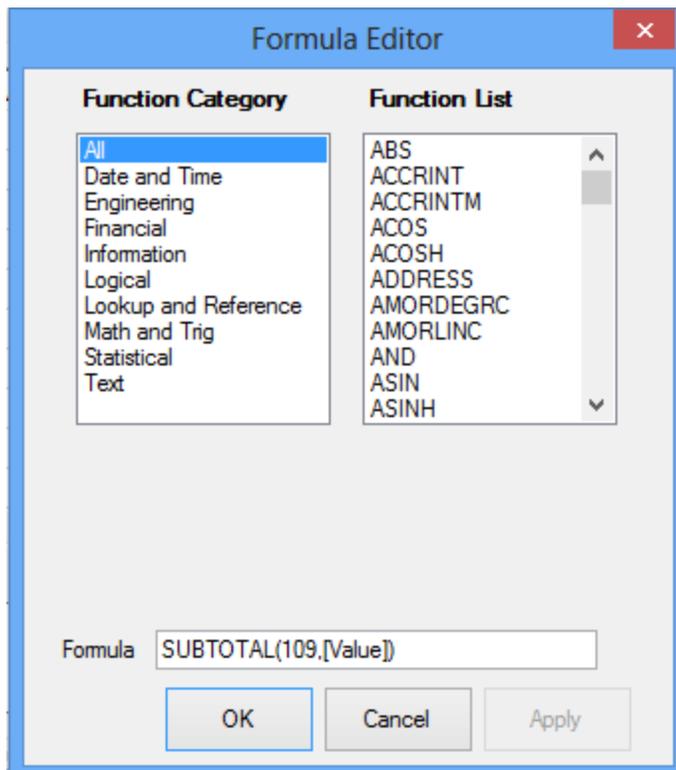
Adding a Table Formula

You can add formulas to the table in a total row at run time with the **Formula Editor** or with code.

Add a total row and then select the drop-down arrow at the bottom right corner of the table to display formulas.



You can select **More Functions** to display the **Formula Editor** as shown in the following figure.



You can add formulas to the table with the **Formula ('Formula Property' in the on-line documentation)** property. For more information about using structured references in table formulas, see **Using Structured References**.

Setting Visibility of Total Row

You can use the **TotalRowVisible ('TotalRowVisible Property' in the on-line documentation)** property to display the total row for the table. The following example adds a total row.

C#

```
fpSpread1.Sheets[0].Cells[1, 1].Text = "Last Name";
fpSpread1.Sheets[0].Cells[1, 2].Text = "Value";
fpSpread1.Sheets[0].Cells[2, 1].Text = "Smith";
fpSpread1.Sheets[0].Cells[2, 2].Value = 50;
fpSpread1.Sheets[0].Cells[3, 1].Text = "Vil";
fpSpread1.Sheets[0].Cells[3, 2].Value = 10;
fpSpread1.Sheets[0].Cells[4, 1].Text = "Press";
fpSpread1.Sheets[0].Cells[4, 2].Value = 78;
FarPoint.Win.Spread.TableView table = fpSpread1.Sheets[0].AddTable("table", 1, 1, 5, 2);
table.TotalRowVisible = true;
```

VB

```
fpSpread1.Sheets(0).Cells(1, 1).Text = "Last Name"
fpSpread1.Sheets(0).Cells(1, 2).Text = "Value"
fpSpread1.Sheets(0).Cells(2, 1).Text = "Smith"
fpSpread1.Sheets(0).Cells(2, 2).Value = 50
fpSpread1.Sheets(0).Cells(3, 1).Text = "Vil"
fpSpread1.Sheets(0).Cells(3, 2).Value = 10
fpSpread1.Sheets(0).Cells(4, 1).Text = "Press"
fpSpread1.Sheets(0).Cells(4, 2).Value = 78
Dim table As FarPoint.Win.Spread.TableView = fpSpread1.Sheets(0).AddTable("table", 1, 1, 5, 2)
table.TotalRowVisible = True
```

Automatically Create Calculated Columns

You can automatically fill a column with the entered formula in the table by using the [AutoCreateCalculatedColumns](#) property. This property accepts a boolean value.

A	B	C	D	E
	Last Na	Value	Sum	
	Smith	50		
	Vil	10		
	Press	78		
	Dias	35		

C#

```
fpSpread1.Sheets[0].Cells[1, 3].Text = "Sum";

fpSpread1.Features.AutoCreateCalculatedTableColumns = true;
```

Visual Basic

```
FpSpread1.Sheets(0).Cells(1, 3).Text = "Sum"
```

```
FpSpread1.Features.AutoCreateCalculatedTableColumns = True
```

Understanding Structured References

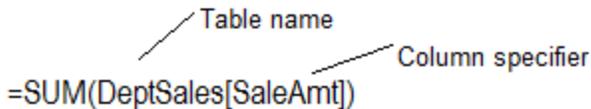
Spread supports structured reference formulas in tables. Components of the structured reference include the table name, the column specifier, the special item specifier, and the table specifier.

A table name is the name assigned to the table. The name references the table data, but not the header and totals rows, if any.

A column specifier is derived from the column header and references the column data (excluding the column header and total, if any). A special item specifier is a way to refer to specific portions of the table, such as the Totals row.

The table specifier is the outer portion of the structured reference. The specifiers follow the table name, and are enclosed in square brackets. A structured reference is the entire string beginning with the table name and ending with the column specifier.

Specifiers are enclosed in brackets.



The following topics provide additional information about structured references in tables.

- [Using Operators and Special Items](#)
- [Understanding Structured Reference Syntax Rules](#)
- [Using Structured References](#)

Using Operators and Special Items

You can use operators and special items in the structured reference. The structured reference can be unqualified or fully qualified.

For added flexibility in specifying ranges of cells, you can use the following reference operators to combine column specifiers. The **Cell Range** column is a general example.

Structured Reference	Refers To	Operator	Cell Range
<code>=DeptSales[[SalesPerson]:[Region]]</code>	All of the cells in two or more adjacent columns	: (colon) range operator	A2:B7
<code>=DeptSales[SalesAmt],DeptSales[ComAmt]</code>	A combination	, (comma) union	C2:C7, E2:E7

	of two or more columns	operator	
=DeptSales[[SalesPerson]:[SaleAmt]] DeptSales[[Region]:[ComPct]]	The intersection of two or more columns	(space) intersection operator	B2:C7

For added convenience, you can also use special items to refer to various portions of a table, such as the Totals row, to make it easier to refer to these portions in formulas. The following are the special item specifiers that you can use in a structured reference:

Special Item Specifier	Refers To	Cell Range
=DeptSales[#All]	The entire table, including column headers, data, and totals (if any)	A1:E8
=DeptSales[#Data]	Just the data	A2:E7
=DeptSales[#Headers]	Just the header row	A1:E1
=DeptSales[#Totals]	Just the total row. If none exists, then it returns null	A8:E8
=DeptSales[#This Row]	Just the portion of the columns in the current row. #ThisRow cannot be combined with any other special item specifiers. Use it to force implicit intersection behavior for the reference or to override implicit intersection behavior and refer to single values from a column.	

When you create a calculated column, you often use a structured reference to create the formula. This structured reference can be unqualified or fully qualified. For example, to create the calculated column called, ComAmt, that calculates the amount of commission in dollars, you can use the following formulas:

Structured Reference	Example	Comment
Unqualified	=[SaleAmt]*[ComPct]	Multiplies the corresponding values from the current row
Fully qualified	=DeptSales[SaleAmt]*DeptSales[ComPct]	Multiplies the corresponding values for each row for both columns

If you are using structured references within a table, such as when you create a calculated column, you can use an unqualified structured reference, but if you use the structured reference outside of the table, you need to use a fully qualified structured reference.

Understanding Structured Reference Syntax Rules

Structured references have additional syntax rules listed as follows:

- Matching brackets are required for tables, specifiers, and special characters.
- Characters with special meaning require an escape character.
- Spaces can be used in certain areas to make the structured reference easier to read.

All table, column, and special item specifiers must be enclosed in matching brackets ([]). A specifier that contains other specifiers requires outer matching brackets to enclose the inner matching brackets of the other specifiers, for example:

=DeptSales[[SalesPerson]:[Region]]

All column headers are text strings, but do not require quotes when they are used in a structured reference. If a column header contains numbers or dates, such as 2004 or 1/1/2004, these are still considered text strings. Because column headers are text strings, you cannot use expressions within brackets, for example:

```
=DeptSalesFYSummary[[2004]:[2002]]
```

If a table column header contains one of the following special characters, the entire column header must be enclosed in brackets. This means double brackets are required in a column specifier with the following special characters: space, tab, line feed, carriage return, comma (,), colon (:), period (.), left bracket ([), right bracket (]), pound sign (#), single quotation mark ('), double quotation mark ("), left brace ({), right brace (}), dollar sign (\$), caret (^), ampersand (&), asterisk (*), plus sign (+), equal sign (=), minus sign (-), greater than symbol (>), less than symbol (<), and division sign (/).

The following structured reference includes a column specifier that contains special characters:

```
=DeptSalesFYSummary[[Total$Amount]]
```

The only exception to this is if the only special character that is used is a space character, for example:

```
=DeptSales[Total Amount]
```

The following characters have special meaning and require the use of a single quotation mark (') as an escape character: left bracket ([), right bracket (]), pound sign(#), and single quotation mark (').

The following example illustrates a structured reference that contains a character with a special meaning:

```
=DeptSalesFYSummary['#OfItems]
```

You can use space characters to improve the readability of a structured reference. You can use one space after the first left bracket ([) and preceding the last right bracket (]). You can also use one space after a comma, as shown in the following examples:

```
=DeptSales[ [SalesPerson]:[Region] ]
```

```
=DeptSales[[#Headers], [#Data], [ComPct]]
```

Using Structured References

You can add structured references to tables using the **Formula ('Formula Property' in the on-line documentation)** property.

A cell outside of the table can have a formula with a table reference; however, the table name must be unique among table names and custom names. The table name must also be valid.

Using Code

Set the **Formula ('Formula Property' in the on-line documentation)** property for the cell.

Example

This example code sums the Value column in the table.

C#

```
fpSpread1.Sheets[0].Cells[1, 1].Text = "Last Name";
fpSpread1.Sheets[0].Cells[1, 2].Text = "Value";
fpSpread1.Sheets[0].Cells[2, 1].Text = "Smith";
fpSpread1.Sheets[0].Cells[2, 2].Value = 50;
fpSpread1.Sheets[0].Cells[3, 1].Text = "Vil";
fpSpread1.Sheets[0].Cells[3, 2].Value = 10;
fpSpread1.Sheets[0].Cells[4, 1].Text = "Press";
fpSpread1.Sheets[0].Cells[4, 2].Value = 78;
```

```
fpSpread1.Sheets[0].AddTable("table", 1, 1, 5, 2);  
fpSpread1.Sheets[0].Cells[5, 1].Formula = "SUM(table[Value])";
```

VB

```
fpSpread1.Sheets(0).Cells(1, 1).Text = "Last Name"  
fpSpread1.Sheets(0).Cells(1, 2).Text = "Value"  
fpSpread1.Sheets(0).Cells(2, 1).Text = "Smith"  
fpSpread1.Sheets(0).Cells(2, 2).Value = 50  
fpSpread1.Sheets(0).Cells(3, 1).Text = "Vil"  
fpSpread1.Sheets(0).Cells(3, 2).Value = 10  
fpSpread1.Sheets(0).Cells(4, 1).Text = "Press"  
fpSpread1.Sheets(0).Cells(4, 2).Value = 78  
fpSpread1.Sheets(0).AddTable("table", 1, 1, 5, 2)  
fpSpread1.Sheets(0).Cells(5, 1).Formula = "SUM(table[Value])"
```

Understanding the Underlying Models

The Spread component is based on a set of underlying models: classes that provide most of the features for the component. You can work directly with these models, or you can work with the Spread Designer or shortcut objects. When you perform tasks using the Spread Designer or shortcut objects, the tasks actually affect the models themselves.

If you want to provide extensive customizations for the component, increase efficiency, or create a template to use within your workgroup, you will probably want to customize the Spread models.

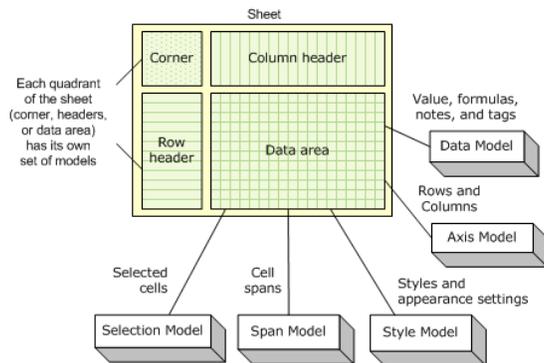
Consult the following topics for lists of the Spread models and more information about the model classes.

These topics can help you customize the component using models:

- **Understanding the Sheet Model Classes and Interfaces**
- **Finding More Details on the Sheet Models**
- **Creating a Custom Sheet Model**
- **Understanding the Optional Interfaces**

Finding More Details on the Sheet Models

The Spread component models are illustrated conceptually in the following diagram:



As shown in the figure, the data area of the spreadsheet has its own set of models, and the row headers and column headers have models assigned to each of them. Finally, the sheet corner has its own set of models.

As the diagram illustrates, it can be useful to think of the sheet (SheetView object) as a composite of the five underlying models.

- **Axis:** The Axis model handles everything to do with the Columns and Rows (for example, the column width, row height, and whether a row or column is visible).
- **Data:** The Data model handles everything to do with the data (for example, the value, the formula, and any optional notes or tags in a cell) and contains the data in the sheet.
- **Selection:** The Selection model handles any cell range selections that are made.
- **Span:** The Span model handles any spanned cells.
- **Style:** The Style model handles the appearance settings for the cells (for example, the background color, the font, and the cell type).

You can do many tasks without ever using the models by using the Spread Designer or properties of the shortcut objects (such as Cells, Columns, and Rows). Since sheet models are the basis for all the shortcut objects, using models is generally faster than using shortcut objects. For example, code using the shortcut object to set a value:

```
FpSpread1.Sheets(0).Cells(0,0).Value = "Test"
```

would be equivalent to using the underlying data model method:

```
FpSpread1.Sheets(0).DataModel.SetValue(FpSpread1.Sheets(0).GetModelRowFromViewRow(0), FpSpread1.Sheets(0).GetModelColumnFromViewColumn(0), "Test")
```

The sheet models correspond to the basis of all the objects and settings of a particular sheet. Each sheet has its own set of models. If you have multiple sheets in your Spread component, then each sheet has its own set of models.

There are many interfaces involved in the models. Each model class implements a number of interfaces, and each model has one "model" interface which must be implemented to make it a valid implementation for that particular model. All references to the model classes are through the interfaces, and no assumptions are made as to what interfaces are implemented on each model (except for the "model" interface which must be present). If the model class does not implement a particular interface, then that functionality is simply disabled in the sheet (that is, if `IDataSourceSupport` is not implemented by `SheetView.Models.Data`, then the `DataSource` and `DataMember` properties are not functional). For complete lists of these interfaces, you can look up the overview for the default model classes in the **Assembly Reference (on-line documentation)**.

Understanding the Sheet Model Classes and Interfaces

The following table lists the models and their associated classes and interfaces.

Sheet Model	Classes and Interface	Description
-------------	-----------------------	-------------

Axis model	<p>BaseSheetAxisModel ('BaseSheetAxisModel Class' in the on-line documentation)</p> <p>DefaultSheetAxisModel ('DefaultSheetAxisModel Class' in the on-line documentation)</p> <p>ISheetAxisModel ('ISheetAxisModel Interface' in the on-line documentation)</p>	Basis for how the sheet of cells is structured in terms of rows and columns. For more information, see Understanding the Axis Model .
Data model	<p>BaseSheetDataModel ('BaseSheetDataModel Class' in the on-line documentation)</p> <p>DefaultSheetDataModel ('DefaultSheetDataModel Class' in the on-line documentation)</p> <p>ISheetDataModel ('ISheetDataModel Interface' in the on-line documentation)</p>	Basis for the manipulation of data in the cells in the sheet. For more information, see Understanding the Data Model .
Selection model	<p>BaseSheetSelectionModel ('BaseSheetSelectionModel Class' in the on-line documentation)</p> <p>DefaultSheetSelectionModel ('DefaultSheetSelectionModel Class' in the on-line documentation)</p> <p>ISheetSelectionModel ('ISheetSelectionModel Interface' in the on-line documentation)</p>	Basis for the behavior of and interaction of selected cells in the sheet. For more information, see Understanding the Selection Model .
Span model	<p>BaseSheetSpanModel ('BaseSheetSpanModel Class' in the on-line documentation)</p> <p>DefaultSheetSpanModel ('DefaultSheetSpanModel Class' in the on-line documentation)</p> <p>ISheetSpanModel ('ISheetSpanModel Interface' in the on-line documentation)</p>	Basis for how cells in the sheet are spanned. For more information, see Understanding the Span Model .
Style model	<p>BaseSheetStyleModel ('BaseSheetStyleModel Class' in the on-line documentation)</p> <p>DefaultSheetStyleModel ('DefaultSheetStyleModel Class' in the on-line documentation)</p> <p>ISheetStyleModel ('ISheetStyleModel Interface' in the on-line documentation)</p>	Basis for the appearance of the cells in the sheet. For more information, see Understanding the Style Model .

Everything you do to the model is automatically updated in the sheet and most of the aspects of the sheet that you can modify are updated in the model. This is also true for Cell, Row, and Column object settings, too. Most of the aspects changed with these objects automatically changes the setting in the corresponding sheet model and vice versa. If you add columns to the data model, then they are added to the sheet. This is true, even down to the parameters; for example the row and column arguments in the **GetValue** and **SetValue** methods for the data model are the same indexes as that of the rows and columns in the sheet as long as the sheet is not sorted.

Not everything in the Spread namespace is in the models. For example, there are aspects of the overall component, for example, the sheet tabs, the sheet background color, and the grid lines, that are not in the models. But the relevant pieces of information about a given cell, both about the data in the cells and about the appearance of the cells, are in the models.

The data area of the spreadsheet has its own set of models, and the row headers and column headers are considered two more such groups having models assigned to each of them, and the sheet corner is another with its own set of models.

Each model has a base model class, a default model class, and an interface.

- The base model is the base on which the default model is created and is for creating custom models from scratch.
- The default model is the model with which you most likely will develop; this provides the default features that the component offers and is used for small customizations to the models.

The base model has the fewest built-in features, and the default model extends the base model. If you want to provide different features or customize the behavior or appearance of your application, you can extend the base models to create new classes. For example you may do this to create a template component for all the developers in your organization. By creating your own class based on one of the base models, you can create the customized class and provide it to all the developers to use. Typically, if you are editing the models, use the default model classes. But if you want to create a custom model (from scratch), use the base model classes.

Each default model class contains the implementation of the interface for that model type as well as additional optional interfaces. Most of the functionality (that is, formulas, data binding, XML serialization, etc.) is optional in the model class, and is implemented in separate interfaces from the main model interfaces (such as `ISheetDataModel`). Therefore, if you want to implement your own model class, you can pick and choose which pieces of functionality you have in your model.

It is important for the models to stay in sync with each other, so that the row count and column count is consistent among the models making up the sheet. The `SheetView` object listens for the `ISheetDataModel.Change` event from the `SheetView.DocumentModels.Data` property, and updates the other models accordingly when the row count or column count changes due to any of these:

- direct property settings for the `RowCount` or `ColumnCount` properties,
- insert/delete row/column operations through the `IRangeSupport` interface
- the entire data model being replaced with a new one

If the models get out of sync, then index out-of-range exceptions can be caused by code trying to get information about nonexistent rows or columns.

For more information on creating a custom model for a sheet, refer to **Creating a Custom Sheet Model**.

Understanding the Data Model

The data model includes the contents of the cells, including the value or the formula in a cell, and the cell notes or cell tags. This includes the `Value` properties for cells in the data area of the spreadsheet, the database properties for data-bound spreadsheets, and anything having to do with the contents in the cells.

You are likely to customize the data model when working with Spread. The data model implements more interfaces, and more optional functionality through it, than any of the other models. Also, if you want to implement the equivalent to the unbound virtual model feature of the ActiveX Spread control, for example, you will need to customize the data model.

The following topics provide more information about the data model:

- Data Model Object
- Setting and Adding to the Data Model
- Implemented Interfaces

For more details, refer to the **BaseSheetDataModel** ('**BaseSheetDataModel Class**' in the **on-line documentation**) class, the **DefaultSheetDataModel** ('**DefaultSheetDataModel Class**' in the **on-line documentation**) class and the **ISheetDataModel** ('**ISheetDataModel Interface**' in the **on-line documentation**) interface.

Data Model Object

The data model is an object that supplies the cell values being displayed in the sheet. In most cases, you can simply use the default data model of Spread that is created when the sheet is created.

The default data model of Spread creates objects to store notes, formulas, tags, and values, and those objects are designed to balance memory usage versus speed based on how big the model is and how sparse the data in the model is. If you are not using notes, formulas, and tags, then the component does not use much memory because the data is fairly sparse. In fact, those objects do not allocate any memory for data until it is actually needed; therefore, as long as there are no notes, formulas, or tags set in the model, memory usage remains low.

The default data model can be used in unbound mode or bound mode. In unbound mode, the data model acts similarly to a two-dimensional array of cell values. In bound mode, the data model wraps the supplied data source and if needed can supply additional settings not available from the data source, for example, cell formulas and unbound rows or columns.

Setting and Adding to the Data Model

The **SetModelDataColumn** ('**SetModelDataColumn Method**' in the **on-line documentation**) is different from **AddColumn** ('**AddColumn Method**' in the **on-line documentation**) in that you can specify which data field you want bound to which column in the data model.

If you add columns to the model, then they are added to the sheet. The row and column in the **GetValue** ('**GetValue Method**' in the **on-line documentation**) and **SetValue** ('**SetValue Method**' in the **on-line documentation**) methods of the data model have the same indexes as that of the columns in the sheet as long as the sheet is not sorted. If the sheet's rows or columns are sorted, then the view coordinates must be mapped to the model coordinates with these **SheetView.GetModelRowFromViewRow** ('**GetModelRowFromViewRow Method**' in the **on-line documentation**) and **SheetView.GetModelColumnFromViewColumn** ('**GetModelColumnFromViewColumn Method**' in the **on-line documentation**) methods.

The **SheetView.GetValue** ('**GetValue Method**' in the **on-line documentation**) and **SheetView.SetValue** ('**SetValue Method**' in the **on-line documentation**) methods always get and set the data in the data model. Calling these methods is the same as calling **SheetView.Models.Data.GetValue** ('**GetValue Method**' in the **on-line documentation**) and **SheetView.Models.Data.SetValue** ('**SetValue Method**' in the **on-line documentation**). The **Cell.Value** ('**Value Property**' in the **on-line documentation**) property returns the value of the cell in the editor control if the cell is currently in edit mode in a **SpreadView** containing the **SheetView**. That value is not updated to the data model until the cell leaves edit mode; however, you can manually update the value in the data model using code:

C#

```
SheetView.SetValue(row, column, SheetView.Cells(row, column).Value);
```

Implemented Interfaces

When the data model implements **IDataSourceSupport** ('**IDataSourceSupport Interface**' in the **on-line documentation**) and it is bound to a data source, the bound parts of the data model get and set data directly from the data source. Some columns in a bound data model can be unbound if columns are added to the data model with

AddColumns ('**AddColumns Method**' in the on-line documentation) after it is bound (**IDataSourceSupport.IsColumnBound** ('**IsColumnBound Method**' in the on-line documentation) returns **False** for those model column indexes), and the values in those unbound columns are stored in the data model rather than the data source.

If the data model also implements **IUnboundRowSupport** ('**IUnboundRowSupport Interface**' in the on-line documentation), then some rows in the data model can also be unbound, and those values are also stored in the data model rather than the data source. Such rows can be made into bound rows by calling **IUnboundRowSupport** ('**IUnboundRowSupport Interface**' in the on-line documentation).**AddRowToDataSource** ('**AddRowToDataSource Method**' in the on-line documentation), and if the **autoFill** parameter is specified as **True**, then the data in the bound columns in that unbound row will be added to the data source in a new record or element, assuming that the data source permits it (you will get an exception if it does not), and the unbound row becomes a bound row.

The default data model class, **DefaultSheetDataModel** ('**DefaultSheetDataModel Class**' in the on-line documentation), implements all of these interfaces, plus many others related to calculation, hierarchy, and serialization.

Understanding the Axis Model

The axis model includes the methods that manage row- and column-related settings of the spreadsheet (how the rows and columns of cells are oriented on the sheet). The axis model includes many of the axis-related settings in the following shortcut objects:

- **Column** ('**Column Class**' in the on-line documentation) and **Columns** ('**Columns Class**' in the on-line documentation)
- **Row** ('**Row Class**' in the on-line documentation) and **Rows** ('**Rows Class**' in the on-line documentation)
- **AlternatingRow** ('**AlternatingRow Class**' in the on-line documentation) and **AlternatingRows** ('**AlternatingRows Class**' in the on-line documentation)

These settings include:

- row height
- column width
- row visible
- column visible

To use the underlying axis model, use the methods of the axis model. These include the **SetSize** ('**SetSize Method**' in the on-line documentation) method, for setting the row height or column width, and the **SetVisible** ('**SetVisible Method**' in the on-line documentation) method for setting the row or column visible properties. There are other methods, too, such as **SetMergePolicy** ('**SetMergePolicy Method**' in the on-line documentation), which set specific properties of the row or column, in this case whether cells can be automatically merged when their content is identical. Refer to the **DefaultSheetAxisModel** ('**DefaultSheetAxisModel Class**' in the on-line documentation) class for more information on the axis model in general and the methods in particular.

As an example of how you could use the axis model to improve performance of a spreadsheet, consider a spreadsheet with a very large number of rows. If you are resizing the rows based on the data, then you might want to create a custom axis model for **SheetView.Models.RowAxis** ('**RowAxis Property**' in the on-line documentation) to return this value. To do so, create a class derived from **DefaultSheetAxisModel** ('**DefaultSheetAxisModel Class**' in the on-line documentation) that takes a reference to the **SheetView** in its constructor and stores it in a field. Then override the **GetSize** ('**GetSize Method**' in the on-line documentation) method to call **GetPreferredRowHeight** ('**GetPreferredRowHeight Method**' in the on-line documentation) (in the **SheetView**) for the row index. You can also override the **GetResizable** ('**GetResizable Method**' in the on-line documentation) method to prevent the user from trying to change the row heights manually, which will not work since **GetSize** ('**GetSize Method**' in the on-line documentation) is always returning the preferred height.

Example

The following code provides an example that makes each row three times wider than the default width.

C#

```
public class MyRowAxisModel : FarPoint.Web.Spread.Model.DefaultSheetAxisModel
{
    public overrides int GetSize(int index)
    {
        if ( index % 2 == 1 )
            return 60;
        else
            return 20; }
}
```

VB

```
Public Class MyRowAxisModel
    Inherits FarPoint.Web.Spread.Model.DefaultSheetAxisModel
    Public Overrides Function GetSize(index As Integer) As Integer
        If index \ 2 = 1 Then
            Return 60
        Else
            Return 20
        End If
    End Function
End Class
```

For more details, refer to the **BaseSheetAxisModel** ('[BaseSheetAxisModel Class](#)' in the on-line documentation) class, the **DefaultSheetAxisModel** ('[DefaultSheetAxisModel Class](#)' in the on-line documentation) class, and the **ISheetAxisModel** ('[ISheetAxisModel Interface](#)' in the on-line documentation) interface.

Understanding the Selection Model

The selection model includes any of the settings related to ranges of selected cells. The selection model includes methods such as counting the number of selected ranges, adding and removing selections, clearing selections, and finding whether a cell is selected.

To use the underlying selection model, use the methods of the selection model. These include the **SetSelection** ('[SetSelection Method](#)' in the on-line documentation) method, for setting cells as selected, and the **AddSelection** ('[AddSelection Method](#)' in the on-line documentation), **ClearSelection** ('[ClearSelection Method](#)' in the on-line documentation), and **RemoveSelection** ('[RemoveSelection Method](#)' in the on-line documentation) methods for adding, clearing, and removing selected ranges from the sheet. Refer to the **DefaultSheetSelectionModel** ('[DefaultSheetSelectionModel Class](#)' in the on-line documentation) class for more information on the selection model in general and the methods in particular.

The default implementation of the selection model (**DefaultSheetSelectionModel** ('[DefaultSheetSelectionModel Class](#)' in the on-line documentation)) handles the selection of cells and ranges in the sheet and stores the actual cell and range coordinates for each selection. The **SpreadView** object handles the user interface for the **SheetView** object and updates the selection model from various event handlers.

The selection model handles the selection data, including computing the range being selected based on the cell clicked (the anchor cell) and the cell under the mouse pointer. The **SheetView**.**GetSelection** ('[GetSelection Method](#)' in the on-line documentation) method and **SheetView**.**SelectionCount** ('[SelectionCount Property](#)' in the on-line documentation) property wrap the **ISheetSelectionModel** ('[ISheetSelectionModel Interface](#)' in the on-line documentation) indexer and **Count** ('[Count Property](#)' in the on-line documentation) property. When there is no selection in the model, the count is 0 and **GetSelection** ('[GetSelection Method](#)' in the on-line

documentation) returns null.

Some events cause the anchor cell in the selection model to be set (for example, left mouse button down on a cell) so the selection model has the active cell as a selection. If you enter edit mode then cancel it by pressing the Escape key (Esc), or if you use the keyboard to move the active cell around instead of the mouse, then the selection model might be cleared. You should check the **SelectionCount** ('**SelectionCount Property**' in the on-line documentation) before using **GetSelection** ('**GetSelection Method**' in the on-line documentation), and in the case where it returns 0, use the **ActiveColumnIndex** ('**ActiveColumnIndex Property**' in the on-line documentation) and **ActiveRowIndex** ('**ActiveRowIndex Property**' in the on-line documentation) properties.

The selection model is saved to the view state only if it contains at least one selection.

For more details, refer to the **BaseSheetSelectionModel** ('**BaseSheetSelectionModel Class**' in the on-line documentation) class, the **DefaultSheetSelectionModel** ('**DefaultSheetSelectionModel Class**' in the on-line documentation) class, and the **ISheetSelectionModel** ('**ISheetSelectionModel Interface**' in the on-line documentation) interface.

For more information on working with selections programmatically, refer to **Working with Selections**.

Understanding the Span Model

The span model includes the objects needed to handle cell spans and automatic merging of cells. Refer to the **Cell** ('**Cell Class**' in the on-line documentation) class, **ColumnSpan** ('**ColumnSpan Property**' in the on-line documentation) and **RowSpan** ('**RowSpan Property**' in the on-line documentation) properties.

To use the underlying span model, use the **Add** ('**Add Method**' in the on-line documentation), **Clear** ('**Clear Method**' in the on-line documentation), and **Remove** ('**Remove Method**' in the on-line documentation) methods of the span model. Refer to the **DefaultSheetSpanModel** ('**DefaultSheetSpanModel Class**' in the on-line documentation) class for more information on the span model in general and the methods in particular.

The default implementation of the span model (**DefaultSheetSpanModel** ('**DefaultSheetSpanModel Class**' in the on-line documentation)) uses an array to stored the cell spans. If there are a moderate number of spans in the sheet (for example, a thousand or less) then the default implementation works fine. If there are a very large number of spans in the sheet (for example, a hundred thousand or more) then the default implementation slows dramatically. In scenarios where you have a very large number of spans that repeat on a regular interval, you should consider writing a custom span model. By writing a custom span model, you can significantly increase the speed and decrease the memory usage.

For more details, refer to the **BaseSheetSpanModel** ('**BaseSheetSpanModel Class**' in the on-line documentation) class, the **DefaultSheetSpanModel** ('**DefaultSheetSpanModel Class**' in the on-line documentation) class, and the **ISheetSpanModel** ('**ISheetSpanModel Interface**' in the on-line documentation) interface.

Understanding the Style Model

The style model includes appearance settings which might be:

- Set in the Spread Designer
- Set as properties in the Properties List
- Inherited from a custom skin for a whole sheet or from a custom style for individual cells

For more information on appearance settings for a sheet, refer to **Creating a Custom Skin for a Sheet and Applying a Skin to a Sheet**. For more information on appearance settings for a cell, refer to **Creating and Applying a Style for Cells**.

More information about the style model is provided in the following topics:

- Settings and Objects for Style
- Order of Inheritance of Styles

- Composited or Inherited Styles
- Style Name
- Format Objects

The style model includes the cell types as well. The various cell types determine the appearance of a cell in several ways. For more information about the various cell types, refer to **Cell Types**.

Refer to the **DefaultSheetStyleModel ('DefaultSheetStyleModel Class' in the on-line documentation)** class for more information on the style model in general and the methods in particular.

For more details, refer to the **BaseSheetStyleModel ('BaseSheetStyleModel Class' in the on-line documentation)** class, the **DefaultSheetStyleModel ('DefaultSheetStyleModel Class' in the on-line documentation)** class, and the **ISheetStyleModel ('ISheetStyleModel Interface' in the on-line documentation)** interface.

Settings and Objects for Style

The appearance settings may be set from any of the following classes in the FarPoint Spread namespace that represent shortcut objects:

- **Cell ('Cell Class' in the on-line documentation)**
- **Column ('Column Class' in the on-line documentation)**
- **Row ('Row Class' in the on-line documentation)**
- **AlternatingRow ('AlternatingRow Class' in the on-line documentation)**

They can also be set from any of these classes in the FarPoint Spread namespace that affect style:

- **Appearance ('Appearance Class' in the on-line documentation)**
- **DefaultSkins ('DefaultSkins Class' in the on-line documentation)**
- **NamedStyle ('NamedStyle Class' in the on-line documentation)**
- **SheetSkin ('SheetSkin Class' in the on-line documentation)**

Properties that correspond to **StyleInfo ('StyleInfo Class' in the on-line documentation)** properties are stored in the style model through the **ISheetStyleModel ('ISheetStyleModel Interface' in the on-line documentation)** interface. Style properties can be set for a cell, row (column index -1), column (row index -1), or the entire model (column and row index -1). Properties that are not set in a cell are inherited from the row setting, or the column setting if the row has no setting, or the model default if the column also has no setting.

The default is exposed through the **DefaultStyle** property (**SheetView ('SheetView Class' in the on-line documentation).DefaultStyle**, **ColumnHeader ('ColumnHeader Class' in the on-line documentation).DefaultStyle**, and **RowHeader ('RowHeader Class' in the on-line documentation).DefaultStyle**). If you set or get a style property using **Rows.Default** or **Rows[-1]** or **Columns.Default** or **Columns[-1]**, then you will actually be setting or getting the **DefaultStyle** property. This is because **Column** and **Row** always use row index -1 and column index -1 when accessing the style model, respectively, and so using a column index or a row index of -1 will be setting or getting the model default.

Order of Inheritance of Styles

The order of inheritance is described in **Object Parentage**. Here is a list of the style properties that are included in the style model, which are basically the members of the **StyleInfo ('StyleInfo Class' in the on-line documentation)** class and affect the appearance or style of a cell:

- **Border ('Border Property' in the on-line documentation)**
- **CellType ('CellType Property' in the on-line documentation)**
- **Editor ('Editor Property' in the on-line documentation)**
- **Formatter ('Formatter Property' in the on-line documentation)**
- **HorizontalAlignment ('HorizontalAlignment Property' in the on-line documentation)**
- **Renderer ('Renderer Property' in the on-line documentation)**

- **VerticalAlignment ('VerticalAlignment Property' in the on-line documentation)**

Composited or Inherited Styles

The style properties for a cell can be composited or merged from the **Cell ('Cell Class' in the on-line documentation)**, **Row ('Row Class' in the on-line documentation)**, **Column ('Column Class' in the on-line documentation)**, **SheetView ('SheetView Class' in the on-line documentation)**, and parent **NamedStyle ('NamedStyle Class' in the on-line documentation)** objects.

To use the underlying style model, use the methods of the style model for that sheet, specifically the **GetDirectInfo ('GetDirectInfo Method' in the on-line documentation)** method and **SetDirectInfo ('SetDirectInfo Method' in the on-line documentation)** method, and the settings in the **StyleInfo ('StyleInfo Class' in the on-line documentation)** object. "Direct" in the style model means "not composite" or "not inherited." **SetDirectInfo ('SetDirectInfo Method' in the on-line documentation)** sets the style properties that have been set for the specified cell, column, or row directly and does not return any settings that are set for higher levels (such as the entire model), while **GetCompositeInfo ('GetCompositeInfo Method' in the on-line documentation)** gives the style properties "composed" or "merged" into one **StyleInfo ('StyleInfo Class' in the on-line documentation)** object that contains all the settings that are used to paint and edit the cell, column, or row, including any inherited settings.

Style Name

Setting **StyleName** replaces the style in the style model with the **NamedStyle ('NamedStyle Class' in the on-line documentation)** having the specified name. Replacing the style with the **NamedStyle** changes the settings of all of the style-related properties, including **ParentStyleName** (which wraps **StyleInfo.Parent ('Parent Property' in the on-line documentation)**). Any previous setting for style-related properties like **BackColor**, **Font**, **Border**, or **ParentStyleName** are overwritten when you set **StyleName**. All of these properties are already set in the **NamedStyle ('NamedStyle Class' in the on-line documentation)** object; the component's properties are not changed just because you assigned the **NamedStyle** to a cell, column, row or alternating row. The named style is expected to already be set up the way you want it to be. If this includes the named style having a parent **NamedStyle**, then you should have that parent set already when you assign it with **StyleName**, or you should assign it separately using a reference to the **NamedStyle** object (this has the same effect as setting **ParentStyleName** after setting **StyleName**).

Keep in mind that after you have set **StyleName**, all cells where you have used that name are sharing the same **NamedStyle ('NamedStyle Class' in the on-line documentation)** object, and any changes that you make to one of those cells will also change all of the other cells sharing the same named style.

The following code creates two **NamedStyle ('NamedStyle Class' in the on-line documentation)** objects with a parent-child relationship, then sets some properties on the styles and adds them to the **NamedStyleCollection** in the **Spread** component.

C#

```
NamedStyle test_parent = new NamedStyle("test_parent");
test_parent.BackColor = Color.Red;
test.ForeColor = Color.White;
fpSpread1.NamedStyles.AddRange(new NamedStyle[] {test_parent, test});
fpSpread1.Sheets(0).Columns(0).BackColor = Color.Blue;
fpSpread1.Sheets(0).Rows(0).CellType = new NumberCellType();
fpSpread1.Sheets(0).Cells(0,0).StyleName = "test";
fpSpread1.Sheets(0).Cells(1,0).StyleName = "test";
```

VB

```
Dim test_parent As NamedStyle = New NamedStyle("test_parent")
test_parent.BackColor = Color.Red
test.ForeColor = Color.White
fpSpread1.NamedStyles.AddRange(New NamedStyle() {test_parent, test})
fpSpread1.Sheets[0].Columns[0].BackColor = Color.Blue
```

```
fpSpread1.Sheets[0].Rows[0].CellType = New NumberCellType()  
fpSpread1.Sheets[0].Cells[0,0].StyleName = "test"  
fpSpread1.Sheets[0].Cells[1,0].StyleName = "test"
```

In the example, the background color for the first column is set to blue and the cell type for the first row is set to Number, and the first cells in the first two rows are both set to use the NamedStyle named "test." The result is that the cells in the first column are blue, except for the cells in the first two rows, which are red because the "test" style inherits the red background color from its parent NamedStyle. The parent style overrides the inherited setting for the column.

The cell type for the first cell is Number, since there is cell type set in either NamedStyle object. There is a cell type set in the first row which is inherited by all cells in the row. The cell type for the second cell is General since there is no cell type setting for the cell, row, or column. The default cell type for the sheet is General. For more information on inheritance of style settings, refer to **Object Parentage**.

Format Objects

The FormatInfo strings in a saved XML file are **DateTimeFormatInfo** or **NumberFormatInfo** objects that store the format of the data. These are created in the style model when a General cell is edited, if the style model implements **IParseFormatSupport ('IParseFormatSupport Interface' in the on-line documentation)**. These format objects allow the cells to display the data in the same format that was used to enter it.

The General cell type parses the string into a number or DateTime, and generates the IFormatProvider and format string necessary to render the data as it was entered. If you use TextCellType instead of GeneralCellType, then it works the same as the Edit cells did in the ActiveX FarPoint Spread, and no FormatInfo is stored in the style model, but the data entered into the cells is always treated as text.

Creating a Custom Sheet Model

You can use a sheet model as a template for a new custom model. For example, consider making a custom data model. Using a custom data model requires creating a class that implements **ISheetDataModel ('ISheetDataModel Interface' in the on-line documentation)**, then setting an instance of the class into the SheetView.Models.Data ('Data Property' in the on-line documentation) property.

ISheetDataModel ('ISheetDataModel Interface' in the on-line documentation) is the only interface required, assuming that you do not need any of the optional interfaces. For more information on the other interfaces, refer to **Understanding the Optional Interfaces**.

 **Note:** In **BaseSheetDataModel ('BaseSheetDataModel Class' in the on-line documentation)**, the Changed event is also implemented.

In a few cases, you might need to create your own custom data model for performance reasons. For example, suppose you want to display a large table of computed values (such as an addition or multiplication table) that consists of a million rows by ten columns. If you used the default sheet data model, you would need to compute and store all ten million values, which would consume a lot of time and memory. Take a look at the following example.

Example

The following example uses formula to set the sum of row and column indices to their respective cells in a sheet with million rows and 10 columns.

C#

```
Stopwatch sw = new Stopwatch();  
sw.Start();//Begin timing  
int ROWCOUNT = 1000000;  
int COLCOUNT = 10;  
fpSpread1.Sheets[0].RowCount = ROWCOUNT;
```

```

fpSpread1.Sheets[0].ColumnCount = COLCOUNT;
for (int r = 0; r < ROWCOUNT; r++)
{
    for (int c = 0; c < COLCOUNT; c++)
    {
        fpSpread1.Sheets[0].Cells[r, c].Value = r + c;
    }
}
sw.Stop();//Stop timing
Console.WriteLine(string.Format("Time elapsed:{0}", sw.Elapsed));

```

Visual Basic

```

Dim sw As New Stopwatch()
sw.Start() 'Begin timing
Dim ROWCOUNT As Integer = 1000000
Dim COLCOUNT As Integer = 10
FpSpread1.Sheets(0).RowCount = ROWCOUNT
FpSpread1.Sheets(0).ColumnCount = COLCOUNT
For r As Integer = 0 To ROWCOUNT - 1
    For c As Integer = 0 To COLCOUNT - 1
        FpSpread1.Sheets(0).Cells(r, c).Value = r + c
    Next
Next
sw.Stop() 'Stop timing
Console.WriteLine(String.Format("Time elapsed:{0}", sw.Elapsed))

```

This example creates your own custom data model and sets cell values. As compare to the above method of setting a formula in a cell, it reduces the processing time significantly.

C#

```

//Custom data model class
class ComputedDataModel : BaseSheetDataModel
{
    public override int RowCount
    {
        get { return 1000000; }
    }
    public override int ColumnCount
    {
        get { return 10; }
    }
    public override object GetValue(int row, int column)
    {
        return row + column;
    }
}
//Write the following in code-behind of a form
Stopwatch sw = new Stopwatch();
sw.Start();//Begin timing
//Set your own custom data model
fpSpread1.Sheets[0].Models.Data = new ComputedDataModel();
sw.Stop();//Stop timing
Console.WriteLine(string.Format("Time elapsed:{0}",sw.Elapsed));

```

Visual Basic

```

'Custom data model class
Class ComputedDataModel
    Inherits BaseSheetDataModel
    Public Overrides Property RowCount As Integer
        Get
            Return 1000000
        End Get
        Set(value As Integer)
        End Set
    End Property
    Public Overrides Property ColumnCount As Integer
        Get
            Return 10
        End Get
        Set(value As Integer)
        End Set
    End Property
    Public Overrides Function GetValue(row As Integer, column As Integer) As Object
        Return row + column
    End Function
End Class
'Write the following in code-behind of a form
Dim sw As New Stopwatch()
sw.Start() 'Begin timing
'Set your own custom data model
FpSpread1.Sheets(0).Models.Data = New ComputedDataModel()
sw.Stop() 'Stop timing
Console.WriteLine(String.Format("Time elapsed:{0}", sw.Elapsed))

```

Understanding the Optional Interfaces

Besides the interfaces that are dedicated to each of the specific models, there are also optional interfaces that provide additional support and may be used when making custom models. These optional interfaces and the customizations they allow are summarized in this table:

Optional Interface

IArraySupport ('IArraySupport Interface' in the on-line documentation)

IDataSourceSupport ('IDataSourceSupport Interface' in the on-line documentation)

IChildModelSupport ('IChildModelSupport Interface' in the on-line documentation)

ICalculationSupport ('ICalculationSupport Interface' in the on-line documentation),

IterationSupport ('IterationSupport Interface' in the on-line documentation)

IDisjointSelections ('IDisjointSelections Interface' in the on-line documentation), IQuerySelection

Customizations Allowed

Allows customization of support for getting and setting arrays of values in a range of cells

Allows customization of data binding on a sheet

Allows customization of hierarchical data models for hierarchies on a sheet; used in conjunction with **IDataSourceSupport ('IDataSourceSupport Interface' in the on-line documentation)**

Allows recalculation of formulas in the cells in the data model.

Allows recursive formulas(with circular references) in the data model.

Allows customization of an ordered array of cell ranges containing the selected cells with the

(IQuerySelection Interface' in the on-line documentation)

minimal overlap between the ranges on a sheet

INamedStyle ('INamedStyleSupport Interface' in the on-line documentation), IParseFormatSupport ('IParseFormatSupport Interface' in the on-line documentation)

Allows customization of collections of custom styles in the style model

INonEmptyCells ('INonEmptyCells Interface' in the on-line documentation)

Allows customization of non-empty counts to find out which rows or columns have data in the cells of that row or column on a sheet

IOptimizedEnumerationSupport ('IOptimizedEnumerationSupport Interface' in the on-line documentation), IOptimizedEnumerationSupport2 ('IOptimizedEnumerationSupport2 Interface' in the on-line documentation)

Allows customization of optimized enumeration for iterating to the next non-empty row or column on a sheet

IUnboundRowSupport ('IUnboundRowSupport Interface' in the on-line documentation)

Allows customization of unbound rows with data binding on a sheet; used in conjunction with **IDataSourceSupport ('IDataSourceSupport Interface' in the on-line documentation)**

None of these optional interfaces are required for saving Excel or text files, or for printing. For more detailed information on these interfaces, refer to the **FarPoint.Win.Spread.Model ('FarPoint.Win.Spread.Model Namespace' in the on-line documentation)** namespace in the Assembly Reference.

For more information on formulas in cells, refer **Managing Formulas in Cells**.

Customizing Row or Column Interaction

You can customize various aspects of user interaction of the spreadsheet in the Spread component. The tasks that relate to customizing the user interaction with the spreadsheet include:

- **Managing Sorting of Rows of User Data**
- **Managing Filtering of Rows of User Data**
- **Managing Grouping of Rows of User Data**
- **Managing Outlines (Range Groups) of Rows and Columns**
- **Customizing User Searching of Data**

You can customize what you allow the user to do. There are several features that are covered in various topics in this section, but they are summarized in one topic for easy reference in **Allowing User Functionality**.

To customize other aspects of interactivity with individual cells, refer to **Customizing Interaction in Cells**.

For details on the spreadsheet objects, refer to these members in the Assembly Reference:

- **Row ('Row Class' in the on-line documentation)**
- **Rows ('Rows Class' in the on-line documentation)**
- **AlternatingRow ('AlternatingRow Class' in the on-line documentation)**
- **AlternatingRows ('AlternatingRows Class' in the on-line documentation)**
- **Column ('Column Property' in the on-line documentation)**
- **Columns ('Columns Class' in the on-line documentation)**

Managing Sorting of Rows of User Data

You can sort the data displayed in the sheet either by column or by row. Typically, all the rows of a sheet are sorted by the values in a particular column. But Spread allows many ways of performing a sort with various properties and methods for each type of sorting. In general, sorting data can be performed and customized by any of the following ways:

- **Allowing the User to Automatically Sort Rows**
- **Using Automatic Sorting**
- **Sorting Rows, Columns, or Ranges**
- **Setting the Appearance of Sort Indicators**
- **Managing Cell Range Sorting**

There are various properties of sorting. The order of the sort can be in ascending order (A to Z, zero to 9) or descending order (Z to A, 9 to zero). The method of comparison can be customized. You can select which values to use as a key when comparing in order to sort the values. The sort indicator, an arrow typically, can be displayed in the header for the column being used as a sort key. For more information on customizing the sorting, refer to the **SortInfo ('SortInfo Class' in the on-line documentation)** object. With this object, you can set the parameters for sorting and then specify this object in the particular sort method you choose.

The cell type does not matter for sorting. The sorting is done depending on the data type of the values in the cells. If you sort cells with data of the DateTime type, then it sorts those cells by date, and if you sort cells with data of the string type, it sorts those cells alphabetically.

Be aware of how sorting works with the data in the models. If you use the automatic sorting by clicking the column header or you call the **SortRows ('SortRows Method' in the on-line documentation)** method of the sheet, then the data model is not sorted, just the data that is displayed to the user. In this case, any data that is hidden before the sort is hidden after the sort, since Spread moves any hidden rows automatically. If you use the **SortRange ('SortRange Method' in the on-line documentation)** method, the data is sorted in the data model and data that is hidden may become visible and vice versa using this method. When you sort data, only the data model is getting sorted. The selection model does not get sorted. If you want the selected row to move, you would need to write code in the **AutoSortedColumn ('AutoSortedColumn Event' in the on-line documentation)** and **AutoSortingColumn**

('AutoSortingColumn Event' in the on-line documentation) events to move the selection. For more information on the models, refer to **Understanding the Underlying Models**.

Sorting performed by clicking column headers sorts only the displayed data and does not affect the order of actual data in the data model; therefore, you can reset the sorted data being displayed to the order of actual data by calling either the **ResetViewRowIndexes** ('ResetViewRowIndexes Method' in the on-line documentation) method or the **ResetViewColumnIndexes** ('ResetViewColumnIndexes Method' in the on-line documentation) method in the **SheetView.DocumentModels** class. You cannot reset the result when the actual data in the data model are sorted with the **SortRange** ('SortRange Method' in the on-line documentation) method in the **SheetView** ('SheetView Class' in the on-line documentation) class.

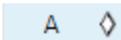
Sorting is not intended to be used when Outlook-style grouping is turned on. For more information about grouping (which is a type of sorting), refer to **Managing Grouping of Rows of User Data**.

For information on sorting within the Spread Designer, refer to the **Spread Designer Guide (on-line documentation)**.

 **Note:** Cell spans become invisible when sorting a sheet with any method except **SortRange**.

Allowing the User to Automatically Sort Rows

You can set the spreadsheet to allow the user to automatically sort the data when a column header is clicked. The first time the column header is clicked (selected) the unsorted icon is displayed. The second click displays the sort icon and sorts the column. If the user clicks successively on the same column, then the direction of the sort is reversed. This does not affect the data model, only how the data is displayed. This figure shows the unsorted icon:



Use the **Column** ('Column Class' in the on-line documentation) object **AllowAutoSort** ('AllowAutoSort Property' in the on-line documentation) property or the **SheetView** ('SheetView Class' in the on-line documentation) **SetColumnAllowAutoSort** ('SetColumnAllowAutoSort Method' in the on-line documentation) method to allow the user to perform automatic sorting when the header cell of a column is clicked. Set the **SortIndicator** ('SortIndicator Property' in the on-line documentation) property of the column you want to show the indicator.

The **SetColumnShowSortIndicator** ('SetColumnShowSortIndicator Method' in the on-line documentation) method or **ShowSortIndicator** ('ShowSortIndicator Property' in the on-line documentation) property can be set to display or hide the sort indicator. The sort indicator appears in the header column as shown in the following figure, illustrating the ascending and descending sort indicators.

Ascending Sort Indicator

	A ▲	B
1	Alignment	
2	Brakes	
3	CarbAdjust	

Descending Sort Indicator

	A ▼	B
1	CarbAdjust	
2	Brakes	
3	Alignment	

When a user sorts data, the **AutoSortingColumn** ('AutoSortingColumn Event' in the on-line documentation) event occurs before the sort and then the **AutoSortedColumn** ('AutoSortedColumn Event' in the on-line documentation) event occurs after the sort.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.

3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet for which to set up sorting.
5. In the properties list, select the **Columns** property and click the button to open the **Cell, Column, and Row Editor**.
6. Select the columns for which you want to allow automatic sorting.
7. In the properties list, select the **AllowAutoSort** property and set the value to True.
8. Click **OK** to close each of the editors.

Using a Shortcut

Use either the **AllowAutoSort** (**'AllowAutoSort Property' in the on-line documentation**) property of the columns or the **SetColumnAllowAutoSort** (**'SetColumnAllowAutoSort Method' in the on-line documentation**) method of the **Sheets** object to allow automatic sorting of the specified columns.

Example

This example allows automatic sorting for the first 30 columns in the sheet.

C#

```
fpSpread1.Sheets[0].Columns[0,29].AllowAutoSort = true;
//or
//fpSpread1.Sheets[0].SetColumnAllowAutoSort(0,30,true);
```

VB

```
fpSpread1.Sheets(0).Columns(0,29).AllowAutoSort = True
'or
'fpSpread1.Sheets(0).SetColumnAllowAutoSort(0,30,True)
```

Using Code

Use the **AllowAutoSort** (**'AllowAutoSort Property' in the on-line documentation**) property to allow automatic sorting of the specified columns.

Example

This example allows automatic sorting for the first 30 columns in the sheet.

C#

```
FarPoint.Win.Spread.Column mycols;
mycols = fpSpread1.ActiveSheet.Columns[0,29];
mycols.AllowAutoSort = true;
```

VB

```
Dim mycols As FarPoint.Win.Spread.Column
mycols = fpSpread1.ActiveSheet.Columns(0,29)
mycols.AllowAutoSort = True
```

Using the Spread Designer

1. Select the sheet tab for the sheet for which you want to allow automatic sorting.
2. From the property list for the sheet, select **Columns**. Click on the button to open the **Cell, Column, and Row**

editor.

3. Select the columns for which you want to allow automatic sorting.
4. In the properties list, select the **AllowAutoSort** property and set the value to **True**.
5. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Using Automatic Sorting

You can sort entire rows or columns automatically in a sheet. The component automatically sorts the rows in a sheet according to the specified column in ascending order unless the sheet was previously automatically sorted ascending. The automatic sorting displays the sort indicator unless the sort indicator for the column has been disabled. This does not affect the data model, only how the data is displayed. Different overloads provide different ways to perform the sorting.

Use the **AutoSortColumn ('AutoSortColumn Method' in the on-line documentation)** method to sort the display in a sheet automatically according to the specified key and use the **SetColumnShowSortIndicator ('SetColumnShowSortIndicator Method' in the on-line documentation)** to set whether to display the sort indicator. The **AutoSortColumn ('AutoSortColumn Method' in the on-line documentation)** method performs the same action as clicking in the column header of the specified column that has its **AllowAutoSort ('AllowAutoSort Property' in the on-line documentation)** property set to True. The **AllowAutoSort ('AllowAutoSort Property' in the on-line documentation)** property does not need to be set to True to use this method. If this method is called successively with the same column index, then the direction of the sort is reversed. If the method is called with a different column index, then the previously sorted column's sort indicator is changed back to **SortIndicator.None** (if there is one) and the specified column is used as the key column in a call to **SortRows ('SortRows Method' in the on-line documentation)** to sort all the rows in the sheet by that column. This affects only the arrangement of rows or columns on a sheet and does not change the arrangement of the data; that is, this does not affect the data model, only how the data is displayed.

 **Note:** The **SetColumnShowSortIndicator ('SetColumnShowSortIndicator Method' in the on-line documentation)** method must be called before the **AutoSortColumn ('AutoSortColumn Method' in the on-line documentation)** method; otherwise, the sort indicator is shown and remains displayed.

Using Code

1. Allow the automatic sorting of a column or columns by using the **SetColumnAllowAutoSort ('SetColumnAllowAutoSort Method' in the on-line documentation)** method.
2. If you want to display a sort indicator, set the column to show the sort indicator with the **SetColumnShowSortIndicator ('SetColumnShowSortIndicator Method' in the on-line documentation)** method.
3. Perform the automatic sort by setting the **AutoSortColumn ('AutoSortColumn Method' in the on-line documentation)** method.

Example

This example automatically sorts the first column.

C#

```
fpSpread1.ActiveSheet.SetColumnAllowAutoSort(0, true);  
fpSpread1.ActiveSheet.SetColumnShowSortIndicator(0, false);  
fpSpread1.ActiveSheet.AutoSortColumn(0);
```

VB

```
fpSpread1.ActiveSheet.SetColumnAllowAutoSort(0, True)  
FpSpread1.ActiveSheet.SetColumnShowSortIndicator(0, False)
```

```
FpSpread1.ActiveSheet.AutoSortColumn(0)
```

Sorting Rows, Columns, or Ranges

You can sort entire rows or columns in a sheet using code or the Spread Designer. To sort all the rows of an entire sheet based on the values of a given column is the most common case, but Spread allows you to sort either rows or columns and to specify which column or row to use as a key for sorting. This sort applies to the entire sheet.

Use the **SortColumns ('SortColumns Method' in the on-line documentation)** (or **SortRows ('SortRows Method' in the on-line documentation)**) method to sort the arrangement of columns (or rows) in a sheet using one or more rows (or columns) as the key. This does not affect the data model, only how the data is displayed. Several overloads provide different ways to sort the columns (or rows). To further customize the way sorting is performed, use the **SortInfo ('SortInfo Class' in the on-line documentation)** object in conjunction with these methods.

You can sort data in a range of cells without re-arranging the entire row or column in a sheet. This may be useful when, for example, you wish to arrange many rows in order of quantity but not include in the sort the final row that contains the totals of those quantities. In this case, you would sort the data in a range of cells but leave the final row, the bottom line, unsorted.

There are two ways to sort data in a range. For bound data, use the **SortRows ('SortRows Method' in the on-line documentation)** and **SortColumns ('SortColumns Method' in the on-line documentation)** methods using the specified parameters in the overloads to specify which range of rows or columns to sort. For unbound data, use the **SortRange ('SortRange Method' in the on-line documentation)** method. For more information on sorting using the Spread Designer at design time, refer to the Using the Spread Designer procedure below.

The **SortRange ('SortRange Method' in the on-line documentation)** method is for unbound data only. This method sorts the data in a range of cells by moving the data around in the data model and moving the cell-level styles along with it. This method is not intended for bound data, as it moves data (not necessarily by entire row or column) and has the effect of moving the data around in the data source.

With the *sortInfo* array of the **SortRange ('SortRange Method' in the on-line documentation)** method, you can specify multiple criteria for sorting the data. This method gives you the ability to sort (or arrange) data in a smaller subset than entire rows or columns in a sheet. For more information, refer to the **SortInfo ('SortInfo Class' in the on-line documentation)** object.

Using Code

To sort rows, use the **SortRows ('SortRows Method' in the on-line documentation)** method; to sort columns, use the **SortColumns ('SortColumns Method' in the on-line documentation)** method.

Example

This example sorts all the rows in the sheet according to the values in the second column. Since column index is zero-based, the second column is 1. The sort indicator is turned on.

C#

```
fpspread1.ActiveSheet.SortRows(1, true, true);
```

VB

```
fpSpread1.ActiveSheet.SortRows(1, True, True)
```

Example

This example sorts rows 12 to 230 using a predefined array of sort information.

C#

```
FarPoint.Win.Spread.SortInfo[] sorter = new FarPoint.Win.Spread.SortInfo[1];
sorter[0] = new FarPoint.Win.Spread.SortInfo(0, false,
System.Collections.Comparer.Default);
fpSpread1.ActiveSheet.SortColumns(12,230,sorter);
```

VB

```
Dim sorter(1) As FarPoint.Win.Spread.SortInfo
sorter(0) = New FarPoint.Win.Spread.SortInfo(0, False,
System.Collections.Comparer.Default)
fpSpread1.ActiveSheet.SortColumns(12,230,sorter)
```

Using the Spread Designer

1. At design time you can sort data using the Spread Designer in a very flexible way. Select the cells you want to sort, either by dragging over the cells or selecting the row or column headers.
2. From the **Data** menu, select **Sort**. The **Sort** dialog is displayed.
3. In the **Sort** dialog, select the options you would like and click **OK**.
4. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting the Appearance of Sort Indicators

You can customize the display of sorting indicators in these ways:

- Using Custom Sort Indicator Images
- Showing and Hiding Sort Indicators
- Determining Which Header Row Displays the Sort Indicators

Using Custom Sort Indicator Images

You can customize the sorting indicator image that appears in the column header of columns that allow sorting. One of the default sort indicators is shown in the header of the first (A) column in the following figure.

	A	B
1	Alignment	
2	Brakes	
3	CarbAdjust	

One way is to override the **PaintSortIndicator** (**'PaintSortIndicator Method' in the on-line documentation**) method in the **CellType.ColumnHeaderRenderer** (**'ColumnHeaderRenderer Class' in the on-line documentation**) class and use your own custom indicator.

Another way is to use the **GetImage** (**'GetImage Method' in the on-line documentation**) and **SetImage** (**'SetImage Method' in the on-line documentation**) methods in the **SpreadView** (**'SpreadView Class' in the on-line documentation**) class, which is described in **Customizing the User Interface Images**.

Showing and Hiding Sort Indicators

You can display for the user or hide from the user the sort indicators that may appear in the column headers. The **Column.ShowSortIndicator** (**'ShowSortIndicator Property' in the on-line documentation**) property and the **SheetView.SetColumnShowSortIndicator** (**'SetColumnSortIndicator Method' in the on-line documentation**) method set whether the column shows the sort indicator the next time that the

SheetView.**AutoSortColumn** ('**AutoSortColumn Method**' in the on-line documentation) method is called for that column. It has no effect until **AutoSortColumn** is called for that column index.

Determining Which Header Row Displays the Sort Indicators

You can specify in which row to display the sort indicators that may appear in the column headers by setting the **ColumnHeaderAutoSortIndex** ('**ColumnHeaderAutoSortIndex Property**' in the on-line documentation) property. By default they appear in the bottom of the column header rows, but if you have more than one header row, you can specify which row. You can specify which row displays the automatic text by setting the **ColumnHeaderAutoTextIndex** ('**ColumnHeaderAutoTextIndex Property**' in the on-line documentation) property (or the **ColumnHeader.AutoTextIndex** ('**AutoTextIndex Property**' in the on-line documentation) property); however, changing the row that displays the automatic text does not change the row that displays the sort indicator.

Using Code

To create a custom sort indicator:

- Create a new column header renderer class.
- Customize the sort indicator that appears in the column header.

Example

The following example creates a custom sort indicator.

C#

```
// In the form load section, allow sorting (and filtering).
private void Form1_Load(object sender, System.EventArgs e)
{
    fpSpread1.ActiveSheet.ColumnHeader.DefaultStyle.Renderer = new
myColumnHeaderRenderer();
    fpSpread1.Sheets[0].Columns[0].AllowAutoSort =true;
    fpSpread1.Sheets[0].Columns[0].AllowAutoFilter =true;
}
// Define a new column header renderer.
public class myColumnHeaderRenderer : FarPoint.Win.Spread.CellType.ColumnHeaderRenderer
{
    // Override the sorting indicator paint method.
    override public void PaintSortIndicator(Graphics g, Rectangle r,
FarPoint.Win.Spread.Appearance appearance, float zoomFactor)
    {
        g.FillRectangle(new SolidBrush(Color.Red), r);
    }
    // Override the filtering indicator paint method.
    override public void PaintFilterIndicator(Graphics g, Rectangle r,
FarPoint.Win.Spread.Appearance appearance, float zoomFactor)
    {
        g.FillRectangle(new SolidBrush(Color.Blue), r);
    }
} //End Form1_Load
```

VB

```
' Define a new column header renderer.
Public Class myColumnHeaderRenderer
Inherits FarPoint.Win.Spread.CellType.ColumnHeaderRenderer
```

```

' Override the sorting indicator paint method.
Public Overrides Sub PaintSortIndicator(ByVal g As Graphics, ByVal r As Rectangle,
ByVal appearance As FarPoint.Win.Spread.Appearance, ByVal zoomFactor As Single)
    g.FillRectangle(New SolidBrush(Color.Red), r)
End Sub 'PaintSortIndicator
' Override the filtering indicator paint method.
Public Overrides Sub PaintFilterIndicator(ByVal g As Graphics, ByVal r As Rectangle,
ByVal appearance As FarPoint.Win.Spread.Appearance, ByVal zoomFactor As Single)
    g.FillRectangle(New SolidBrush(Color.Blue), r)
End Sub 'PaintFilterIndicator
End Class 'myColumnHeaderRenderer
' In the form load section, allow sorting (and filtering).
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    fpSpread1.ActiveSheet.ColumnHeader.DefaultStyle.Renderer = New myColumnHeaderRenderer
    fpSpread1.Sheets(0).Columns(0).AllowAutoSort = True
    fpSpread1.Sheets(0).Columns(0).AllowAutoFilter = True
End Sub 'Form1_Load

```

Managing Cell Range Sorting

Spread for WinForms allows users to manage range sorting operations without any hassle via applying sort logic on a specific cell range in a worksheet. This makes it easier and quicker to manipulate data and arrange records based on a specific sort order while working with bulk data in spreadsheets.

You can sort textual information in alphabetical order (like A-Z or Z-A), arrange numbers in ascending and descending order, sort dates from newest to oldest and vice versa and order data based on the colors and icons in a cell range.

Users can work with cell range sorting in the following two ways:

1. **Using Sort Object of AutoFilter**
2. **Using Sort Object in Worksheet**

Using Sort Object of AutoFilter

You can apply sort logic on the data in a cell range using the **Apply** method of the **ISort** interface and the **Add** method of the **ISortFields** interface.

Example

Refer to the following example code in order to apply sort logic on a range of cells in a spreadsheet.

C#

```

// Gets the sort column
ISort filterSort = activeSheet.AutoFilter.Sort;

// Sort column C by ascending order.
filterSort.SortFields.Add(1);

// Set Sort Order for column B by descending
filterSort.SortFields.Add(0).Order = GrapeCity.Spreadsheet.SortOrder.Descending;
filterSort.Apply();

```

VB

```

'Gets the sort column

```

```
Dim filterSort As ISort = activeSheet.AutoFilter.Sort

'Sort column C by ascending order
filterSort.SortFields.Add(1)

'Set Sort Order for column B by descending
filterSort.SortFields.Add(0).Order = GrapeCity.Spreadsheet.SortOrder.Descending
filterSort.Apply()
```

Using Sort Object in Worksheet

You can add and apply sort logic on a cell range using the **Add** method of the **ISortFields** interface and the **Apply** method.

Using Code

In order to apply the sort logic in a cell range using the sort object of the worksheet, you can use the **Apply** method of the **ISort** interface.

Example

Refer to the following example code to access the current filter settings in a worksheet.

C#

```
// Sort column C
worksheetSort.SortFields.Add("C1")
// Sort range is B1:E10
worksheetSort.SetRange("B1:E10");
worksheetSort.Apply();
```

VB

```
'Sort column C
worksheetSort.SortFields.Add("C1")

'Sort range is B1:E10
worksheetSort.SetRange("B1:E10")
worksheetSort.Apply()
```

Managing Filtering of Rows of User Data

Each type of filtering provides a way for users to change data's appearance or temporarily hide data based on conditions that they specify.

Filtering includes the following tasks.

- **Allowing the User to Filter Rows**
- **Setting the Appearance of Filtered Rows**
- **Customizing Simple Filtering**
- **Customizing Enhanced Filtering**
- **Customizing the Filter Bar**
- **Setting the Appearance of Filter Indicators**
- **Managing Cell Range Filtering**

Allowing the User to Filter Rows

By default, the spreadsheet does not allow the user to filter the rows of a spreadsheet. You can turn on this feature and allow row filtering for all or specific columns in a sheet. Hidden rows are not displayed even if they match the filter criteria.

Use the **HideRowFilter** ('**HideRowFilter Class**' in the on-line documentation) class or **StyleRowFilter** ('**StyleRowFilter Class**' in the on-line documentation) class depending on whether you want to hide rows that are filtered or change the style of those rows. Use the **Column.AllowAutoFilter** ('**AllowAutoFilter Property**' in the on-line documentation) property to turn on filtering for a given column.

Use the **AutoFilterMode** ('**AutoFilterMode Property**' in the on-line documentation) property to specify the type of filtering. You can specify the simple filter style (FilterGadget), the enhanced filter style (EnhancedContextMenu), or the enhanced filter style with filter bar (FilterBar).

The user can then click the items in the drop-down to filter the rows. To understand how the filtering works for the end user, refer to **Understanding Simple Row Filtering**.

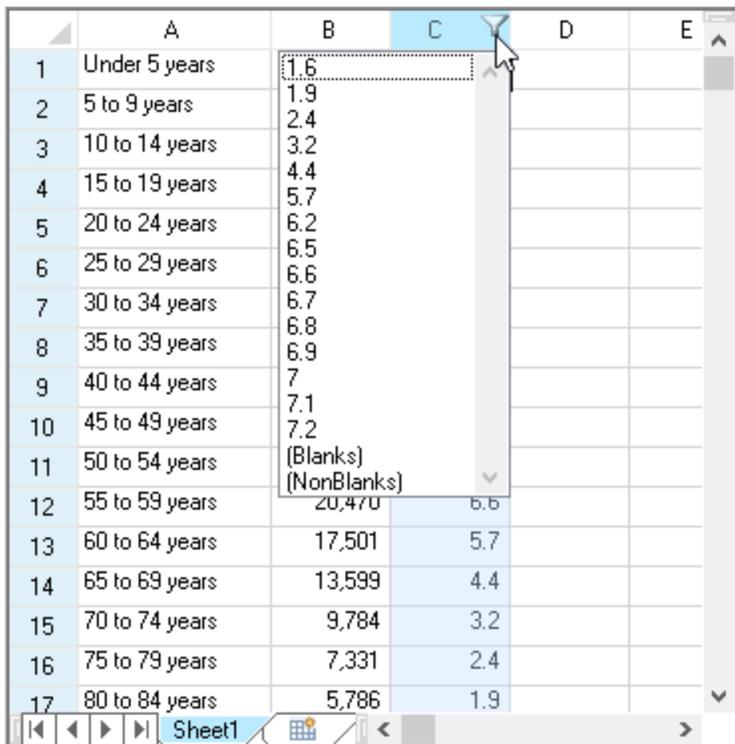
To understand the other aspects of filtering that can be customized, return to the overview at **Managing Filtering of Rows of User Data**.

Using Code

Allow row filtering (hiding rows that are filtered out) based on values in one column. Make sure the column headers are visible. (Refer to **Showing or Hiding Headers**.)

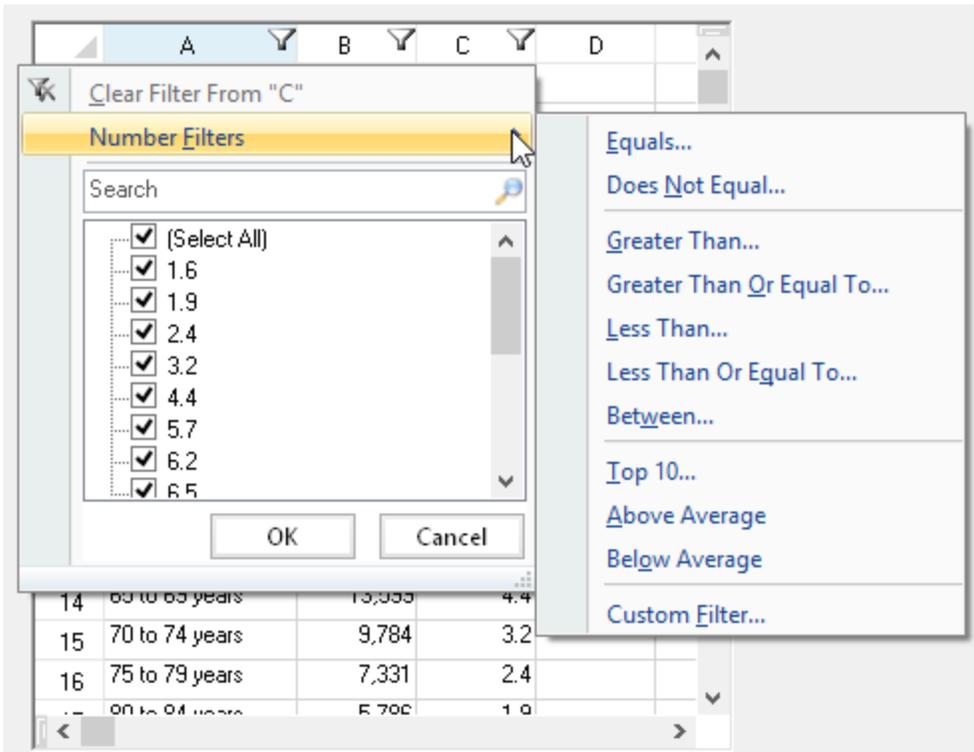
Select the type of row filtering, using the **HideRowFilter** ('**HideRowFilter Class**' in the on-line documentation) class, then allow that filtering for a specified column, using the **AllowAutoFilter** ('**AllowAutoFilter Property**' in the on-line documentation) property.

Simple filter (Specify `AutoFilterMode.FilterGadget`)

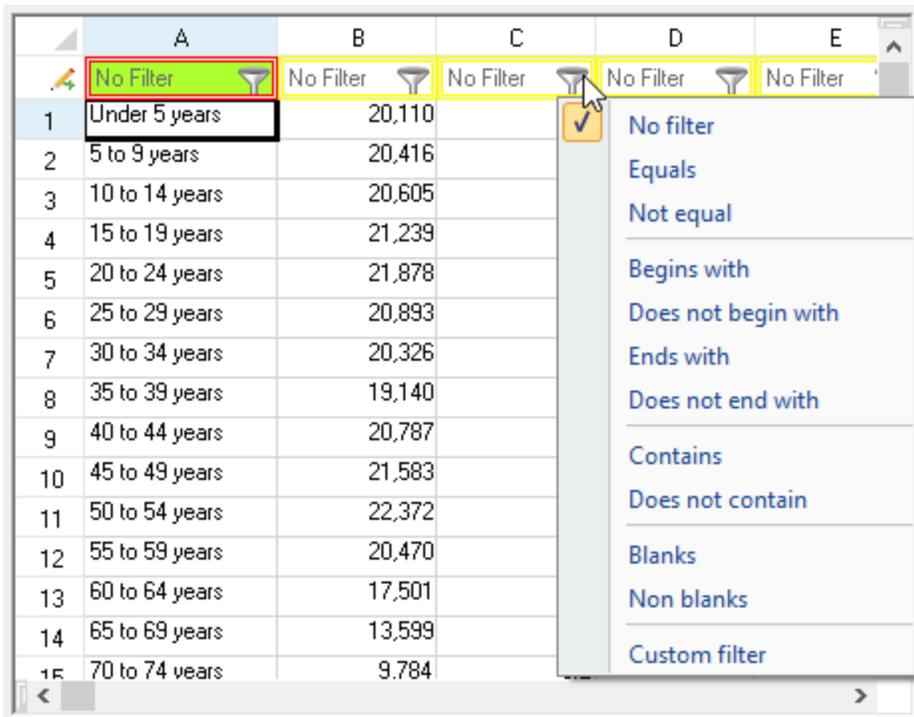


	A	B	C	D	E
1	Under 5 years	1.6			
2	5 to 9 years	1.9			
3	10 to 14 years	2.4			
4	15 to 19 years	3.2			
5	20 to 24 years	4.4			
6	25 to 29 years	5.7			
7	30 to 34 years	6.2			
8	35 to 39 years	6.5			
9	40 to 44 years	6.6			
10	45 to 49 years	6.7			
11	50 to 54 years	6.8			
12	55 to 59 years	6.9			
13	60 to 64 years	7			
14	65 to 69 years	7.1			
15	70 to 74 years	7.2			
16	75 to 79 years	(Blanks)			
17	80 to 84 years	(NonBlanks)			

Enhanced filter (Specify AutoFilterMode.EnhancedContextMenu)



Filter bar (AutoFilterMode.FilterBar)



Example

The following example sets up simple filtering and assumes that you have some data in the cells, either by assigning values or binding to a data source.

C#

```
fpSpread1.ActiveSheet.ColumnHeaderVisible = true;
FarPoint.Win.Spread.HideRowFilter hideRowFilter = new
FarPoint.Win.Spread.HideRowFilter(fpSpread1.ActiveSheet);
fpSpread1.ActiveSheet.Columns[1,3].AllowAutoFilter = true;
fpSpread1.ActiveSheet.AutoFilterMode = FarPoint.Win.Spread.AutoFilterMode.FilterGadget;
```

VB

```
fpSpread1.ActiveSheet.ColumnHeaderVisible = True
Dim hideRowFilter As New FarPoint.Win.Spread.HideRowFilter(fpSpread1.ActiveSheet)
fpSpread1.ActiveSheet.Columns(1, 3).AllowAutoFilter = True
fpSpread1.ActiveSheet.AutoFilterMode = FarPoint.Win.Spread.AutoFilterMode.FilterGadget
```

Using the Spread Designer

1. Select a column by clicking on the column header.
2. Set **AllowAutoFilter** under the **Misc** section of the Property window.
3. Use the **File** menu, then **Apply and Exit** to save the changes.

Setting the Appearance of Filtered Rows

You can customize the appearance of filtered rows to allow you to see which rows are filtered in and which ones are filtered out. Rows that meet the criteria for the row filter are said to be "filtered in"; rows that do not meet the criteria are said to be "filtered out." Filtering may either hide the rows that are filtered out, or change the styles for both filtered-in and filtered-out rows. If you want the styles to change, so that you can continue to display all the data but highlight rows that match some criteria, then you must define a filtered-in style and a filtered-out style.

You define styles by creating NamedStyle objects that contain all the style settings. Then when the row filtering is applied to a column, you specify those defined style settings by referring to the NamedStyle object for that filtered state.

In the figure below, from the example code given here, the choice of Gibson in the filter items results in the rows with that filter item being formatted with one appearance style and all the other rows being formatted with another appearance style.

	A	B	C
1	Fender	AST-100 DMC	
2	Gibson	Les Paul Standard Double Cut Plus	
3	Fender	ST58-70TX	
4	Ibanez	AGS83B	
5	Gibson	Les Paul Supreme	
6	Yamaha	ATTITUDE-Limited II	

For more information about the row filter that uses styles, refer to the **StyleRowFilter ('StyleRowFilter Class' in the on-line documentation)** class.

Using Code

1. Define the style settings in a **NamedStyle** object.
2. Apply the **NamedStyle** object to the filter.
3. Allow filtering for a specified column or columns.

Example

This example creates a filter.

C#

```
// Define styles to apply to filtered rows.
FarPoint.Win.Spread.NamedStyle inStyle = new FarPoint.Win.Spread.NamedStyle();
FarPoint.Win.Spread.NamedStyle outStyle = new FarPoint.Win.Spread.NamedStyle();
inStyle.BackColor = Color.LightCyan;
inStyle.ForeColor = Color.DarkRed;
outStyle.BackColor = Color.LemonChiffon;
outStyle.ForeColor = Color.Green;

// Create a new filter column definition for the first column (Definition of the
// default setting).
FarPoint.Win.Spread.FilterColumnDefinition fcdef = new
```

```

FarPoint.Win.Spread.FilterColumnDefinition(0);
// Create a StyleRowFilter object (Style filter), and add the above filtering column
definition.
FarPoint.Win.Spread.StyleRowFilter styleFilter = new
FarPoint.Win.Spread.StyleRowFilter(fpSpread1.ActiveSheet, inStyle, outStyle);
styleFilter.AddColumn(fcdef);
// Set the row filtering object you created to sheets.
fpSpread1.ActiveSheet.RowFilter = styleFilter;
fpSpread1.ActiveSheet.AutoFilterMode = FarPoint.Win.Spread.AutoFilterMode.FilterGadget;

// Fill the data area with text data.
fpSpread1.ActiveSheet.DefaultStyle.CellType = new
FarPoint.Win.Spread.CellType.TextCellType();
fpSpread1.ActiveSheet.SetText(0, 0, "Fender");
fpSpread1.ActiveSheet.SetText(1, 0, "Gibson");
fpSpread1.ActiveSheet.SetText(2, 0, "Fender");
fpSpread1.ActiveSheet.SetText(3, 0, "Ibanez");
fpSpread1.ActiveSheet.SetText(4, 0, "Gibson");
fpSpread1.ActiveSheet.SetText(5, 0, "Yamaha");
fpSpread1.ActiveSheet.SetText(0, 1, "AST-100 DMC");
fpSpread1.ActiveSheet.SetText(1, 1, "Les Paul Standard Double Cut Plus");
fpSpread1.ActiveSheet.SetText(2, 1, "ST58-70TX");
fpSpread1.ActiveSheet.SetText(3, 1, "AGS83B");
fpSpread1.ActiveSheet.SetText(4, 1, "Les Paul Supreme");
fpSpread1.ActiveSheet.SetText(5, 1, "ATTITUDE-Limited II");
fpSpread1.ActiveSheet.SetColumnWidth(0, 90);
fpSpread1.ActiveSheet.SetColumnWidth(1, 210);

```

VB

```

' Create a named style object for Filter-In rows.
Dim inStyle As New FarPoint.Win.Spread.NamedStyle
Dim outStyle As New FarPoint.Win.Spread.NamedStyle
inStyle.BackColor = Color.LightCyan
inStyle.ForeColor = Color.DarkRed
outStyle.BackColor = Color.LemonChiffon
outStyle.ForeColor = Color.Green

' Create a new filter column definition for the first column (definition of the default
setting).
Dim fcdef As New FarPoint.Win.Spread.FilterColumnDefinition(0)
' Create a StyleRowFilter object (Style filter), and add the above filtering column
definition.
Dim styleFilter As New FarPoint.Win.Spread.StyleRowFilter(fpSpread1.ActiveSheet,
inStyle, outStyle)
styleFilter.AddColumn(fcdef)
' Set the row filtering object you created to sheets.
fpSpread1.ActiveSheet.RowFilter = styleFilter
fpSpread1.ActiveSheet.AutoFilterMode = FarPoint.Win.Spread.AutoFilterMode.FilterGadget

' Fill the data area with text data.
fpSpread1.ActiveSheet.DefaultStyle.CellType = New
FarPoint.Win.Spread.CellType.TextCellType
fpSpread1.ActiveSheet.SetText(0, 0, "Fender")
fpSpread1.ActiveSheet.SetText(1, 0, "Gibson")
fpSpread1.ActiveSheet.SetText(2, 0, "Fender")
fpSpread1.ActiveSheet.SetText(3, 0, "Ibanez")

```

```

fpSpread1.ActiveSheet.SetText(4, 0, "Gibson")
fpSpread1.ActiveSheet.SetText(5, 0, "YAMAHA")
fpSpread1.ActiveSheet.SetText(0, 1, "AST-100 DMC")
fpSpread1.ActiveSheet.SetText(1, 1, "Les Paul Standard Double Cut Plus")
fpSpread1.ActiveSheet.SetText(2, 1, "ST58-70TX")
fpSpread1.ActiveSheet.SetText(3, 1, "AGS83B")
fpSpread1.ActiveSheet.SetText(4, 1, "Les Paul Supreme")
fpSpread1.ActiveSheet.SetText(5, 1, "ATTITUDE-Limited II")
fpSpread1.ActiveSheet.SetColumnWidth(0, 90)
fpSpread1.ActiveSheet.SetColumnWidth(1, 210)

```

This example illustrates how to set a hidden filter by creating the filter definition for the first column, adding it to the **HideRowFilter** (**'HideRowFilter Class' in the on-line documentation**) object, and then setting the HideRowFilter object to a sheet.

C#

```

// Create a new filter column definition for the first column (Definition as per
// default settings).
FarPoint.Win.Spread.FilterColumnDefinition fcdef = new
FarPoint.Win.Spread.FilterColumnDefinition(0);
// Create a HideRowFilter object (Style filter) and add the above filtering column
// definition to it.
FarPoint.Win.Spread.HideRowFilter hideFilter = new
FarPoint.Win.Spread.HideRowFilter(fpSpread1.ActiveSheet);
hideFilter.AddColumn(fcdef);
// Set the created row filtering object to the sheet.
fpSpread1.ActiveSheet.RowFilter = hideFilter;
fpSpread1.ActiveSheet.AutoFilterMode = FarPoint.Win.Spread.AutoFilterMode.FilterGadget;

```

Visual Basic

```

' Create a new filter column definition for the first column (Definition as per default
// settings).
Dim fcdef As New FarPoint.Win.Spread.FilterColumnDefinition(0)
' Create a HideRowFilter object (Style filter) and add the above filtering column
// definition to it.
Dim hideFilter As New FarPoint.Win.Spread.HideRowFilter(FpSpread1.ActiveSheet)
hideFilter.AddColumn(fcdef)
' Set the created row filtering object to the sheet.
FpSpread1.ActiveSheet.RowFilter = hideFilter
FpSpread1.ActiveSheet.AutoFilterMode = FarPoint.Win.Spread.AutoFilterMode.FilterGadget

```

Customizing Simple Filtering

You can customize many things about simple filtering, including the contents, order, and appearance of the filter list, and the appearance of filter indicators.

For instructions, see the following topics.

- **Understanding Simple Row Filtering**
- **Customizing the Filter List**
 - **Defining the Contents of the Filter Item List**
 - **Defining the Order of the Items in the Filter Item List**
 - **Setting the Appearance of the Display of the Filter Item List**
- **Creating a Custom Filter**

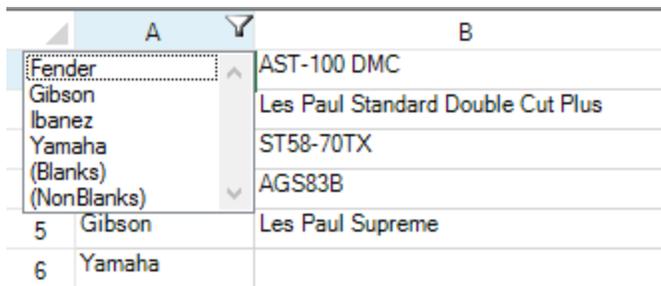
Understanding Simple Row Filtering

This topic summarizes how the end user interacts with the simple filter feature.

Once you have row filtering applied to a column (as described in **Allowing the User to Filter Rows**), an indicator appears in the column header. The different appearances of the row filtering indicator are summarized in this table:

Row Filtering Indicator	Description
	Appearance of header cell with no row filtering; this occurs when there is no row filtering, or when row filtering is off
	Appearance of header cell with row filtering allowed but no rows filtered; this occurs when row filtering is set to (All) thus not restricting rows based on the contents of this column
	Appearance of header cell with row filtering allowed and some rows filter; this occurs when row filtering some of the rows based on the contents of this column

The column header displays the row filtering indicator, a drop-down arrow symbol. Clicking on this indicator provides a drop-down list of the filter choices. Picking an item from this list causes that filter to be applied and all the rows meeting that condition (in this column) are filtered. The default drop-down list contains all the unique text values in cells in this column. The figure below shows an example of a drop-down list of filters.



The table below summarizes the entries in the drop-down list.

Filter List Item	Description
(All)	Include or allow all the rows in this column regardless of content
[contents]	Include or allow only those rows with this particular cell content in this column
(Blanks)	Include or allow only rows that have blanks (empty cells) in this column
(NonBlanks)	Include or allow only rows that have non-blanks (non-empty cells) in this column, in other words any cell that has any content

You can customize the way this list is displayed, as described in **Customizing the Filter List**. You can create custom filters to add to the drop-down list, as described in **Creating a Custom Filter**.

For a given sheet, multiple columns may have filtering set. The different columns may have different filters, depending on the contents of cells in that column. The results of filtering would be similar to what one would expect with primary and secondary keys when sorting data. The choice from the filter list from the initial column would filter some rows, leaving the choices in the subsequent filter list to be a subset of the total possible. By selecting choices from more than one filter, the results include only those rows that satisfy all the selected filtering conditions.

For more information on setting up row filters, refer to **Allowing the User to Filter Rows**. For more information on setting the appearance of filtered rows, refer to **Setting the Appearance of Filtered Rows**.

To understand the aspects of filtering that can be customized, return to the overview at **Managing Filtering of Rows**

of User Data.

Customizing the Filter List

When the user clicks on the row filter indicator, a drop-down list of filter items is displayed. You can customize that drop-down list in these ways:

- **Defining the Contents of the Filter Item List**
- **Defining the Order of the Items in the Filter Item List**
- **Setting the Appearance of the Display of the Filter Item List**

The user can then click the items in the drop-down to filter the rows. For more information on using the row filters, refer to **Understanding Simple Row Filtering**.

Defining the Contents of the Filter Item List

You can filter all rows in a sheet based on criteria of the contents of a particular cell in a column. To set up row filters, follow these basic steps:

1. Define filter criteria
2. Define filter result behavior (change styles of rows or hide rows)
3. Define any custom filters
4. Apply filter

Define the filter criteria for each column, which are called the column filter definitions. This is the criteria that is used to filter rows based on the contents of the column and is assigned to an individual column. Combine these individual column criteria or column filter definitions into a collection.

Define the appearance of the rows to be filtered, either by defining a filtered-in style and a filtered-out style or by deciding to hide the filtered out rows. For more information about styles and the appearance of rows of filtered data, refer to **Setting the Appearance of Filtered Rows**.

You can customize the words that appear in the following choices in the drop-down list, using the corresponding properties in the **DefaultRowFilter ('DefaultRowFilter Class' in the on-line documentation)** class.

- All - **AllString ('AllString Property' in the on-line documentation)** Property
- Blanks - **BlanksString ('BlanksString Property' in the on-line documentation)** Property
- NonBlanks - **NonBlanksString ('NonBlanksString Property' in the on-line documentation)** Property

Apply the row filtering to all or specific columns in a sheet (which applies the column filter definition collection to the columns of that sheet).

Using Code

1. Define the column filter definitions.
2. Group them into a collection.
3. Define the styles.
4. Apply the row filter.

Example

The following example sets up filtered rows, including filtered row styles.

C#

```
// Declare the row filter and column definitions.
FarPoint.Win.Spread.FilterColumnDefinitionCollection fcdc = new
FarPoint.Win.Spread.FilterColumnDefinitionCollection();
FarPoint.Win.Spread.FilterColumnDefinition fcd1 = new
FarPoint.Win.Spread.FilterColumnDefinition(2);
FarPoint.Win.Spread.FilterColumnDefinition fcd2 = new
FarPoint.Win.Spread.FilterColumnDefinition(3);
FarPoint.Win.Spread.FilterColumnDefinition fcd3 = new
FarPoint.Win.Spread.FilterColumnDefinition(1);
// Add column filter definitions to a collection.
fcdc.Add(fcd1);
fcdc.Add(fcd2);
fcdc.Add(fcd3);
FarPoint.Win.Spread.NamedStyle inStyle = new FarPoint.Win.Spread.NamedStyle();
FarPoint.Win.Spread.NamedStyle outStyle = new FarPoint.Win.Spread.NamedStyle();
inStyle.BackColor = Color.Yellow;
outStyle.BackColor = Color.Aquamarine;
// Apply styles and column filter definitions to the row filter.
FarPoint.Win.Spread.StyleRowFilter rowFilter = new
FarPoint.Win.Spread.StyleRowFilter(fpSpread1.ActiveSheet, inStyle, outStyle);
// Apply the column definition to the filter.
rowFilter.ColumnDefinitions = fcdc;
// Apply the row filter to the sheet.
fpSpread1.ActiveSheet.RowFilter = rowFilter;
fpSpread1.ActiveSheet.AutoFilterMode = FarPoint.Win.Spread.AutoFilterMode.FilterGadget;
```

VB

```
' Declare the row filter and column definitions
Dim fcdc As New FarPoint.Win.Spread.FilterColumnDefinitionCollection()
Dim fcd1 As New FarPoint.Win.Spread.FilterColumnDefinition(2)
Dim fcd2 As New FarPoint.Win.Spread.FilterColumnDefinition(3)
Dim fcd3 As New FarPoint.Win.Spread.FilterColumnDefinition(1)
' Add column filter definitions to a collection.
fcdc.Add(fcd1)
fcdc.Add(fcd2)
fcdc.Add(fcd3)
Dim inStyle As New FarPoint.Win.Spread.NamedStyle()
Dim outStyle As New FarPoint.Win.Spread.NamedStyle()
inStyle.BackColor = Color.Yellow
outStyle.BackColor = Color.Aquamarine
' Apply styles and column filter definitions to the row filter.
Dim rowFilter As New FarPoint.Win.Spread.StyleRowFilter(fpSpread1.ActiveSheet, inStyle,
outStyle)
' Apply the column definition to the filter.
rowFilter.ColumnDefinitions = fcdc
' Apply the row filter to the sheet.
fpSpread1.ActiveSheet.RowFilter = rowFilter
fpSpread1.ActiveSheet.AutoFilterMode = FarPoint.Win.Spread.AutoFilterMode.FilterGadget
```

Example

This example code creates row filters in a drop-down list that can be accessed by clicking on the drop-down arrow icon in the column header. Two filters are created (hide and style). Comment out the style filter to see the hide filter.

This example defines filters based on criteria from the contents of columns 1 and 2 of the spreadsheet using the text of

the items in the columns as the filter choices.

C#

```
// Set the rows to hide when they are filtered out.
FarPoint.Win.Spread.HideRowFilter hideRowFilter = new
FarPoint.Win.Spread.HideRowFilter(fpSpread1.ActiveSheet);
hideRowFilter.AddColumn(1);
hideRowFilter.AddColumn(2);
fpSpread1.ActiveSheet.RowFilter = hideRowFilter;

// Set the styles for the filtered-in rows and filtered-out rows.
FarPoint.Win.Spread.NamedStyle inStyle = new FarPoint.Win.Spread.NamedStyle();
FarPoint.Win.Spread.NamedStyle outStyle = new FarPoint.Win.Spread.NamedStyle();
FarPoint.Win.Spread.StyleRowFilter styleRowFilter = new
FarPoint.Win.Spread.StyleRowFilter(fpSpread1.ActiveSheet, inStyle, outStyle);
inStyle.BackColor = Color.Yellow;
outStyle.BackColor = Color.Aquamarine;
// Apply the row filter to the two columns.
styleRowFilter.AddColumn(1);
styleRowFilter.AddColumn(2);
fpSpread1.ActiveSheet.RowFilter = styleRowFilter;

// Fill the cells with test data.
fpSpread1.ActiveSheet.Cells[0,1].Value = "aaa";
fpSpread1.ActiveSheet.Cells[1,1].Value = "aaa";
fpSpread1.ActiveSheet.Cells[2,1].Value = "bbb";
fpSpread1.ActiveSheet.Cells[3,1].Value = "ccc";
fpSpread1.ActiveSheet.Cells[4,1].Value = "ddd";
fpSpread1.ActiveSheet.Cells[5,1].Value = "bbb";
fpSpread1.ActiveSheet.Cells[6,1].Value = "aaa";
fpSpread1.ActiveSheet.Cells[7,1].Value = "eee";
fpSpread1.ActiveSheet.Cells[8,1].Value = "jjj";
fpSpread1.ActiveSheet.Cells[9,1].Value = "jjj";
fpSpread1.ActiveSheet.Cells[10,1].Value = "fff";
fpSpread1.ActiveSheet.Cells[11,1].Value = "fff";
fpSpread1.ActiveSheet.Cells[12,1].Value = "eee";
fpSpread1.ActiveSheet.Cells[13,1].Value = "jjj";
fpSpread1.ActiveSheet.Cells[14,1].Value = "eee";
fpSpread1.ActiveSheet.Cells[15,1].Value = "jjj";
fpSpread1.ActiveSheet.Cells[16,1].Value = "fff";
fpSpread1.ActiveSheet.Cells[0,2].Value = "111";
fpSpread1.ActiveSheet.Cells[1,2].Value = "222";
fpSpread1.ActiveSheet.Cells[2,2].Value = "333";
fpSpread1.ActiveSheet.Cells[3,2].Value = "222";
fpSpread1.ActiveSheet.Cells[4,2].Value = "555";
fpSpread1.ActiveSheet.Cells[5,2].Value = "444";
fpSpread1.ActiveSheet.Cells[6,2].Value = "444";
fpSpread1.ActiveSheet.Cells[0,3].Value = "North";
fpSpread1.ActiveSheet.Cells[1,3].Value = "South";
fpSpread1.ActiveSheet.Cells[2,3].Value = "East";
fpSpread1.ActiveSheet.Cells[3,3].Value = "South";
fpSpread1.ActiveSheet.Cells[4,3].Value = "North";
fpSpread1.ActiveSheet.Cells[5,3].Value = "North";
fpSpread1.ActiveSheet.Cells[6,3].Value = "West";
```

VB

```
' Set the rows to hide when they are filtered out.
Dim hRowFilter As New FarPoint.Win.Spread.HideRowFilter(fpSpread1.ActiveSheet)
hRowFilter.AddColumn(1)
hRowFilter.AddColumn(2)
fpSpread1.ActiveSheet.RowFilter = hRowFilter

' Set the styles for the filtered-in rows and filtered-out rows.
Dim inStyle As New FarPoint.Win.Spread.NamedStyle()
Dim outStyle As New FarPoint.Win.Spread.NamedStyle()
Dim styleRowFilter As New FarPoint.Win.Spread.StyleRowFilter(fpSpread1.ActiveSheet,
inStyle, outStyle)
inStyle.BackColor = Color.Yellow
outStyle.BackColor = Color.Aquamarine
' Apply the row filter to the two columns.
styleRowFilter.AddColumn(1)
styleRowFilter.AddColumn(2)
fpSpread1.ActiveSheet.RowFilter = styleRowFilter

' Fill the cells with test data.
fpSpread1.ActiveSheet.Cells(0, 1).Value = "aaa"
fpSpread1.ActiveSheet.Cells(1, 1).Value = "aaa"
fpSpread1.ActiveSheet.Cells(2, 1).Value = "bbb"
fpSpread1.ActiveSheet.Cells(3, 1).Value = "ccc"
fpSpread1.ActiveSheet.Cells(4, 1).Value = "ddd"
fpSpread1.ActiveSheet.Cells(5, 1).Value = "bbb"
fpSpread1.ActiveSheet.Cells(6, 1).Value = "aaa"
fpSpread1.ActiveSheet.Cells(7, 1).Value = "eee"
fpSpread1.ActiveSheet.Cells(8, 1).Value = "jjj"
fpSpread1.ActiveSheet.Cells(9, 1).Value = "jjj"
fpSpread1.ActiveSheet.Cells(10, 1).Value = "fff"
fpSpread1.ActiveSheet.Cells(11, 1).Value = "fff"
fpSpread1.ActiveSheet.Cells(12, 1).Value = "eee"
fpSpread1.ActiveSheet.Cells(13, 1).Value = "jjj"
fpSpread1.ActiveSheet.Cells(14, 1).Value = "eee"
fpSpread1.ActiveSheet.Cells(15, 1).Value = "jjj"
fpSpread1.ActiveSheet.Cells(16, 1).Value = "fff"
fpSpread1.ActiveSheet.Cells(0, 2).Value = "111"
fpSpread1.ActiveSheet.Cells(1, 2).Value = "222"
fpSpread1.ActiveSheet.Cells(2, 2).Value = "333"
fpSpread1.ActiveSheet.Cells(3, 2).Value = "222"
fpSpread1.ActiveSheet.Cells(4, 2).Value = "555"
fpSpread1.ActiveSheet.Cells(5, 2).Value = "444"
fpSpread1.ActiveSheet.Cells(6, 2).Value = "444"
fpSpread1.ActiveSheet.Cells(0, 3).Value = "North"
fpSpread1.ActiveSheet.Cells(1, 3).Value = "South"
fpSpread1.ActiveSheet.Cells(2, 3).Value = "East"
fpSpread1.ActiveSheet.Cells(3, 3).Value = "South"
fpSpread1.ActiveSheet.Cells(4, 3).Value = "North"
fpSpread1.ActiveSheet.Cells(5, 3).Value = "North"
fpSpread1.ActiveSheet.Cells(6, 3).Value = "West"
```

Defining the Order of the Items in the Filter Item List

You can customize how the drop-down list of filter items is displayed. By default, the list shows the possible filter items alphabetically and includes all the options. By changing the value of the **FilterListBehavior** (**FilterListBehavior**

Enumeration' in the on-line documentation) enumeration, you change how the filter list is displayed. For example you can set the filter list to display items in order of number of occurrences in that column.

Use the **AddColumn ('AddColumn Method' in the on-line documentation)** methods and specify the column filter definition. This also defines the way the filter items appear in the drop-down.

Using Code

Set the **FilterListBehavior ('FilterListBehavior Enumeration' in the on-line documentation)** enumeration to change how the filter list is displayed.

Example

The following example illustrates setting the **FilterListBehavior ('FilterListBehavior Enumeration' in the on-line documentation)** enumeration in code.

C#

```
FarPoint.Win.Spread.NamedStyle instyle = new FarPoint.Win.Spread.NamedStyle();
FarPoint.Win.Spread.NamedStyle outstyle = new FarPoint.Win.Spread.NamedStyle();
instyle.BackColor = Color.Yellow;
outstyle.BackColor = Color.Aquamarine;
FarPoint.Win.Spread.FilterColumnDefinition fcd = new
FarPoint.Win.Spread.FilterColumnDefinition(1,
FarPoint.Win.Spread.FilterListBehavior.SortByMostOccurrences |
FarPoint.Win.Spread.FilterListBehavior.Default);
FarPoint.Win.Spread.FilterColumnDefinition fcd1 = new
FarPoint.Win.Spread.FilterColumnDefinition(2);
FarPoint.Win.Spread.FilterColumnDefinition fcd2 = new
FarPoint.Win.Spread.FilterColumnDefinition();
FarPoint.Win.Spread.StyleRowFilter sf = new
FarPoint.Win.Spread.StyleRowFilter(fpSpread1.ActiveSheet, instyle, outstyle);
sf.AddColumn(fcd);
sf.AddColumn(fcd1);
sf.AddColumn(fcd2);
fpSpread1.ActiveSheet.RowFilter = sf;
fpSpread1.ActiveSheet.AutoFilterMode = FarPoint.Win.Spread.AutoFilterMode.FilterGadget;
```

VB

```
Dim instyle As New FarPoint.Win.Spread.NamedStyle
Dim outstyle As New FarPoint.Win.Spread.NamedStyle
instyle.BackColor = Color.Yellow
outstyle.BackColor = Color.Aquamarine
Dim fcd As New FarPoint.Win.Spread.FilterColumnDefinition(1,
FarPoint.Win.Spread.FilterListBehavior.SortByMostOccurrences Or
FarPoint.Win.Spread.FilterListBehavior.Default)
Dim fcd1 As New FarPoint.Win.Spread.FilterColumnDefinition(2)
Dim fcd2 As New FarPoint.Win.Spread.FilterColumnDefinition
Dim sf As New FarPoint.Win.Spread.StyleRowFilter(fpSpread1.ActiveSheet, instyle,
outstyle)
sf.AddColumn(fcd)
sf.AddColumn(fcd1)
sf.AddColumn(fcd2)
fpSpread1.ActiveSheet.RowFilter = sf
fpSpread1.ActiveSheet.AutoFilterMode = FarPoint.Win.Spread.AutoFilterMode.FilterGadget
```

Setting the Appearance of the Display of the Filter Item List

You can set the appearance of the outline of the drop-down list. The following figures show the types of outlines (or border) styles.

Outline (Border) Style

Fixed, three-dimensional (default)

Example

A		B
Fender		AST-100 DMC
Gibson		Les Paul Standard Double Cut Plus
Ibanez		ST58-70TX
YAMAHA		AGS83B
(Blanks)		
(NonBlanks)		
5	Gibson	Les Paul Supreme
6	YAMAHA	ATTITUDE-Limited II

Fixed, single-line

A		B
Fender		AST-100 DMC
Gibson		Les Paul Standard Double Cut Plus
Ibanez		ST58-70TX
YAMAHA		AGS83B
(Blanks)		
(NonBlanks)		
5	Gibson	Les Paul Supreme
6	YAMAHA	ATTITUDE-Limited II

None

A		B
Fender		AST-100 DMC
Gibson		Les Paul Standard Double Cut Plus
Ibanez		ST58-70TX
YAMAHA		AGS83B
(Blanks)		
(NonBlanks)		
5	Gibson	Les Paul Supreme
6	YAMAHA	ATTITUDE-Limited II

For more details, refer to the **DefaultRowFilter** class **DropDownBorderStyle** ('**DropDownBorderStyle Property**' in the on-line documentation) property and the .NET **BorderStyle** enumeration.

Using Code

1. Set the **AllowAutoFilter** ('**AllowAutoFilter Property**' in the on-line documentation) property.
2. Set the **DropDownBorderStyle** ('**DropDownBorderStyle Property**' in the on-line documentation) property.

Example

This example creates a filter and sets the border style.

C#

```
// Activate the automatic filtering features.
fpSpread1.ActiveSheet.Columns[0].AllowAutoFilter = true;
```

```

fpSpread1.ActiveSheet.AutoFilterMode = FarPoint.Win.Spread.AutoFilterMode.FilterGadget;
// Change the drop-down list style to "Single Line".
fpSpread1.ActiveSheet.RowFilter.DropDownBorderStyle = BorderStyle.FixedSingle;

fpSpread1.ActiveSheet.DefaultStyle.CellType = new
FarPoint.Win.Spread.CellType.TextCellType();
fpSpread1.ActiveSheet.SetText(0, 0, "Fender");
fpSpread1.ActiveSheet.SetText(1, 0, "Gibson");
fpSpread1.ActiveSheet.SetText(2, 0, "Fender");
fpSpread1.ActiveSheet.SetText(3, 0, "Ibanez");
fpSpread1.ActiveSheet.SetText(4, 0, "Gibson");
fpSpread1.ActiveSheet.SetText(5, 0, "YAMAHA");
fpSpread1.ActiveSheet.SetText(0, 1, "AST-100 DMC");
fpSpread1.ActiveSheet.SetText(1, 1, "Les Paul Standard Double Cut Plus");
fpSpread1.ActiveSheet.SetText(2, 1, "ST58-70TX");
fpSpread1.ActiveSheet.SetText(3, 1, "AGS83B");
fpSpread1.ActiveSheet.SetText(4, 1, "Les Paul Supreme");
fpSpread1.ActiveSheet.SetText(5, 1, "ATTITUDE-Limited II");
fpSpread1.ActiveSheet.SetColumnWidth(0, 90);
fpSpread1.ActiveSheet.SetColumnWidth(1, 210);

```

VB

```

' Activate the automatic filtering features.
fpSpread1.ActiveSheet.Columns(0).AllowAutoFilter = True
' Change the drop-down list style to "Single Line".
fpSpread1.ActiveSheet.AutoFilterMode = FarPoint.Win.Spread.AutoFilterMode.FilterGadget
fpSpread1.ActiveSheet.RowFilter.DropDownBorderStyle = BorderStyle.FixedSingle

fpSpread1.ActiveSheet.DefaultStyle.CellType = New
FarPoint.Win.Spread.CellType.TextCellType
fpSpread1.ActiveSheet.SetText(0, 0, "Fender")
fpSpread1.ActiveSheet.SetText(1, 0, "Gibson")
fpSpread1.ActiveSheet.SetText(2, 0, "Fender")
fpSpread1.ActiveSheet.SetText(3, 0, "Ibanez")
fpSpread1.ActiveSheet.SetText(4, 0, "Gibson")
fpSpread1.ActiveSheet.SetText(5, 0, "YAMAHA")
fpSpread1.ActiveSheet.SetText(0, 1, "AST-100 DMC")
fpSpread1.ActiveSheet.SetText(1, 1, "Les Paul Standard Double Cut Plus")
fpSpread1.ActiveSheet.SetText(2, 1, "ST58-70TX")
fpSpread1.ActiveSheet.SetText(3, 1, "AGS83B")
fpSpread1.ActiveSheet.SetText(4, 1, "Les Paul Supreme")
fpSpread1.ActiveSheet.SetText(5, 1, "ATTITUDE-Limited II")
fpSpread1.ActiveSheet.SetColumnWidth(0, 90)
fpSpread1.ActiveSheet.SetColumnWidth(1, 210)

```

Creating a Custom Filter

You can create a custom filter that you can then include in the column filter definition collection. In order to create a custom filter, follow these steps:

1. Create a class that inherits from `FarPoint.Win.Spread.BaseFilterItem` (**'BaseFilterItem Class' in the on-line documentation**) or `FarPoint.Win.Spread.DefaultFilterItem` (**'DefaultFilterItem Class' in the on-line documentation**).
2. Override the **DisplayName** (**'DisplayName Property' in the on-line documentation**) property to return the name to be displayed in the drop-down list of filter items.

3. Override the **ShowInDropDown ('ShowInDropDown Method' in the on-line documentation)** method to specify if this filter item should be displayed in the drop-down list given the current filtered in rows.
4. Override the **Filter ('Filter Method' in the on-line documentation)** method to perform the filter action on the specified column.
5. Override the **Serialize ('Serialize Method' in the on-line documentation)** and **Deserialize ('Deserialize Method' in the on-line documentation)** methods. Make calls to the **base.Serialize** and **base.Deserialize** methods unless your methods handle persisting the default properties.
6. Create a **HideRowFilter ('HideRowFilter Class' in the on-line documentation)** or **StyleRowFilter ('StyleRowFilter Class' in the on-line documentation)** object.
7. Add the custom filter to the custom filter's list of the column filter definition in the row filtering object from the previous step.

For more details, refer to these members:

- **BaseFilterItem ('BaseFilterItem Class' in the on-line documentation)**
- **IFilterItem ('IFilterItem Interface' in the on-line documentation)**
- **DefaultFilterItem ('DefaultFilterItem Class' in the on-line documentation)**
- **FilterItemCollection ('FilterItemCollection Class' in the on-line documentation)**

Example

This example creates a custom filter that filters rows with values between 1000 and 5000 and sets the custom filter in the second column of the sheet.

C#

```
[Serializable()]
public class CustomFilter : FarPoint.Win.Spread.BaseFilterItem
{
    FarPoint.Win.Spread.SheetView sv = null;
    public CustomFilter() { }
    public override string DisplayName
    {
        // String to be displayed in the filter
        get { return "1000 to 5000"; }
    }
    public override FarPoint.Win.Spread.SheetView SheetView
    {
        set { sv = value; }
    }
    private bool IsNumeric(object ovalue)
    {
        System.Text.RegularExpressions.Regex _isNumber = new
System.Text.RegularExpressions.Regex(@"^\-?\d+\.\d*$");
        System.Text.RegularExpressions.Match m = _isNumber.Match(Convert.ToString(ovalue));
        return m.Success;
    }
    public bool IsFilteredIn(object ovalue)
    {
        bool ret = false;
        if (IsNumeric(ovalue))
        {
            if (Double.Parse(Convert.ToString(ovalue)) >= 1000 &&
Double.Parse(Convert.ToString(ovalue)) <= 5000)
                ret = true;
        }
        return ret;
    }
    public override bool ShowInDropDown(int columnIndex, int[] filteredInRowList)
```

```
{
    // filteredInRowList argument is displayed in the item list only when
    // there is data that meets the filtered in row list condition.
    if (filteredInRowList == null)
    {
        for (int i = 0; i < sv.RowCount; i++)
        {
            object value = sv.GetValue(i, columnIndex);
            if (value != null)
            {
                if (IsFilteredIn(value))
                    return true;
            }
        }
    }
    else
    {
        // Check if the current row list meets the condition
        for (int i = 0; i < filteredInRowList.Length; i++)
        {
            int row = filteredInRowList[i];
            object value = sv.GetValue(row, columnIndex);
            if (value != null)
            {
                if (IsFilteredIn(value))
                    return true;
            }
        }
    }
    return false;
}

public override int[] Filter(int columnIndex)
{
    System.Collections.ArrayList ar = new System.Collections.ArrayList();
    object val;
    for (int i = 0; i < sv.RowCount; i++)
    {
        val = sv.GetValue(i, columnIndex);
        if (IsFilteredIn(val))
            ar.Add(i); // Add row numbers to the list that match the conditions
    }
    return (Int32[]) (ar.ToArray(typeof(Int32)));
}

public override bool Serialize(System.Xml.XmlTextWriter w)
{
    w.WriteStartElement("CustomFilter");
    base.Serialize(w);
    w.WriteEndElement();
    return true;
}

public override bool Deserialize(System.Xml.XmlNodeReader r)
{
    if (r.NodeType == System.Xml.XmlNodeType.Element)
    {
        if (r.Name.Equals("CustomFilter"))
        {
            base.Deserialize(r);
        }
    }
}
```

```

        return true;
    }
}
// Write the following in the code-behind.
// Create a filter column definition in the second column.
FarPoint.Win.Spread.FilterColumnDefinition fcd1 = new
FarPoint.Win.Spread.FilterColumnDefinition(1,
    FarPoint.Win.Spread.FilterListBehavior.Custom |
FarPoint.Win.Spread.FilterListBehavior.Default);
// Add custom filter to column definition.
fcd1.Filters.Add(new CustomFilter() { SheetView = fpSpread1.Sheets[0] });
// Create hidden filter.
FarPoint.Win.Spread.HideRowFilter hideRowFilter = new
FarPoint.Win.Spread.HideRowFilter(fpSpread1.Sheets[0]);
hideRowFilter.AddColumn(fcd1);
fpSpread1.Sheets[0].RowFilter = hideRowFilter;
// Set test data.
fpSpread1.Sheets[0].SetValue(0, 1, 999);
fpSpread1.Sheets[0].SetValue(1, 1, 1000);
fpSpread1.Sheets[0].SetValue(2, 1, 5000);

```

Visual Basic

```

<Serializable>
Public Class CustomFilter
    Inherits FarPoint.Win.Spread.BaseFilterItem
    Private sv As FarPoint.Win.Spread.SheetView = Nothing
    Public Sub New()
    End Sub
    Public Overrides ReadOnly Property DisplayName() As String
        ' String to be displayed in the filter
    Get
        Return "1000 to 5000"
    End Get
    End Property
    Public Overrides WriteOnly Property SheetView() As FarPoint.Win.Spread.SheetView
    Set
        sv = Value
    End Set
    End Property
    Private Function IsNumeric(ovalue As Object) As Boolean
        Dim _isNumber As New System.Text.RegularExpressions.Regex("^\\-?\\d+\\.?\\d*$")
        Dim m As System.Text.RegularExpressions.Match =
        _isNumber.Match(Convert.ToString(ovalue))
        Return m.Success
    End Function
    Public Function IsFilteredIn(ovalue As Object) As Boolean
        Dim ret As Boolean = False
        If IsNumeric(ovalue) Then
            If [Double].Parse(Convert.ToString(ovalue)) >= 1000 AndAlso
[Double].Parse(Convert.ToString(ovalue)) <= 5000 Then
                ret = True
            End If
        End If
        Return ret
    End Function
    Public Overrides Function ShowInDropDown(columnIndex As Integer, filteredInRowList As
Integer()) As Boolean
        ' filteredInRowList argument is displayed in the item list only when

```

```

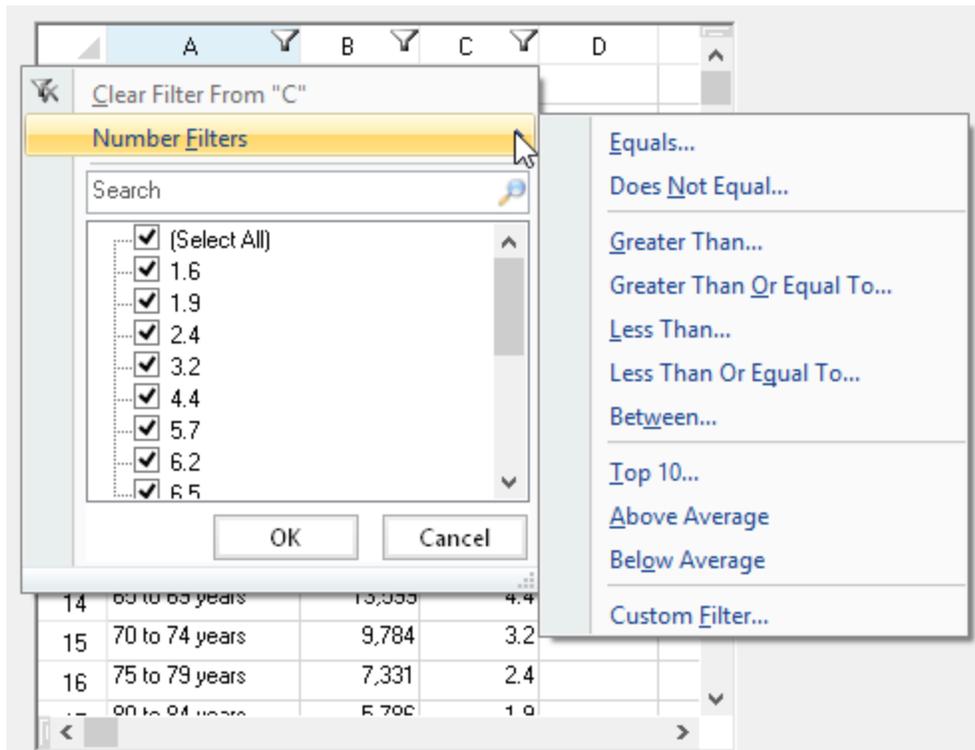
' there is data that meets the filtered in row list condition.
If filteredInRowList Is Nothing Then
    For i As Integer = 0 To sv.RowCount - 1
        Dim value As Object = sv.GetValue(i, columnIndex)
        If value IsNot Nothing Then
            If IsFilteredIn(value) Then
                Return True
            End If
        End If
    Next
Else
    ' Check if the current row list meets the condition
    For i As Integer = 0 To filteredInRowList.Length - 1
        Dim row As Integer = filteredInRowList(i)
        Dim value As Object = sv.GetValue(row, columnIndex)
        If value IsNot Nothing Then
            If IsFilteredIn(value) Then
                Return True
            End If
        End If
    Next
End If
Return False
End Function
Public Overrides Function Filter(columnIndex As Integer) As Integer()
    Dim ar As New System.Collections.ArrayList()
    Dim val As Object
    For i As Integer = 0 To sv.RowCount - 1
        val = sv.GetValue(i, columnIndex)
        If IsFilteredIn(val) Then
            ar.Add(i)
            ' Add row numbers to the list that match the conditions
        End If
    Next
    Return DirectCast(ar.ToArray(GetType(Int32)), Int32())
End Function
Public Overrides Function Serialize(w As System.Xml.XmlTextWriter) As Boolean
    w.WriteStartElement("CustomFilter")
    MyBase.Serialize(w)
    w.WriteEndElement()
    Return True
End Function
Public Overrides Function Deserialize(r As System.Xml.XmlNodeReader) As Boolean
    If r.NodeType = System.Xml.XmlNodeType.Element Then
        If r.Name.Equals("CustomFilter") Then
            MyBase.Deserialize(r)
        End If
    End If
    Return True
End Function
End Class
' Write the following in the code-behind.
' Create a filter column definition in the second column.
Dim fcd1 As New FarPoint.Win.Spread.FilterColumnDefinition(1,
    FarPoint.Win.Spread.FilterListBehavior.Custom Or
FarPoint.Win.Spread.FilterListBehavior.Default)
' Add custom filter to column definition.
fcd1.Filters.Add(New CustomFilter() With {.SheetView = FpSpread1.Sheets(0)})
' Create hidden filter.

```

```
Dim hideRowFilter As New FarPoint.Win.Spread.HideRowFilter(FpSpread1.Sheets(0))
hideRowFilter.AddColumn(fcd1)
FpSpread1.Sheets(0).RowFilter = hideRowFilter
' Set test data.
FpSpread1.Sheets(0).SetValue(0, 1, 999)
FpSpread1.Sheets(0).SetValue(1, 1, 1000)
FpSpread1.Sheets(0).SetValue(2, 1, 5000)
```

Customizing Enhanced Filtering

When the control has enhanced filtering turned on, the user can drop-down a list of available filters to apply to the data, as shown in the following figure.



Filters list

The default filter that is displayed depends on the data in the column. The filter can be a number, text, date, or color filter.

The filters are described in the following table.

Filter Type	Description
Number Filters	
Equals	Values in rows are equal to condition
Does Not Equal	Values in rows do not equal condition
Greater Than	Values in rows are greater than condition
Greater Than Or Equal To	Values in rows are greater than or equal to condition

Less Than	Values in rows are less than condition
Less Than Or Equal To	Values in rows are less than or equal to condition
Between	Values in rows are greater than one condition and less than another condition
Top 10	Values in the rows with the ten highest values
Above Average	Values in the rows that are above the average of the values in all the rows
Below Average	Values in the rows that are below the average of the values in all the rows
Custom Filter	Values in rows that meet the conditions of a custom filter

Text Filters

Equals	Values in rows equal the condition
Does Not Equal	Values in rows do not equal the condition
Begins With	Values in rows begin with the specified characters
Ends With	Values in rows end with the specified characters
Contains	Values in rows contain the specified characters
Does Not Contain	Values in rows do not contain the specified characters
Custom Filter	Values in rows that meet the conditions of a custom filter

Date Filters

Equals	Values in rows equal the condition
Before	Values in rows are dates before the condition
After	Values in rows are dates after the condition
Between	Values in rows are dates between two specified dates for the condition
Tomorrow	Values in rows are tomorrow's date
Today	Values in rows are today's date
Yesterday	Values in rows are yesterday's date
Next Week	Values in rows are during next week
This Week	Values in rows are during current week
Last Week	Values in rows are during last week
Next Month	Values in rows are during next month
This Month	Values in rows are during current month
Last Month	Values in rows are during last month
Next Quarter	Values in rows are during next quarter
This Quarter	Values in rows are during current quarter
Last Quarter	Values in rows are during last quarter
Next Year	Values in rows are during next year
This Year	Values in rows are during current year
Last Year	Values in rows are during last year
Year to Date	Values in rows are during current year to present date
All Dates in the Period	Values in rows are within a specified period

Custom Filter

Values in rows that meet the conditions of a custom filter

Users can specify wildcards in conditions. The "?" character represents any single character. The "*" character represents any series of characters.

When the user chooses a filter, the control either filters the data to display only the items that match the filter criteria, or the control displays the rows that meet the criteria with one appearance, and the rows that do not meet the criteria with another appearance. For information about setting the styles for rows, see **Setting the Appearance of Filtered Rows**.

Filter indicators

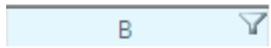
In the control, columns with filters display filter indicators, which indicate whether a filter has been applied, as shown in the following table.

Row Filtering Indicator

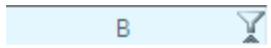
Description



Appearance of header cell with no row filtering

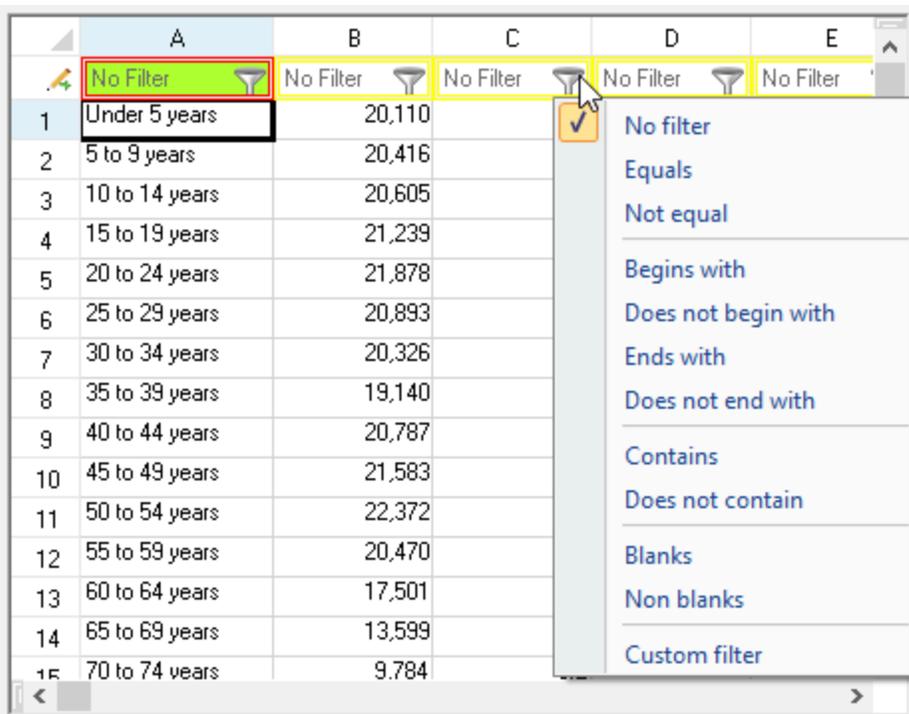


Appearance of header cell with row filtering allowed but no rows filtered



Appearance of header cell with row filtering allowed and some rows filtered

If you prefer, you can have the control display a filter bar that allows the user to choose the filter to apply. The filter bar is displayed in the control at all times, and choosing a filter from the filter bar makes the filter go into effect immediately. The following figure illustrates a control with a filter bar.



Using Code

1. Set the **AllowAutoFilter** ('AllowAutoFilter Property' in the on-line documentation) property.

2. Set the **AutoFilterMode** ('**AutoFilterMode Property**' in the on-line documentation) property.

Example

The following example creates an enhanced filter in the first three columns. Add different types of data to see the various filter options.

C#

```
fpSpread1.Sheets[0].Columns[0, 2].AllowAutoFilter = true;
fpSpread1.Sheets[0].AutoFilterMode =
FarPoint.Win.Spread.AutoFilterMode.EnhancedContextMenu;
```

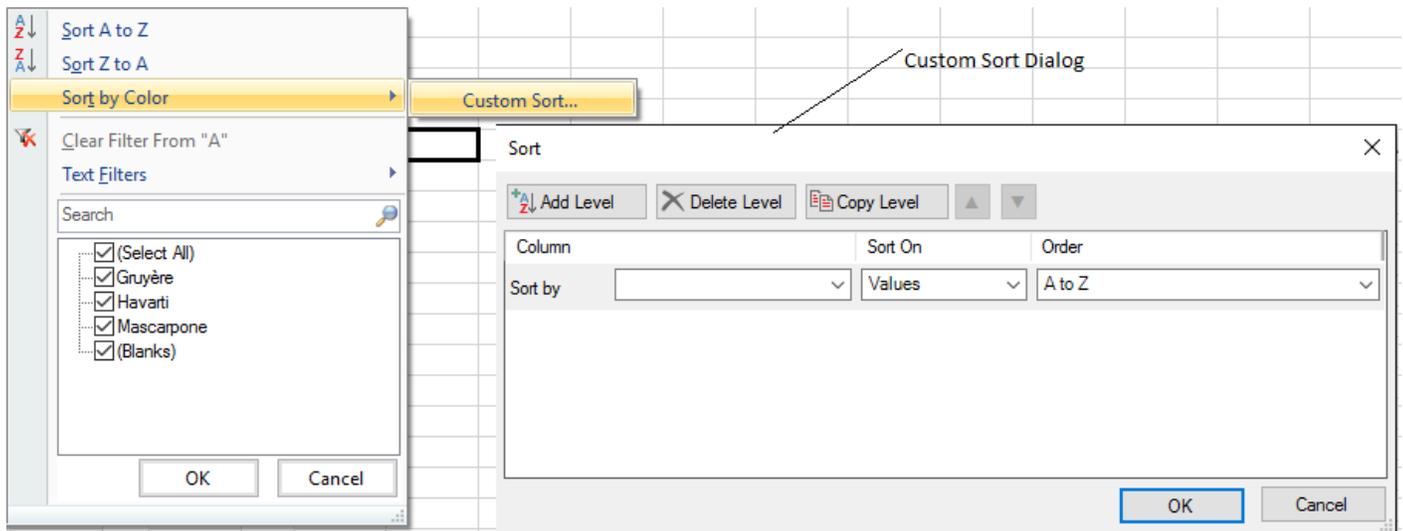
VB

```
FpSpread1.Sheets(0).Columns(0, 2).AllowAutoFilter = True
FpSpread1.Sheets(0).AutoFilterMode =
FarPoint.Win.Spread.AutoFilterMode.EnhancedContextMenu
```

You can customize the appearance of the filter bar, as described in **Customizing the Filter Bar**.

Adding a Custom Sort Dialog

You can add a custom sort dialog in the enhanced row filter. The custom sort dialog allows you to add multiple sort keys and specify what to sort on.



You can also specify the sorting behavior with the **AutoSortEnhancedMode** ('**AutoSortEnhancedMode Property**' in the on-line documentation) property.

Using Code

1. Set the **AutoFilterMode** ('**AutoFilterMode Property**' in the on-line documentation) property to **EnhancedContextMenu**.
2. Set the **AutoSortEnhancedContextMenu** ('**AutoSortEnhancedContextMenu Property**' in the on-line documentation) property to true.
3. Set the **AllowAutoFilter** ('**AllowAutoFilter Property**' in the on-line documentation) property for the columns to be filtered.

4. Set the **AllowAutoSort** ('**AllowAutoSort Property**' in the on-line documentation) property for the columns to be sorted.

Example

This example adds the custom sort dialog option.

C#

```
fpSpread1.ActiveSheet.AutoFilterMode =  
FarPoint.Win.Spread.AutoFilterMode.EnhancedContextMenu;  
    fpSpread1.ActiveSheet.AutoSortEnhancedContextMenu = true;  
    fpSpread1.ActiveSheet.Columns[0].AllowAutoFilter = true;  
    fpSpread1.ActiveSheet.Columns[0].AllowAutoSort = true;  
    fpSpread1.ActiveSheet.Columns[1].AllowAutoFilter = true;  
    fpSpread1.ActiveSheet.Columns[1].AllowAutoSort = true;
```

VB

```
fpSpread1.ActiveSheet.AutoFilterMode =  
FarPoint.Win.Spread.AutoFilterMode.EnhancedContextMenu  
    fpSpread1.ActiveSheet.AutoSortEnhancedContextMenu = True  
    fpSpread1.ActiveSheet.Columns[0].AllowAutoFilter = True  
    fpSpread1.ActiveSheet.Columns[0].AllowAutoSort = True  
    fpSpread1.ActiveSheet.Columns[1].AllowAutoFilter = True  
    fpSpread1.ActiveSheet.Columns[1].AllowAutoSort = True
```

Using the Spread Designer

1. Select **Sheet** in the property grid.
2. Set **AutoFilterMode** to **EnhancedContextMenu**.
3. Set **AutoSortEnhancedContextMenu** to true.
4. Select the column in the Spread Designer. Then set **AllowAutoFilter** and **AllowAutoSort** to true in the property grid.

Customizing the Filter Bar

You can customize the appearance of the filter bar. You can change the background and text colors and the grid lines and their color. The following figure illustrates a filter bar with a custom appearance.

	A	B	C	D	E
	No Filter	No Filter	No Filter	No Filter	No Filter
1	Under 5 years	20,110	6.5		
2	5 to 9 years	20,416	6.6		
3	10 to 14 years	20,605	6.7		
4	15 to 19 years	21,239	6.9		
5	20 to 24 years	21,878	7.1		
6	25 to 29 years	20,893	6.8		
7	30 to 34 years	20,326	6.6		
8	35 to 39 years	19,140	6.2		
9	40 to 44 years	20,787	6.7		
10	45 to 49 years	21,583	7		
11	50 to 54 years	22,372	7.2		
12	55 to 59 years	20,470	6.6		
13	60 to 64 years	17,501	5.7		
14	65 to 69 years	13,599	4.4		
15	70 to 74 years	9,784	3.2		

The filter bar also provides a date picker to pick a date to filter by. Certain filter menu choices will display the date picker (before or after, for example). You can also type the value in the edit portion of the filter after you select a filter menu option.

	A	DateTime
	No Fil...	Before
1		10/1/2014 10:05:57 AM

Setting the **AutoFormat ('AutoFormat Property' in the on-line documentation)** property to true specifies to use the **DateTImeFormatInfo ('DateTImeFormatInfo Property' in the on-line documentation)**, **FormatString ('FormatString Property' in the on-line documentation)**, and **NumberFormatInfo ('NumberFormatInfo Property' in the on-line documentation)** properties to format the value in the filter bar. Set these properties if the format of the data in the cell is different from the format in the filter bar. The Equals filter menu option requires that the cell format and the filter bar format be the same.

Using Code

1. To customize specific cells in the filter bar, set the **FilterBar ('FilterBar Class' in the on-line documentation)** class's **Cells ('Cells Property' in the on-line documentation)** properties.
2. To customize the filter bar overall, set the FilterBar's **DefaultStyle ('DefaultStyle Property' in the on-line documentation)**, **Height ('Height Property' in the on-line documentation)**, **HorizontalGridLine ('HorizontalGridLine Property' in the on-line documentation)**, and **VerticalGridLine ('VerticalGridLine Property' in the on-line documentation)** properties.

Example

The following example sets one cell in the filter bar to display a custom border and background color, and the entire filter bar to display a custom border.

C#

```
FarPoint.Win.Spread.SheetView sheetView = fpSpread1.ActiveSheet;
```

```
sheetView.AutoFilterMode = FarPoint.Win.Spread.AutoFilterMode.FilterBar;
sheetView.FilterBar.Cells[0].Border = new FarPoint.Win.DoubleLineBorder(Color.Red);
sheetView.FilterBar.Cells[0].BackColor = Color.GreenYellow;
sheetView.FilterBar.DefaultStyle.Border = new
FarPoint.Win.DoubleLineBorder(Color.Yellow);
```

VB

```
Dim sheetView As FarPoint.Win.Spread.SheetView = fpSpread1.ActiveSheet
sheetView.AutoFilterMode = FarPoint.Win.Spread.AutoFilterMode.FilterBar
sheetView.FilterBar.Cells(0).Border = New FarPoint.Win.DoubleLineBorder(Color.Red)
sheetView.FilterBar.Cells(0).BackColor = Color.GreenYellow
sheetView.FilterBar.DefaultStyle.Border = New
FarPoint.Win.DoubleLineBorder(Color.Yellow)
```

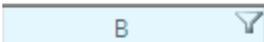
Setting the Appearance of Filter Indicators

The filter indicators are displayed in the column header when row filtering is available to the user. You can customize various aspects of the filter indicators. You can do any of the following:

- **Using Custom Filter Indicator Images**
- **Showing or Hiding Filter Indicators**
- **Determining Which Header Row Displays the Indicators**

Using Custom Filter Indicator Images

You can customize the filter indicator image that appears in the column header of columns that have filters assigned. One of the default images is shown here in the second (B) column.



One way is to override the **PaintFilterIndicator** ('**PaintFilterIndicator Method**' in the **on-line documentation**) method in the **CellType ColumnHeaderRenderer** ('**ColumnHeaderRenderer Class**' in the **on-line documentation**) class and create your own custom filter indicator.

Another way is to use the **GetImage** ('**GetImage Method**' in the **on-line documentation**) and **SetImage** ('**SetImage Method**' in the **on-line documentation**) methods in the **SpreadView** class, which is described in **Customizing the User Interface Images**.

Using Code

1. Create a new column header renderer with the **ColumnHeaderRenderer** ('**ColumnHeaderRenderer Class**' in the **on-line documentation**) class.
2. Override the **PaintSortIndicator** ('**PaintSortIndicator Method**' in the **on-line documentation**) method.
3. Override the **PaintFilterIndicator** ('**PaintFilterIndicator Method**' in the **on-line documentation**) method.

Example

This example illustrates how to set create a custom filter in code, by first creating a new column header renderer, and then customizing the filter indicator that appears in the column header.

C#

```
// In the form load section, allow filtering (and sorting).
private void Form1_Load(object sender, System.EventArgs e)
{
    fpSpread1.ActiveSheet.ColumnHeader.DefaultStyle.Renderer = new
myColumnHeaderRenderer();
    fpSpread1.Sheets[0].Columns[0].AllowAutoSort =true;
    fpSpread1.Sheets[0].Columns[0].AllowAutoFilter =true;
}
// Define a new column header renderer.
public class myColumnHeaderRenderer : FarPoint.Win.Spread.CellType.ColumnHeaderRenderer
{
    // Override the sorting indicator paint method.
    override public void PaintSortIndicator(Graphics g, Rectangle r,
FarPoint.Win.Spread.Appearance appearance, float zoomFactor)
    {
        g.FillRectangle(new SolidBrush(Color.Red), r);
    }
    // Override the filtering indicator paint method.
    override public void PaintFilterIndicator(Graphics g, Rectangle r,
FarPoint.Win.Spread.Appearance appearance, float zoomFactor)
    {
        g.FillRectangle(new SolidBrush(Color.Blue), r);
    }
} //End Form1_Load
```

VB

```
' Define a new column header renderer.
Public Class myColumnHeaderRenderer
Inherits FarPoint.Win.Spread.CellType.ColumnHeaderRenderer
    ' Override the sorting indicator paint method.
    Public Overrides Sub PaintSortIndicator(ByVal g As Graphics, ByVal r As Rectangle,
ByVal appearance As FarPoint.Win.Spread.Appearance, ByVal zoomFactor As Single)
        g.FillRectangle(New SolidBrush(Color.Red), r)
    End Sub 'PaintSortIndicator
    ' Override the filtering indicator paint method.
    Public Overrides Sub PaintFilterIndicator(ByVal g As Graphics, ByVal r As Rectangle,
ByVal appearance As FarPoint.Win.Spread.Appearance, ByVal zoomFactor As Single)
        g.FillRectangle(New SolidBrush(Color.Blue), r)
    End Sub 'PaintFilterIndicator
End Class 'myColumnHeaderRenderer
' In the form load section, allow sorting and filtering.
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    fpSpread1.ActiveSheet.ColumnHeader.DefaultStyle.Renderer = New myColumnHeaderRenderer
    fpSpread1.Sheets(0).Columns(0).AllowAutoSort = True
    fpSpread1.Sheets(0).Columns(0).AllowAutoFilter = True
End Sub 'Form1_Load
```

Showing or Hiding Filter Indicators

You can display for the user or hide from the user the filter indicators that appear in the column headers.

Using Code

1. Set the **AllowAutoFilter** ('**AllowAutoFilter Property**' in the on-line documentation) property.
2. Set the **ShowFilterIndicator** ('**ShowFilterIndicator Property**' in the on-line documentation) property.

Example

This example enables the filtering on first column and hides the filter indicator.

C#

```
fpSpread1.Sheets[0].Columns[0].AllowAutoFilter = true;  
fpSpread1.Sheets[0].RowFilter.ShowFilterIndicator = false;
```

VB

```
fpSpread1.Sheets(0).Columns(0).AllowAutoFilter = True  
fpSpread1.Sheets(0).RowFilter.ShowFilterIndicator = False
```

Determining Which Header Row Displays the Indicators

If you have multiple column header rows then you can specify which row displays the filter indicators. By default they appear in the bottom column header row. You can specify which row displays the indicator by setting the **AutoFilterIndex** ('**AutoFilterIndex Property**' in the on-line documentation) property of the **ColumnHeader** ('**ColumnHeader Class**' in the on-line documentation) class.

Using Code

Set the ColumnHeader class **AutoFilterIndex** ('**AutoFilterIndex Property**' in the on-line documentation) property to the row index to display the indicator. The row index is zero based, so the first row is 0.

Example

The following example sets the indicator to appear in the next to the bottom column header row.

C#

```
fpSpread1.ActiveSheet.Columns[1].AllowAutoFilter = true;  
fpSpread1.ActiveSheet.ColumnHeader.RowCount = 3;  
fpSpread1.ActiveSheet.ColumnHeader.AutoFilterIndex = 1;
```

VB

```
fpSpread1.ActiveSheet.Columns(1).AllowAutoFilter = True  
fpSpread1.ActiveSheet.ColumnHeader.RowCount = 3  
fpSpread1.ActiveSheet.ColumnHeader.AutoFilterIndex = 1
```

Managing Cell Range Filtering

Spread for WinForms allows users to manage range filtering operations without any hassle via applying filters on a specific cell range in a worksheet. This makes it easier and quicker to find and view records matching to a specific filter condition in a cell range while working with bulk data in spreadsheets.

Managing cell range filtering includes the following tasks:

1. **Apply Range Filtering**

2. Access Filter Settings

Apply Range Filtering

You can apply filter on a cell range by using the **AutoFilter** method of **IRange** interface. If no filter is applied to a cell range, invoking the `AutoFilter` method will create a new empty filter. However, if a filter is already applied, invoking this method in the filtered range will remove the existing filter.

After applying filters on a cell range in a worksheet, a drop-down arrow indicator appears on the right side of each cell at the topmost row in the filtered range. Upon clicking the indicator, a drop-down list containing all the unique values available in the column will be displayed. Users can select the required filter options from the list in order to arrange the data.

Using Code

In order to filter a range of cells, you can use the **AutoFilter** method of **IRange** interface.

Example

Refer to the following example code to apply filter on a range of cells in a spreadsheet.

C#

```
// Apply range filter in cells B2:C5 using the AutoFilter method
fpSpread1.AsWorkbook().ActiveSheet.Cells["B2:C5"].AutoFilter();

// Apply range filter with custom filter condition
// Create a range filter - B2:E10 and filter Column C with AboveAverage filter
condition
fpSpread1.AsWorkbook().ActiveSheet.Cells["B2:E10"].AutoFilter(1,
GrapeCity.Spreadsheet.DynamicFilterType.AboveAverage);
```

VB

```
'Apply range filter in cells B2:C5 using the AutoFilter method
FpSpread1.AsWorkbook().ActiveSheet.Cells("B2:C5").AutoFilter()

'Apply range filter with custom filter condition
'Create a range filter - B2:E10 and filter Column C with AboveAverage filter condition
FpSpread1.AsWorkbook().ActiveSheet.Cells("B2:C10").AutoFilter(1,
GrapeCity.Spreadsheet.DynamicFilterType.AboveAverage)
```

Access Filter Settings

While working with range filters in a worksheet, you can access the current filter settings along with the existing table filter settings. This includes accessing information such as the cell range on which the filter is applied, the specified filter criteria and the filter mode.

Using Code

In order to access the existing filter settings and table filter settings, you can use the **AutoFilter** property.

Example

Refer to the following example code to access the current filter settings and the table filter settings in a worksheet.

C#

```
// Access current filter settings
```

```
IAutoFilter worksheetAutoFilter = fpSpread1.AsWorkbook().ActiveSheet.AutoFilter;
IFilter columnCFilter = worksheetAutoFilter.Filters[1];

// Access table filter settings
IAutoFilter tableAutoFilter =
fpSpread1.AsWorkbook().ActiveSheet.Tables["table1"].AutoFilter;
IFilter columnFilter = tableAutoFilter.Filters[0];
```

VB

```
'Access current filter settings
Dim worksheetAutoFilter As GrapeCity.Spreadsheet.IAutoFilter
worksheetAutoFilter = FpSpread1.AsWorkbook().ActiveSheet.AutoFilter
Dim columnCFilter As IFilter = worksheetAutoFilter.Filters(1)
'Access table filter settings
Dim tableAutoFilter As IAutoFilter =
FpSpread1.AsWorkbook().ActiveSheet.Tables("table1").AutoFilter
Dim columnFilter As IFilter = tableAutoFilter.Filters(0)
```

 **Note:** Users cannot use both column filters and range filters simultaneously in a worksheet at the same time. Also, users can filter only one cell range in a worksheet.

Managing Grouping of Rows of User Data

You can set the display of the spreadsheet component to allow rows to be grouped according to the column headers. You can customize the user experience for grouping data on a sheet. With grouping, you can allow the user to group rows of data according to the column headers that are dragged into the group bar. Special group headings are displayed above the grouped rows. Grouping of rows includes the following tasks.

- **Allowing the User to Group Rows**
- **Using Grouping**
- **Setting the Appearance of Grouped Rows**
- **Customizing the Group Bar**
- **Creating a Custom Group**
- **Interoperability of Grouping with Other Features**

Allowing the User to Group Rows

By default, the spreadsheet does not allow the user to group the rows of a spreadsheet. You can turn on this feature and allow grouping of rows for an entire sheet. Besides allowing grouping, you also need to allow columns to move, since the user performs grouping by clicking and dragging a column header into the group bar, which is similar to the act of moving a column. Also, the group bar must be visible and the column headers (at least one row) should be visible.

The following image displays the component with grouping allowed.

Drag a column here to group by that column.						
	A	B	C	D	E	F
1						
2						
3						
4						

Use the **AllowGroup** ('**AllowGroup Property**' in the on-line documentation) property of the sheet to turn on grouping. Use the **Visible** ('**Visible Property**' in the on-line documentation) property of the **GroupBarInfo** ('**GroupBarInfo Class**' in the on-line documentation) class to display the group bar (the area at the top of the sheet into which the user can drag column headers). Remember to set the **AllowColumnMove** ('**AllowColumnMove Property**' in the on-line documentation) property of the Spread to True to allow the user to click and drag column headers. Unless you are using the default value, set the **ColumnHeaderVisible** ('**ColumnHeaderVisible Property**' in the on-line documentation) property of the sheet to True to ensure that the column headers are displayed.

You can turn on or off the row headers; these have no effect on the display of grouping.

The **AllowDragDrop** ('**AllowDragDrop Property**' in the on-line documentation) property is not supported with grouping.

You can set the maximum number of levels of grouping that the end user can set. This limits the number of column headers that can be dragged consecutively to the group bar.

To understand how grouping works for the end user, refer to **Using Grouping**.

Using Code

1. Set the **AllowColumnMove** ('**AllowColumnMove Property**' in the on-line documentation) property to True.
2. Set the **Visible** ('**Visible Property**' in the on-line documentation) property to True.
3. Set the **AllowGroup** ('**AllowGroup Property**' in the on-line documentation) property to True to allow the user to group the data.

Example

This example allows grouping.

C#

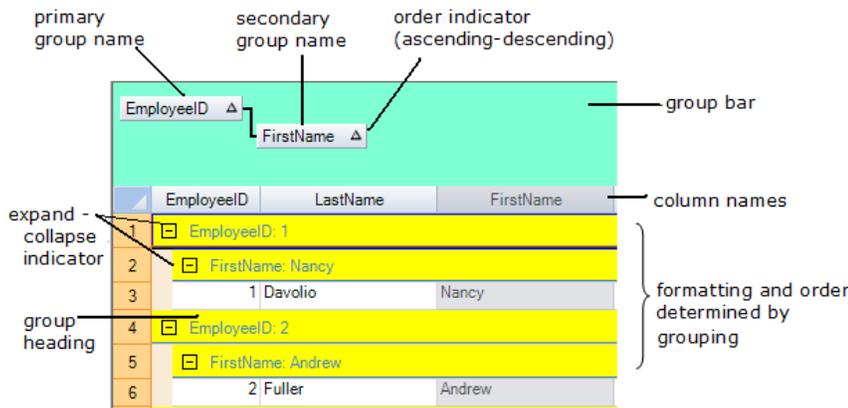
```
fpSpread1.AllowColumnMove = true;
fpSpread1.ActiveSheet.GroupBarInfo.Visible = true;
fpSpread1.ActiveSheet.AllowGroup = true;
```

VB

```
FpSpread1.AllowColumnMove = True
FpSpread1.ActiveSheet.GroupBarInfo.Visible = True
FpSpread1.ActiveSheet.AllowGroup = True
```

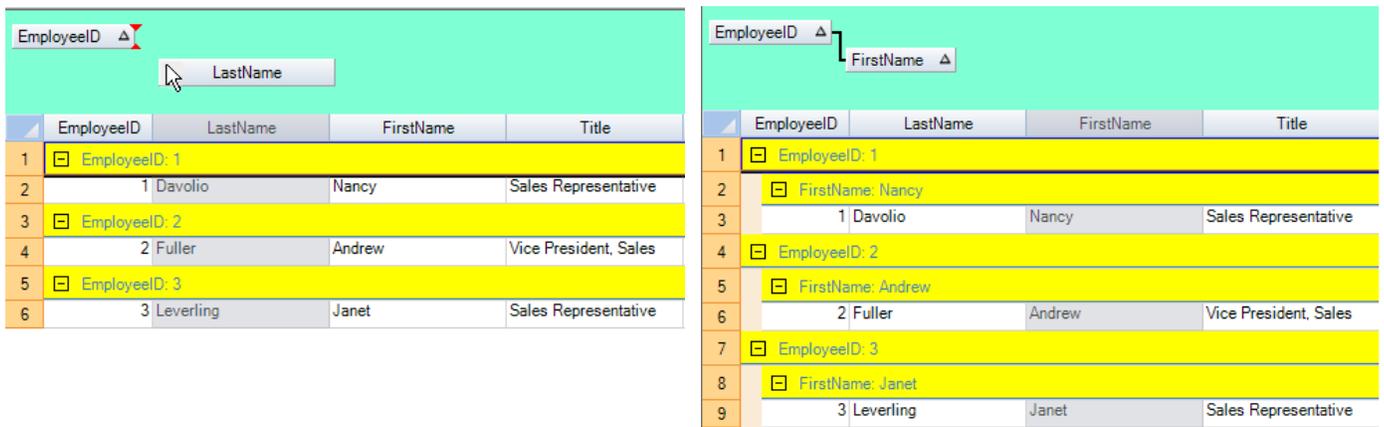
Using Grouping

You can set up the display to allow Outlook-style grouping of rows. For large amounts of data, this is helpful to display the data in the order the user needs. The user selects columns by which to sort, then organizes and displays the data in a hierarchy with rows organized accordingly. To select a column by which to group and display that data, either double-click on the header of that column or click and drag that column into the grouping bar at the top of the page. See the figure below for an example of the terms used with grouping.



You can expand or collapse groups by clicking the expand (+) or collapse (-) indicators.

You can provide grouping to allow users to sort the data with multiple levels of groups by dragging additional column headers into the grouping area. An example of the process of setting up two levels of grouping is shown in the following figure.



Before secondary grouping: dragging the column header into the grouping bar.

After secondary grouping: now a second level of hierarchy is shown.

When more than one level is chosen, the higher level is called the parent group and the lower level is called the child group. In the preceding figure with secondary grouping, the Employee ID is the parent group and the First Name is the child group.

Setting the Appearance of Grouped Rows

You can customize the appearance of the group headers and the grouped rows. For an introduction to the user interface for grouping, refer to **Using Grouping**.

You can set up the display so that the items are shown initially all expanded or all collapsed when grouping is performed. The **GroupingPolicy** ('**GroupingPolicy Property**' in the on-line documentation) property only applies to new groups.

You can set the colors and other formatting of both the hierarchy names and the data in the rows when grouping is performed.

You can hide or display the grouping bar at the top of the sheet.

The following table describes the members used for customizing the appearance of grouped rows:

Grouping API Member	Description
---------------------	-------------

GroupInfo ('GroupInfo Class' in the on-line documentation)

Class that represents grouping information

GroupInfoCollection ('GroupInfoCollection Class' in the on-line documentation)

Collection of grouping information

For more information on other hierarchical displays of data, refer to **Working with Hierarchical Data Display**.

You can also define a set of properties in an array list called GroupInfo. Set the appearance of grouped rows by adding styles to the array list of appearance properties for grouping. A collection of GroupInfo objects is in the GroupInfoCollection. To set the appearance settings in a GroupInfo to a particular sheet, set the **GroupInfos** property on that sheet. Appearance settings for grouping include:

- Background color
- Border
- Font
- Foreground (text) color
- Horizontal alignment
- Indent
- Indent color
- Vertical alignment

Example

This example sets the different appearance for group headers in primary and secondary groups. The appearance of group headers remains same from secondary group onwards.

C#

```
fpSpread1.AllowColumnMove = true;
fpSpread1.ActiveSheet.GroupBarInfo.Visible = true;
fpSpread1.ActiveSheet.GroupBarInfo.BackColor = Color.Aquamarine;
fpSpread1.ActiveSheet.GroupBarInfo.Height = 75;
fpSpread1.ActiveSheet.GroupMaximumLevel = 3;
fpSpread1.ActiveSheet.GroupBarInfo.GroupVerticalIndent = 20;
fpSpread1.ActiveSheet.AllowGroup = true;
FarPoint.Win.Spread.GroupInfo gi = new FarPoint.Win.Spread.GroupInfo();
gi.BackColor = Color.Yellow;
FarPoint.Win.Spread.GroupInfo gi2 = new FarPoint.Win.Spread.GroupInfo();
gi2.BackColor = Color.Green;
FarPoint.Win.Spread.GroupInfoCollection gic = new
FarPoint.Win.Spread.GroupInfoCollection();
gic.AddRange(new FarPoint.Win.Spread.GroupInfo[] { gi, gi2 });
fpSpread1.ActiveSheet.GroupInfos = gic;
```

VB

```
FpSpread1.AllowColumnMove = True
FpSpread1.ActiveSheet.GroupBarInfo.Visible = True
FpSpread1.ActiveSheet.GroupBarInfo.BackColor = Color.Aquamarine
FpSpread1.ActiveSheet.GroupBarInfo.Height = 75
FpSpread1.ActiveSheet.GroupMaximumLevel = 3
FpSpread1.ActiveSheet.GroupBarInfo.GroupVerticalIndent = 20
FpSpread1.ActiveSheet.AllowGroup = True
Dim gi As New FarPoint.Win.Spread.GroupInfo()
gi.BackColor = Color.Yellow
Dim gi2 As New FarPoint.Win.Spread.GroupInfo()
gi2.BackColor = Color.Green
```

```
Dim gic As New FarPoint.Win.Spread.GroupInfoCollection()
gic.AddRange(New FarPoint.Win.Spread.GroupInfo() {gi, gi2})
FpSpread1.ActiveSheet.GroupInfos = gic
```

 **Note:**

- Only column and sheet appearance settings remain when grouping is turned on. Since rows and cells are moved when the grouping feature is turned on, any style or span settings are ignored.
- For more information about the group data model and the effect on the sheet data model, refer to **Creating a Custom Group**.

Customizing the Group Bar

You can customize the appearance of the group bar at the top of the grouping display.

You can hide or display the grouping bar at the top of the sheet. The properties on the sheet (**GroupBarInfo** ('GroupBarInfo Property' in the on-line documentation) object) include:

GroupBarInfo Property	Description
BackColor ('BackColor Property' in the on-line documentation)	Set the background color of the grouping bar
Height ('Height Property' in the on-line documentation)	Set the height of the grouping bar
Visible ('Visible Property' in the on-line documentation)	Set whether to display the grouping bar
GroupVerticalIndent ('GroupVerticalIndent Property' in the on-line documentation)	Set the vertical distance between group names (when more than one group name is used) in the grouping bar

You can set the maximum levels of grouping allowed on the sheet by setting the SheetView object's **GroupMaximumLevel** ('GroupMaximumLevel Property' in the on-line documentation) property.

Creating a Custom Group

When grouping is turned on for a sheet, a separate target group data model is available to the sheet (or spreadsheet component) and this group data model is flat, completely without a hierarchy. This contains the group headers and other grouping-specific display data. Underneath that model is a target data model where the row data resides.

The following table describes the members used for grouping:

Grouping API Member	Description
IGroupSupport ('IGroupSupport Interface' in the on-line documentation) interface	Interface that supports grouping
GroupDataModel ('GroupDataModel Class' in the on-line documentation) class	Class of grouping data in the underlying models
Group ('Group Class' in the on-line documentation) class	Class in the underlying models that supports grouping
Grouping ('Grouping Event' in the on-line documentation) and Grouped ('Grouped Event' in the on-line documentation) events	Events in FpSpread class
GroupInfo ('GroupInfo Class' in the on-line documentation)	Class that represents grouping information

GroupInfoCollection ('GroupInfoCollection Class' in the on-line documentation)

Collection of grouping information

Using a Group Data Model

In the grouped sheet, you can reference the **GroupDataModel ('GroupDataModel Class' in the on-line documentation)** object that represents the grouped data in the **Data ('Data Property' in the on-line documentation)** property of the **SheetView.DocumentModels ('SheetView.DocumentModels Class' in the on-line documentation)** object referenced in the **Models ('Models Property' in the on-line documentation)** property of the **SheetView ('SheetView Class' in the on-line documentation)** class. You can use the **IsGroup ('IsGroup Method' in the on-line documentation)** method of the GroupDataModel class to determine whether the specified row is a group header. You can also use the **GetGroup ('GetGroup Method' in the on-line documentation)** method to reference a **Group ('Group Class' in the on-line documentation)** object that represents the group to which the specified row belongs.

Example

The following example references the GroupDataModel object on sheet and the each group.

C#

```
private void button1_Click(object sender, EventArgs e)
{
    if (fpSpread1.ActiveSheet.Models.Data is
        FarPoint.Win.Spread.Model.GroupDataModel)
    {
        FarPoint.Win.Spread.Model.GroupDataModel gdm =
            (FarPoint.Win.Spread.Model.GroupDataModel) fpSpread1.ActiveSheet.Models.Data;
        for (int i = 0; i < gdm.RowCount; i++)
        {
            // Determine whether it is group header row or not
            if (gdm.IsGroup(i))
            {
                // Get group
                var g = gdm.GetGroup(i);
                // Get the reference column of the group and the number of rows
                // included in the group
                Console.WriteLine(string.Format("Column index: {0}, Row count:
                {1}", g.Column, g.Rows.Count));
            }
        }
    }
}
```

Visual Basic

```
Private Sub button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    If TypeOf FpSpread1.ActiveSheet.Models.Data Is
        FarPoint.Win.Spread.Model.GroupDataModel Then
        Dim gdm As FarPoint.Win.Spread.Model.GroupDataModel =
            DirectCast(FpSpread1.ActiveSheet.Models.Data,
            FarPoint.Win.Spread.Model.GroupDataModel)
        For i As Integer = 0 To gdm.RowCount - 1
            'Determine whether it is group header row or not
            If gdm.IsGroup(i) Then
```

```

        ' Get group
        Dim g = gdm.GetGroup(i)
        ' Get the reference column of the group and the number of rows
included in the group
        Console.WriteLine(String.Format("Column index: {0}, Row count: {1}",
g.Column, g.Rows.Count))
        End If
    Next
End If
End Sub

```

Creating a Custom Group

You can customize grouping by specifying your own comparer. For example, you can create a custom group that is by decade if the column has year information. As the **Grouping ('Grouping Event' in the on-line documentation)** event is raised, you can pass in your own **IComparer** (call it **MyComparer**, for example). You can determine what is displayed in the group header by setting the **Text** property for that group.

Example

The following example creates its own comparer and converts the date value to a number representing the year in decade for comparison.

C#

```

[Serializable()]
public class MyGroupComparer : System.Collections.IComparer
{
    public int Compare(object x1, object y1)
    {
        int x=0, y=0;
        if ((x1) is DateTime)
        {
            x = ((DateTime) (x1)).Year % 10;
            x = ((DateTime) (x1)).Year - x;
        }
        if ((y1) is DateTime)
        {
            y = ((DateTime) (y1)).Year % 10;
            y = ((DateTime) (y1)).Year - y;
        }
        if (x == y) return 0;
        else if (x > y) return 1;
        else return -1;
    }
}
// Write the following code in the code behind of the form
private void Form1_Load(object sender, EventArgs e)
{
    // Enable grouping
    fpSpread1.AllowColumnMove = true;
    fpSpread1.ActiveSheet.GroupBarInfo.Visible = true;
    fpSpread1.ActiveSheet.AllowGroup = true;
    // Set test data
    fpSpread1.ActiveSheet.SetValue(0, 0, DateTime.Today);
    fpSpread1.ActiveSheet.SetValue(1, 0, DateTime.Today.AddYears(1));
}

```

```

        fpSpread1.ActiveSheet.SetValue(2, 0, DateTime.Today.AddYears(11));
    }
    private void fpSpread1_Grouping(object sender, FarPoint.Win.Spread.GroupingEventArgs
e)
    {
        var colIndex = e.SortInfo.Last().Index;
        // Generate custom group in first column
        if (colIndex == 0)
        {
            e.GroupComparer = new MyGroupComparer();
        }
    }
}

```

Visual Basic

```

<Serializable>
Public Class MyGroupComparer
    Implements System.Collections.IComparer
    Public Function Compare(x1 As Object, y1 As Object) As Integer Implements
IComparer.Compare
        Dim x As Integer = 0, y As Integer = 0
        If TypeOf (x1) Is DateTime Then
            x = DirectCast(x1, DateTime).Year Mod 10
            x = DirectCast(x1, DateTime).Year - x
        End If
        If TypeOf (y1) Is DateTime Then
            y = DirectCast(y1, DateTime).Year Mod 10
            y = DirectCast(y1, DateTime).Year - y
        End If
        If x = y Then
            Return 0
        ElseIf x > y Then
            Return 1
        Else
            Return -1
        End If
    End Function
End Class
' Write the following code in the code behind of the form
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    ' Enable grouping
    FpSpread1.AllowColumnMove = True
    FpSpread1.ActiveSheet.GroupBarInfo.Visible = True
    FpSpread1.ActiveSheet.AllowGroup = True
    ' Set test data
    FpSpread1.ActiveSheet.SetValue(0, 0, DateTime.Today)
    FpSpread1.ActiveSheet.SetValue(1, 0, DateTime.Today.AddYears(1))
    FpSpread1.ActiveSheet.SetValue(2, 0, DateTime.Today.AddYears(11))
End Sub
Private Sub FpSpread1_Grouping(sender As Object, e As GroupingEventArgs) Handles
FpSpread1.Grouping
    Dim colIndex = e.SortInfo.Last().Index
    ' Generate custom group in first column
    If colIndex = 0 Then
        e.GroupComparer = New MyGroupComparer()
    End If
End Sub
End Sub

```

Interoperability of Grouping with Other Features

The grouping feature affects the display and is not intended to work with some other features of Spread that also work with the display of the spreadsheet. When grouping happens, the data model is changed and a new model (the **GroupDataModel ('GroupDataModel Class' in the on-line documentation)**) is used. Many features are not affected by grouping at all, but some features, listed below, are not intended to operate with grouping. In general, if the feature involves the appearance or interactivity of the sheet or column, check the list to see if it is affected by grouping.

Some formatting features can work with grouping, but need to be applied after grouping occurs. If you need to format cells (colors, locked, and so on), you must apply the formatting after grouping.

After grouping rows, you should not change the column count and row count. The **GroupDataModel** does not support changing the column or row count. To add or remove columns or rows, you need to call the original data model methods. You can access the original data model using the **TargetModel ('TargetModel Property' in the on-line documentation)** property of the **GroupDataModel ('GroupDataModel Class' in the on-line documentation)** class.

The following features can work with grouping or are not affected by grouping:

- Grouping and hidden columns work together.
- The grouping feature affects only the visual display. Such things as export and printing are not affected.
- Grouping and input maps or action maps work together.

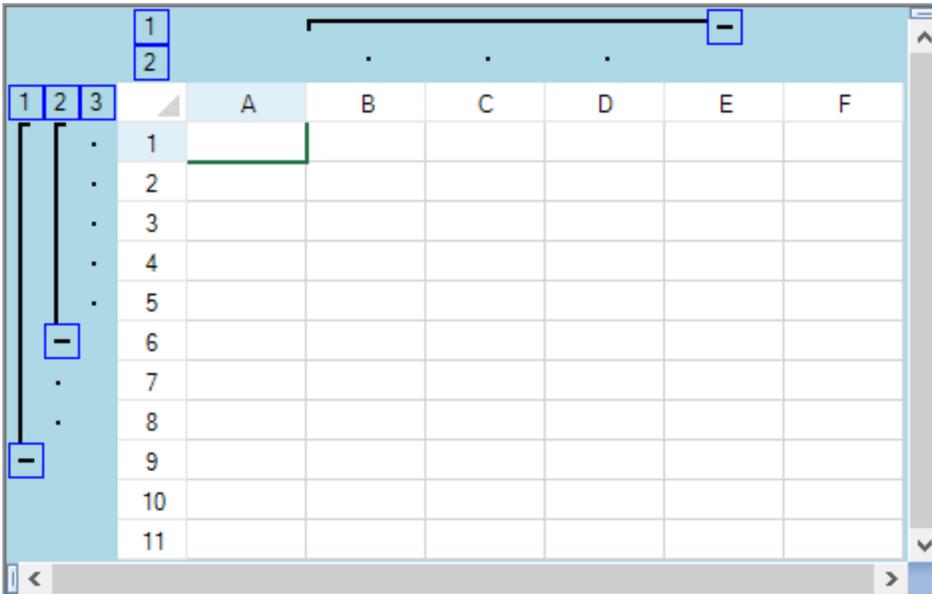
Features That Do Not Interoperate with Grouping

These features do not interoperate with grouping in Spread.

Feature	Description
Alternating Rows	Grouping and alternating rows do not work together. Grouping changes the order of the rows, so having a display of alternating rows would not make sense. Thus, these features do not work together.
Clipboard Paste	Pasting does not work with grouping.
Conditional Formatting	Grouping and conditional formatting do not work together. Conditional formatting requires the default data model. Thus, these features do not work together.
Filtering	Grouping and filtering do not work together. If you want to use grouping, you should not use filtering and you should clear the filter under the Grouping ('Grouping Event' in the on-line documentation) event.
Formulas	Grouping and formulas do not work together. Formulas requires the default data model. Thus, these features do not work with grouping.
Outlines	Grouping (Outlook style) and outlines (range groups) are not intended to work together.
Sorting	Grouping and sorting do not work together. Grouping is a type of sorting. When grouping is on, clicking on column headers will cause grouping not sorting. Thus, these features do not work together.
Hierarchical Data Display	Grouping and hierarchical display do not work together.
Error icons	Error icons do not work with grouping.

Managing Outlines (Range Groups) of Rows and Columns

You can set the display of the spreadsheet component to allow rows or columns to be grouped as an outline according to the headers. This displays a separate area beyond the headers that contains outlines to allow expanding or collapsing levels of rows or columns. The figure below shows three levels of outline for rows and two levels of outline for columns.



Collapsed rows that are visible are still visible when expanding the outline again. This behavior of the outline is similar to other spreadsheet programs with some subtle differences. This feature is also called "range grouping" since it operates on a range of rows or columns.

The following options are available to group rows and columns into outlines.

- **Using an Outline (Range Group) of Rows or Columns**
- **Customizing the Appearance of an Outline (Range Group)**

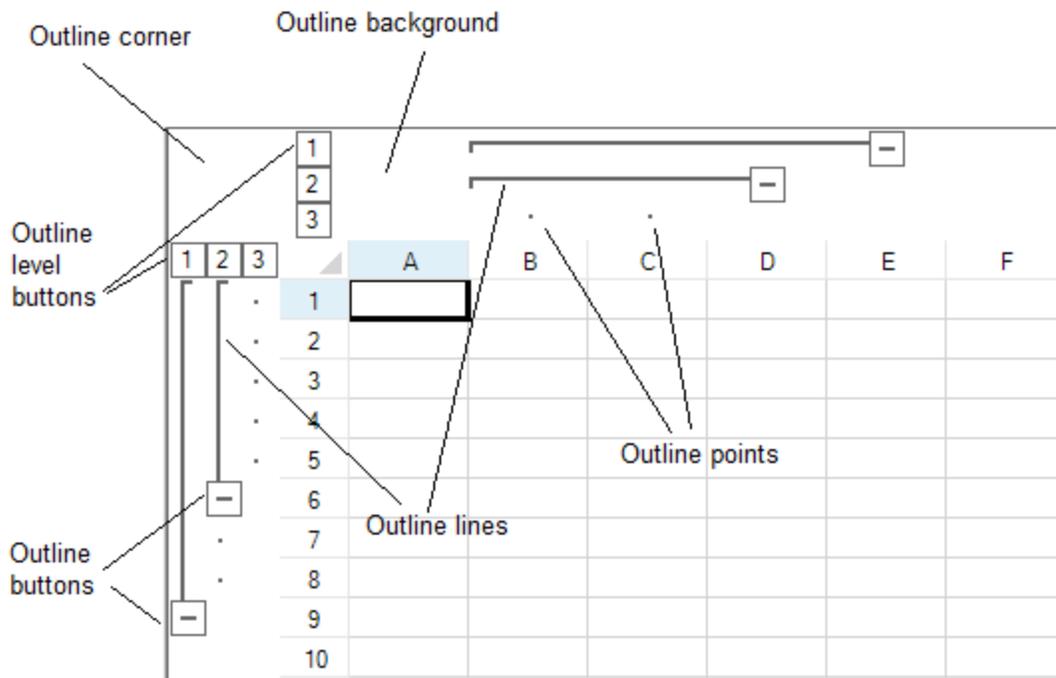
Since outlines affect the performance of other features, be sure to read **Interoperability of Outlines with Other Features**.

Using an Outline (Range Group) of Rows or Columns

You can form outlines of one or more rows or columns. There are several methods that create an outline (range group) such as the **AddRangeGroup ('AddRangeGroup Method' in the on-line documentation)** method for the **SheetView ('SheetView Class' in the on-line documentation)** class.

The outline appears at the left (for rows) and top (for columns) of the spreadsheet beyond the headers. Outlines can be nested, creating levels of outlines. The numbered boxes that appear in the outline area allow you to expand or collapse all the outlines of that level. You can expand and collapse rows and columns by clicking on the expand and collapse icons or on the numbered outline headers.

The figure below shows three levels of outline for rows and columns, and shows the terminology of the parts of the outline area.



Using Code

1. Set the **InterfaceRenderer** ('**InterfaceRenderer Property**' in the on-line documentation) property to change the default style.
2. Set the **RangeGroupBackgroundColor** ('**RangeGroupBackgroundColor Property**' in the on-line documentation) property to specify the outline background.
3. Set the **RangeGroupButtonStyle** ('**RangeGroupButtonStyle Property**' in the on-line documentation) property to specify the button style.
4. Use **AddRangeGroup** ('**AddRangeGroup Method**' in the on-line documentation) to add outlines.

Example

This example creates two column outline groups.

C#

```
fpSpread1.ActiveSheet.Rows.Count = 11;
fpSpread1.ActiveSheet.Columns.Count = 6;
fpSpread1.InterfaceRenderer = null;
fpSpread1.ActiveSheet.RangeGroupBackgroundColor = Color.LightGreen;
fpSpread1.ActiveSheet.RangeGroupButtonStyle =
FarPoint.Win.Spread.RangeGroupButtonStyle.Enhanced;
fpSpread1.ActiveSheet.AddRangeGroup(0, 8, true);
fpSpread1.ActiveSheet.AddRangeGroup(0, 5, true);
fpSpread1.ActiveSheet.AddRangeGroup(1, 3, false);
fpSpread1.ActiveSheet.AddRangeGroup(1, 2, false);
```

VB

```
fpSpread1.ActiveSheet.Rows.Count = 11
fpSpread1.ActiveSheet.Columns.Count = 6
```

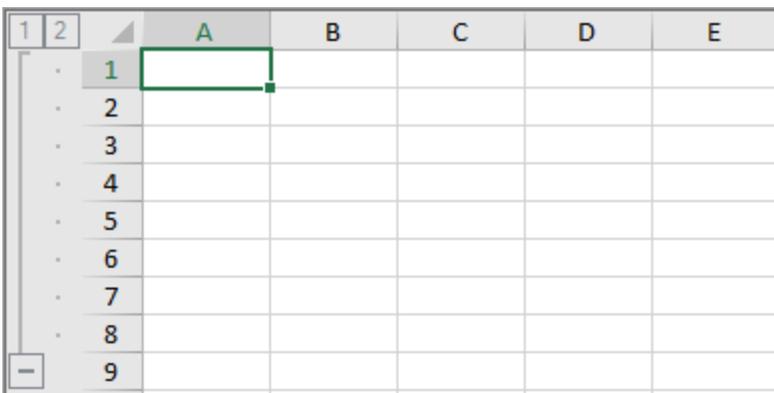
```

fpSpread1.InterfaceRenderer = Nothing
fpSpread1.ActiveSheet.RangeGroupBackgroundColor = Color.LightGreen
fpSpread1.ActiveSheet.RangeGroupButtonStyle =
FarPoint.Win.Spread.RangeGroupButtonStyle.Enhanced
fpSpread1.ActiveSheet.AddRangeGroup(0, 8, True)
fpSpread1.ActiveSheet.AddRangeGroup(0, 5, True)
fpSpread1.ActiveSheet.AddRangeGroup(1, 3, False)
fpSpread1.ActiveSheet.AddRangeGroup(1, 2, False)

```

You can hide the outline area by using the **ShowOutline ('ShowOutline Property' in the on-line documentation)** property of the **FpSpread ('FpSpread Class' in the on-line documentation)** class. It accepts **RowCol ('RowCol Enumeration' in the on-line documentation)** enumeration values as Both (Default), Rows, Columns or None.

The following example shows the rows outline and hides the columns outline.



C#

```

fpSpread1.ActiveSheet.AddRangeGroup(0, 8, true);
fpSpread1.ActiveSheet.AddRangeGroup(1, 3, false);
// Show only rows outline
fpSpread1.ShowOutline = FarPoint.Win.Spread.RowCol.Rows;

```

Visual Basic

```

fpSpread1.ActiveSheet.AddRangeGroup(0, 8, true)
fpSpread1.ActiveSheet.AddRangeGroup(1, 3, false)
'Show only rows outline
fpSpread1.ShowOutline = FarPoint.Win.Spread.RowCol.Rows

```

Customizing the Appearance of an Outline (Range Group)

You can customize the appearance of the outline (range group) using properties on the **SheetView ('SheetView Class' in the on-line documentation)** class or in an interface renderer.

The two properties in the **SheetView** class that can be used to customize the appearance are:

- **RangeGroupBackgroundColor ('RangeGroupBackgroundColor Property' in the on-line documentation)**
- **RangeGroupButtonStyle ('RangeGroupButtonStyle Property' in the on-line documentation)**

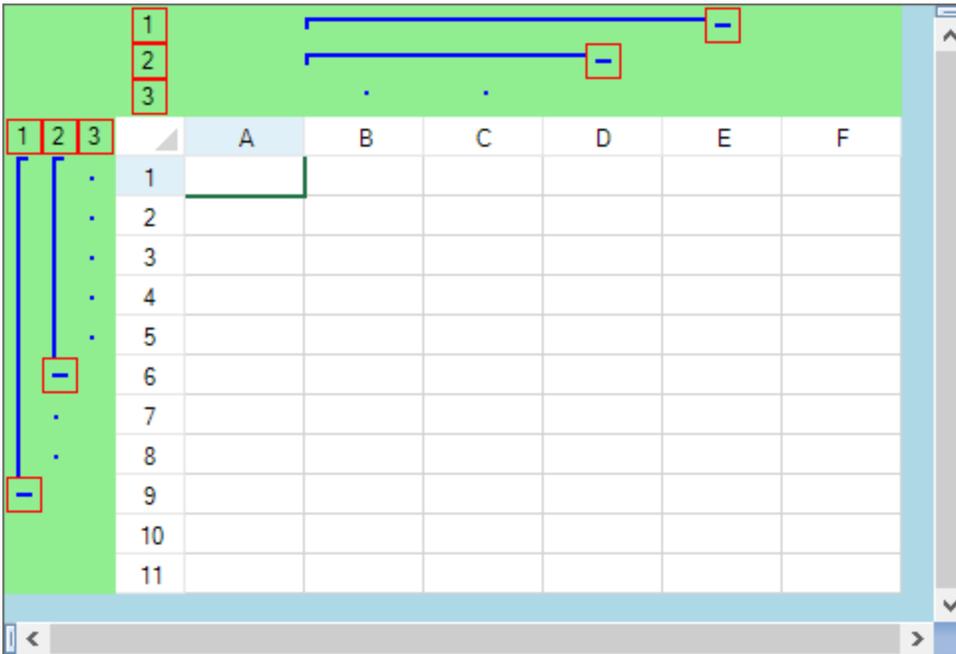
You can also use an **EnhancedInterfaceRenderer** to customize the appearance. The properties include:

- **RangeGroupBackgroundColor ('RangeGroupBackgroundColor Property' in the on-line documentation)**

documentation)

- **RangeGroupButtonBorderColor** ('RangeGroupButtonBorderColor Property' in the on-line documentation)
- **RangeGroupLineColor** ('RangeGroupLineColor Property' in the on-line documentation)

The line color also sets the color of the points in the outline. The following figure shows the results of the example code below where several of these properties are set.



Notice that the outline background is different from the gray area of the spreadsheet.

Using Code

1. Create a new interface renderer to provide a custom look to outlines.
2. Set the **RangeGroupBackgroundColor** ('RangeGroupBackgroundColor Property' in the on-line documentation) to specify the color.
3. Set the **RangeGroupButtonBorderColor** ('RangeGroupButtonBorderColor Property' in the on-line documentation) to specify the color for the button border.
4. Set the **RangeGroupLineColor** ('RangeGroupLineColor Property' in the on-line documentation) to specify the group line color.
5. Add the range groups with the **AddRangeGroup** ('AddRangeGroup Method' in the on-line documentation) method.

Example

This example creates an outline in the rows and in the columns and changes various colors. The result is shown in the preceding figure.

C#

```
fpSpread1.ActiveSheet.Rows.Count = 11;
fpSpread1.ActiveSheet.Columns.Count = 6;
FarPoint.Win.Spread.EnhancedInterfaceRenderer outlineLook = new
FarPoint.Win.Spread.EnhancedInterfaceRenderer();
outlineLook.RangeGroupBackgroundColor = Color.LightGreen;
```

```

outlinelook.RangeGroupButtonBorderColor = Color.Red;
outlinelook.RangeGroupLineColor = Color.Blue;
fpSpread1.InterfaceRenderer = outlinelook;
fpSpread1.ActiveSheet.AddRangeGroup(0, 8, true);
fpSpread1.ActiveSheet.AddRangeGroup(0, 5, true);
fpSpread1.ActiveSheet.AddRangeGroup(1, 3, false);
fpSpread1.ActiveSheet.AddRangeGroup(1, 2, false);

```

VB

```

fpSpread1.ActiveSheet.Rows.Count = 11
fpSpread1.ActiveSheet.Columns.Count = 6
Dim outlinelook As New FarPoint.Win.Spread.EnhancedInterfaceRenderer
outlinelook.RangeGroupBackgroundColor = Color.LightGreen
outlinelook.RangeGroupButtonBorderColor = Color.Red
outlinelook.RangeGroupLineColor = Color.Blue
fpSpread1.InterfaceRenderer = outlinelook
fpSpread1.ActiveSheet.AddRangeGroup(0, 8, True)
fpSpread1.ActiveSheet.AddRangeGroup(0, 5, True)
fpSpread1.ActiveSheet.AddRangeGroup(1, 3, False)
fpSpread1.ActiveSheet.AddRangeGroup(1, 2, False)

```

Interoperability of Outlines with Other Features

The outline (range group) feature affects the display and is not intended to work with some other features of Spread that also work with the display of the spreadsheet. Many features are not affected by outlines at all, but some features, listed below, are not intended to operate with this feature. In general, if the feature involves the appearance or interactivity of the sheet or column, check the list to see if it is affected by outlines.

Be careful when adding rows or columns to a display that has an outline.

The following features can work with grouping or are not affected by grouping:

- Outlines and hidden columns work together.
- The outline feature affects only the visual display. Such things as export and printing are not affected.
- Outlines and input maps or action maps work together.

Features That Do Not Interoperate with Outlines

These features do not interoperate with outlines in Spread.

Feature	Description
Alternating Rows	Outlines and alternating rows may not work together. Outlines changes the order of the rows, so having a display of alternating rows would not make sense. Thus, these features do not work together.
Hierarchical Display	Outlines and hierarchies are two different and dissimilar ways of grouping and displaying data. These features are not intended to work together.
Outlook-style Grouping	Outlines and Outlook-style grouping are two different and dissimilar ways of grouping and displaying data. These features are not intended to work together.
Sorting	Outlines and sorting might not work together.

Customizing User Searching of Data

You can search for data in any of the cells in the workbook by specifying the sheet and the string of data for which to search. You can also have the component display a search dialog and allow the end user to search for data. The methods and properties that relate to searching and search dialogs are part of the Spread component.

The tasks for searching include:

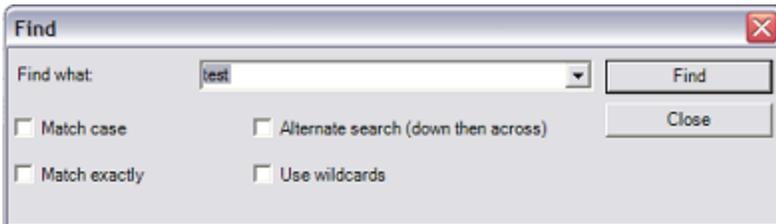
- **Allowing the User to Perform a Standard Search**
- **Allowing the User to Perform an Advanced Search**
- **Searching for Data with Code**

 **Note:**

- There are limitations to the search. Row and column headers are not searched with the search dialogs; only cells are searched.
- Use the **SearchHeaders ('SearchHeaders Method' in the on-line documentation)** method to search the headers.
- Not all tags are included when you search and "include tags"; only cell tags are included.
- None of the information in the sheet, column, or row object is included in the search.

Allowing the User to Perform a Standard Search

You can have the component display a search (find) dialog for the end-user to allow them to search the text (unformatted data) of cells in a sheet for a particular string of text, as shown in the following figure.



You can customize many features of the search dialog box by setting its properties. In addition, you can display a default search string in the **Find what** combo box. And you can set the check boxes for these options:

- **Match case** - finding only strings that match the case of the search string (upper or lower case).
- **Match exactly** - finding only strings that match the search string exactly.
- **Alternate search** - searching down rows then across columns rather than vice versa.
- **Use wildcards** - allow the use of wildcard characters in the search string.

For information about the advanced options available on the search dialog, refer to **Allowing the User to Perform an Advanced Search**.

Using Code

Use the **SearchWithDialog ('SearchWithDialog Method' in the on-line documentation)** methods for the **FpSpread ('FpSpread Class' in the on-line documentation)** component to customize the search dialog.

Example

This example provides a search dialog with several settings preset. In this case, it as an exact-match search on the fourth sheet (Sheet 3) for the phrase "Not Available" and start at the first row and column.

C#

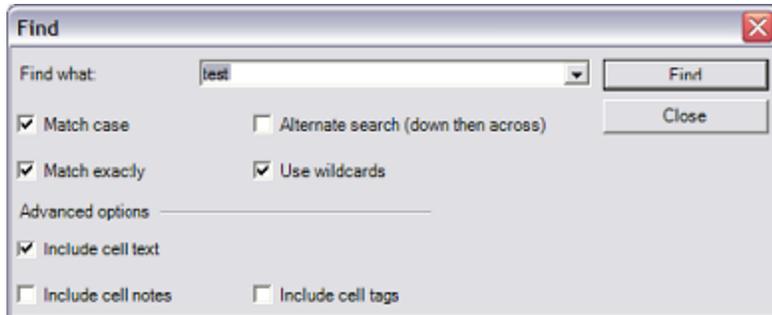
```
fpSpread1.SearchWithDialog(3, "Not Available", true, true, false, false, 0, 0);
```

VB

```
FpSpread1.SearchWithDialog(3, "Not Available", True, True, False, False, 0, 0)
```

Allowing the User to Perform an Advanced Search

You can provide a more advanced search dialog for the end-users to allow them to search other areas of the spreadsheet, including cell notes and cell tags.



There are several advanced options you can set that extend the scope of the search. The advanced options include:

- Include cell text- searches the row and column cells.
- Include cell tags- searches the cell tags in the data area.
- Include cell notes- searches the cell notes in the data area.

For information about performing a search with the search dialog with the standard options, refer to **Allowing the User to Perform a Standard Search**.

Using Code

Use the **SearchWithDialogAdvanced** ('**SearchWithDialogAdvanced Method**' in the on-line **documentation**) methods for the **FpSpread** ('**FpSpread Class**' in the on-line **documentation**) component to customize the advanced search dialog.

Example

This example uses the **SearchWithDialogAdvanced** ('**SearchWithDialogAdvanced Method**' in the on-line **documentation**) method and provides the users a search dialog with several settings preset.

C#

```
fpSpread1.SearchWithDialogAdvanced(0, 4, "This", true, true, false, false, 0, 0);
```

VB

```
fpSpread1.SearchWithDialogAdvanced(0, 4, "This", True, True, False, False, 0, 0)
```

Searching for Data with Code

To search for data in any of the cells of a sheet, use any of these sets of methods in the **FpSpread** ('**FpSpread Class**' in the on-line **documentation**) class:

- **Search** ('**Search Method**' in the on-line **documentation**) methods

- **SearchHeaders** ('**SearchHeaders Method**' in the on-line documentation) methods
- **SearchWithDialog** ('**SearchWithDialog Method**' in the on-line documentation) methods
- **SearchWithDialogAdvanced** ('**SearchWithDialogAdvanced Method**' in the on-line documentation) methods

The parameters of the various search methods allow you to specify the sheet to search, the string for which to search, and the matching criteria. For a list of qualifications (restrictions) of the search, refer to the set of methods listed above for more details.

Most searches (except for the method that specifies a block range of cells) start at the specified start cell and continue to the end of the row and then start the next row at the first cell. The search continues until either the end cell or the end of the sheet.

For information about the search dialogs, refer to **Allowing the User to Perform a Standard Search** and **Allowing the User to Perform an Advanced Search**.

Using Code

Use the **Search** ('**Search Method**' in the on-line documentation) method for the Spread component to perform a search.

Example

This example uses the **Search** ('**Search Method**' in the on-line documentation) method for the Spread component to perform an exact-match search on the third sheet (Sheet 2) for the word "Total" and return the values of the row index and column index of the found cell.

C#

```
fpSpread1.Search(2, "Total", true, true, false, false, 1, 1, 56, 56, ref rowindx, ref colindx);
```

VB

```
FpSpread1.Search(2, "Total", True, True, False, False, 1, 1, 56, 56, rowindx, colindx)
```

Formulas in Cells

You can set up and perform calculations using formulas in cells. This may involve these tasks:

- **Placing a Formula in Cells**
- **Specifying a Cell Reference in a Formula**
- **Specifying a Sheet Reference in a Formula**
- **Specifying an External Reference in a Formula**
- **Using a Circular Reference in a Formula**
- **Nesting Functions in a Formula**
- **Recalculating and Updating Formulas Automatically**
- **Finding a Value Using GoalSeek**
- **Allowing the User to Enter Formulas**
- **Creating and Using a Custom Name**
- **Creating and Using a Custom Function**
- **Creating and Using a Visual Function**
- **Creating and Using External Variable**
- **Using the Array Formula**
- **Working With Dynamic Array Formulas**
- **Working with the Formula Text Box**
- **Setting up the Name Box**
- **Using Language Package**
- **Accessing Data from Header or Footer**
- **Auto Format Formulas**
- **Managing External Reference**
- **Precedents and Dependents**

For more information on entering formulas using the Spread Designer, refer to **Entering a Formula in Spread Designer (on-line documentation)**.

For information on the floating formula bar, where users can type in formulas and select cells dynamically, refer to **Setting up the Formula Text Box**.

For an overview of formulas and details on the functions and operators that can be used to create a formula, refer to the [Formula Reference](#).

For information on the **CalcEngine** assembly, which is responsible for the formula calculations, refer to the **FarPoint.CalcEngine ('FarPoint.CalcEngine Assembly' in the on-line documentation)**.

Placing a Formula in Cells

You can add a formula to a cell or range of cells. You can also add a formula to all the cells in a row or column. The formula is a string with the expression of the formula, typically containing a combination of functions, operators, and constants.

When assigning a formula to the **Row ('Row Class' in the on-line documentation)** class or **Column ('Column Class' in the on-line documentation)** class, you are assigning a default formula for that row or column. In other words, the formula is used for every cell in the row or column (assuming that the formula is not overridden at the cell level). For a formula in a row or column, Spread uses the first cell in the row or column as the base location. The formula evaluates to a different result for each cell in column A if you use relative addressing. If you want each cell in column A to evaluate to the sum of the values in C2 and D2 (and not the value in the C and D columns for each row) then you would need to use the formula `C2+D2`, which uses absolute address. For examples of formulas that use cell references, refer to **Specifying a Cell Reference in a Formula**.

You can add a formula by specifying the **Formula** property for the object or by entering it in the Spread Designer. The procedures for using code are given below. For instructions on using Spread Designer to enter a formula, refer to **Entering a Formula in Spread Designer (on-line documentation)**. You can allow end users to enter formulas by allowing them to type the equals sign and then the formula; refer to **Allowing the User to Enter Formulas**.

Be careful of the type of cell in which the data is found, and whether you use the **Text** or **Value** property when assigning data that is used in a formula. When you assign cell data using the **Text** property, the spreadsheet uses the cell type to parse an assigned string into the needed data type. For example, a number cell type parses a string into a double data type. When you assign the cell data using the **Value** property, the spreadsheet accepts the assigned object as is and no parsing occurs, so if you set it with a string, it remains a string. Some numeric functions (for example, SUM) ignore non-numeric values in a cell range. For example, if the cell range A1:A3 contains the values {1, "2", 3}, then the formula SUM(A1:A3) evaluates to 4 because the SUM function ignores the string "2". Be sure that you set the value correctly for any cells used in the calculation of a formula and that you set them with the correct data type.

A string constant in a formula can contain special characters such as the new line character (that is, '\n'). Make sure that you enclose the string constant in quotes in the text representation of the formula. The following C# code creates a multiple-line text cell and assigns a formula that contains a string constant that contains a new line character.

C#

```
TextCellType ct = new TextCellType();
ct.Multiline = true;
fpspread1.Sheets[0].Cells[0,0].Formula = "\"line1\nline2\"";
```

Using a Shortcut

Add a formula to a cell, row, or column by specifying the **Formula** property for that cell, row, or column.

Example

This example shows how to specify a formula that finds the product of five times the value in the first cell, and puts the result in another cell. Then it finds the sum of a range of cells (A1 through A4) and puts the result in every cell of the fourth column.

C#

```
fpSpread1.ActiveSheet.Cells[2, 0].Formula = "PRODUCT(A1,5)";
fpSpread1.ActiveSheet.Columns[3].Formula = "SUM(A1:A4)";
```

VB

```
fpSpread1.ActiveSheet.Cells(2, 0).Formula = "PRODUCT(A1,5) "
fpSpread1.ActiveSheet.Columns(3).Formula = "SUM(A1:A4) "
```

Using Code

1. Specify the cell, row, or column.
2. Add a formula to the cell, row, or column.

Example

This example shows how to specify a formula that sums two cells, doubles it, and puts the result in a third cell.

C#

```
FarPoint.Win.Spread.Cell mycell;
mycell = fpSpread1.Cells[2, 0];
```

```
mycell.Formula = "SUM(A1:A2) * 2";
```

VB

```
Dim mycell As FarPoint.Win.Spread.Cell
mycell = fpSpread1.ActiveSheet.Cells(2, 0)
mycell.Formula = "SUM(A1:A2) * 2"
```

You can retain the suffix and prefix whitespace characters of the formula expression by using the **ReserveFormulaWhiteSpaces** ('CalcFeatures Enumeration' in the on-line documentation) enumeration. The whitespace characters can include many characters such as space, enter, new-line etc.

	A	B	C	D
1	1		22-10-2021 16:49:20	
2	1		+	2
3				
4				
5				
6				

Example

This example shows how to keep the whitespace characters in the expression.

C#

```
IWorkbook workbook = fpSpread1.AsWorkbook();
IWorksheet sheet1 = workbook.Worksheets[0];

sheet1.Cells["A1"].Value = 1;
sheet1.Cells["A2"].Value = 1;

fpSpread1.AsWorkbook().WorkbookSet.CalculationEngine.CalcFeatures |=
CalcFeatures.ReserveFormulaWhiteSpaces;
sheet1.Cells["C1"].Formula2 = "NOW( )";
sheet1.Cells["C2"].Formula2 = "SUM( Sheet1!A1:A2 )";
fpSpread1.AllowUserFormulas = true;
```

VB

```
Dim workbook As IWorkbook = fpSpread1.AsWorkbook()
Dim sheet1 As IWorksheet = workbook.Worksheets(0)

sheet1.Cells("A1").Value = 1
sheet1.Cells("A2").Value = 1

fpSpread1.AsWorkbook().WorkbookSet.CalculationEngine.CalcFeatures =
fpSpread1.AsWorkbook().WorkbookSet.CalculationEngine.CalcFeatures Or
CalcFeatures.ReserveFormulaWhiteSpaces
sheet1.Cells("C1").Formula2 = "NOW( )"
sheet1.Cells("C2").Formula2 = "SUM( Sheet1!A1:A2 )"
fpSpread1.AllowUserFormulas = True
```

Using the Formula Editor

At design time, you can enter formulas in cells using either the **Formula** bar or the **Formula Editor**, both of which are available from the Spread Designer. The **Formula Editor** is also available from the **Properties** Window. For more information, refer to **Entering a Formula in Spread Designer (on-line documentation)**. For details on the functions and operators that can be used to create a formula, refer to the [Formula Reference](#).

1. Select the sheet tab of the sheet that contains the cells in which to place formulas.
2. Select the cell or cells in the sheet.
3. In the Formula property, click the arrow button. This opens the **Formula Editor**.
4. In the **Formula Editor**, you may type in the formula in the edit box. To assist in entering functions in the formula, you can double-click on a function name to have it appear in the edit box. Functions are organized by category. You can also type operators and constants to construct your formula.
5. When done, click **Apply**. Click **OK** to close the editor.
6. If you were working from within the Spread Designer, from the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Displaying Formula in Cells

Spread for WinForms allows you to display the formula in a cell instead of its final calculated value. The **DisplayFormulas** property in **IWorksheetView** interface can be used to enable or disable the display of cell formulas. By default, it is set to false. The **DisplayFormulas** property is set at the worksheet level. Hence, the other worksheets are not impacted by the value of any particular sheet.

When **DisplayFormulas** is set to true, the column widths are doubled to display the long formulas in cells conveniently. However when set to false, the column widths are restored to their original width.

 **Note:** When **DisplayFormulas** property is enabled, text overflow and auto merge cells are disabled.

Example

This example code sets the DisplayFormulas property to true to display the formula in cell C1.

C#

```
// Set value in Cells
fpSpread1.ActiveSheet.Cells["A1"].Value = 10;
fpSpread1.ActiveSheet.Cells["B1"].Value = 20;
// Set formula in Cell
fpSpread1.ActiveSheet.Cells["C1"].Formula = "SUM(A1, B1)";

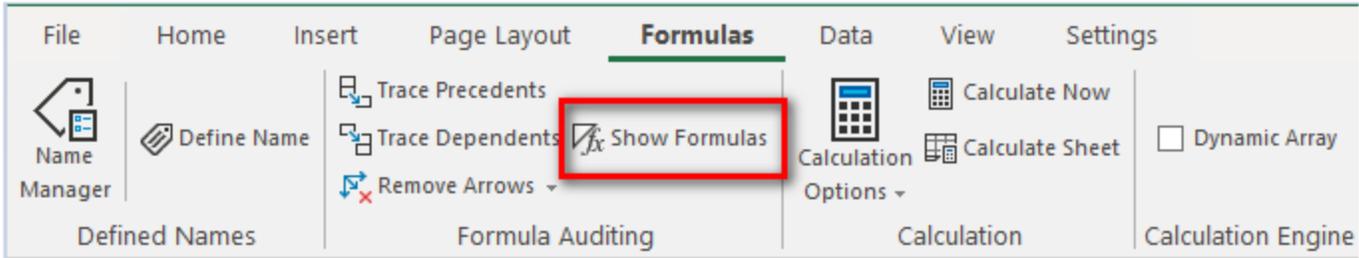
// Set DisplayFormula to true for the activeSheet
fpSpread1.AsWorkbook().ActiveSheet.View.DisplayFormulas = true;
```

VB

```
' Set value in Cells
fpSpread1.ActiveSheet.Cells("A1").Value = 10
fpSpread1.ActiveSheet.Cells("B1").Value = 20
' Set formula in Cell
fpSpread1.ActiveSheet.Cells("C1").Formula = "SUM(A1, B1)"
' Set DisplayFormula to true for the activeSheet
fpSpread1.AsWorkbook().ActiveSheet.View.DisplayFormulas = True
```

Using Spread Designer

To enable 'Show Formulas' option in Spread Designer, click on the 'Show Formulas' option in Formulas tab. The "Show Formulas" button is grayed when the mode is already enabled.



Specifying a Cell Reference in a Formula

Besides values, operators, and functions, a formula can contain references to values in other cells. For example, to find the sum of the values in two cells, the formula can refer to the cell coordinates by row and column. You can use an absolute cell reference (with the actual coordinates of the row and column) or a relative cell reference (with the coordinates relative to the current cell). You choose which type of cell reference for the sheet by using the **ReferenceStyle ('ReferenceStyle Property' in the on-line documentation)** property. For details on the way to specify the reference style, refer to the **ReferenceStyle ('ReferenceStyle Property' in the on-line documentation)** property of the **SheetView ('SheetView Class' in the on-line documentation)** class, and the **ReferenceStyle ('ReferenceStyle Enumeration' in the on-line documentation)** enumeration.

If you have changed the cell reference style to a style that cannot represent the formula, the Spread component provides the formula with question marks as placeholders for cell references that cannot be represented.

The following table contains examples of valid formulas using references:

Function	Description
SUM(A1:A10)	Sums rows 1 through 10 in the first column
PI()*C6	PI times the value in cell C6
(A1 + B1) * C1	Adds the values in the first two cells and multiplies the result by the value in the third cell
IF(A1>5, A1*2, A1*3)	If the contents of cell A1 are greater than 5, then multiply the contents of cell A1 by 2, else multiply the contents of cell A1 by 3

If you have defined relative cell references used in a formula in cell B1 as RC[-1]+R[-1]C, the formula is interpreted as add the value in the cell to the left (A1) to the value in the cell above ("Bo"). The component treats the value in the cell "Bo" as an empty cell. If you change the cell reference style to the A1 style, the formula becomes A1+B?, because the A1 style cannot represent cell "Bo". However, the component still evaluates the formula as it would using the R1C1 reference style.

 **Note:** Although most of Spread uses zero-based references to rows and columns, in the creation of formulas you must use one-based references. The column and row numbers start at one (1), not zero (0).

Range Reference

Spread supports range references where the start row and end row consists of same reference types (either both row coordinates are absolute or both row coordinates are relative) and different reference types (one coordinate is absolute, other coordinate is relative and vice a versa).

The following table describes some examples of range references in R1C1 (Number-Number) notation:

Reference	Whether Supported
-----------	-------------------

R1C[-1]:R5C[-1]	supported (absolute row : absolute row)
R1C[-1]:RC[-1]	supported (absolute row : relative row)
RC[-1]:R5C[-1]	supported (relative row : absolute row)
R[-5]C[-1]:RC[-1]	supported (relative row : relative row)

The following table describes some examples of range references with different reference types in A1 (Letter-Number) notation:

Reference	Whether Supported
A\$1:\$B1	supported (cell range starting with relative column plus one, absolute first row and ending with absolute second column, relative row plus one)
\$A1:B\$1	supported (cell range starting with absolute first column, relative row plus one and ending with relative column plus one, absolute first row)
A\$1:B\$1	supported (cell range starting with relative column plus one, absolute first row and ending with relative row plus one, absolute first column)

The following table describes some examples of special usage cases that are supported in range references in Spread:

Reference	Whether Supported
\$A:\$B	
A:B	supported (refers to whole column)
C1:C2	
C[-2]:C[-1]	
\$1:\$2	
1:2	supported (refers to whole row)
R1:R2	
R[-5]:R[-4]	

For more information on range reference styles, refer to the [Formula Reference](#).

Using the Properties Window

1. At design time, in the **Properties** window select the **Sheets** property and click on the button to open the **SheetView Collection** editor.
2. Select the sheet from the Member list.
3. In the properties list (in the **Calculation** category), select the **ReferenceStyle** property.
4. Click the drop-down arrow to display the choices and select the value, either **A1** or **R1C1**.

Using Code

Specify the reference style by setting the **ReferenceStyle** ('ReferenceStyle Property' in the on-line **documentation**) property or use the default ReferenceStyle value.

Example

This example sets the reference style.

C#

```
fpSpread1.Sheets[0].ReferenceStyle = FarPoint.Win.Spread.Model.ReferenceStyle.A1;  
fpSpread1.Sheets[0].Cells[2, 2].Formula = "SUM(A1:A6)";
```

VB

```
fpSpread1.Sheets(0).ReferenceStyle = FarPoint.Win.Spread.Model.ReferenceStyle.A1  
fpSpread1.Sheets(0).Cells(2, 2).Formula = "SUM(A1:A6) "
```

Using the Spread Designer

1. Select the sheet tab name for the sheet.
2. In the property list (in the **Calculation** category), select the **ReferenceStyle** property.
3. Click the drop-down arrow to display the choices and select the value, either **A1** or **R1C1**.
4. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Specifying a Sheet Reference in a Formula

A formula can contain references to other sheets. When a reference to a cell includes a reference to a cell on another sheet, this is called cross-sheet referencing. An example of cross-sheet referencing in a formula that uses the addition operator would be:

```
(FirstRoundData!A2 + SecondRoundData!A2)
```

 **Note:** Although most of Spread uses zero-based references to rows and columns, in the creation of formulas you must use one-based references. The column and row numbers start at one (1), not zero (0).

Another example would be keeping a running total of cells of one sheet on a separate sheet. Use the **Formula** property to put a formula on one sheet that references the cells you want added from another sheet, as shown in the following code.

```
fpSpread1.Sheets(1).Cells(0,0).Formula = "SUM(Sheet1!A1:Sheet1:A100) "
```

Then use the **ReferenceStyle ('ReferenceStyle Property' in the on-line documentation)** property to set the reference style.

You can have formulas that reference other worksheets or you can have automatic calculations at the worksheet level (applies to all sheets). You cannot have both. When **EnableCrossSheetReference ('EnableCrossSheetReference Property' in the on-line documentation)** is True (which is the default setting), the entire workbook acts as a single calculation unit with all worksheets sharing the same calculation settings (auto calculations, iterations, custom functions, custom names, etc). Changing a calculation setting affects all worksheets. Formulas can reference cells on other worksheets. When **EnableCrossSheetReference ('EnableCrossSheetReference Property' in the on-line documentation)** is False, it allows users to set formula reference to a cell in another sheet but the value of the cell will be #REF!.

If the sheet name contains non alpha-numeric characters (for example, a space), then enclose the sheet name in single quotes in the formula. For example, suppose sheet name is "page one" then the formula would be SUM('page one'!\$A\$1:\$A\$5).

If the sheet name contains the single quote character, then use two single quote characters in the formula. For example, suppose the sheet name is "scott's page" then the formula would be SUM('scott's page'!\$A\$1:\$A\$5).

If the sheet name contains a colon, then use two single quotes around the sheet name. For example ("Sheet:name'!\$B\$1:\$F\$1").

For more information on cross-sheet referencing, refer to the [Formula Reference](#).

Using Code

The following example uses default sheet names in a formula.

Example

This example sets the formula.

C#

```
fpSpread1.Sheets[0].Cells[0,0].Formula = "Sheet1!A3 + Sheet2!A2";
```

VB

```
fpSpread1.Sheets(0).Cells(0,0).Formula = "Sheet1!A3 + Sheet2!A2"
```

Specifying an External Reference in a Formula

You can use external references in formulas, such as data from an xlsx file.

The external reference uses the file name and sheet. For example: 'C:\\Mescius\\[Sum.xlsx]Sheet1'.

Using Code

The following example references data from a file.

C#

```
fpSpread1.Sheets[0].SetFormula(0, 0, "SUM('C:\\Program Files (x86)\\Mescius\\  
[Sum.xlsx]Sheet1'!A1:A4)");  
fpSpread1.AllowUserFormulas = true;
```

VB

```
FpSpread1.Sheets(0).SetFormula(0, 0, "SUM('C:\\Program Files (x86)\\Mescius\\  
[Sum.xlsx]Sheet1'!A1:A4)")  
FpSpread1.AllowUserFormulas = True
```

Using a Circular Reference in a Formula

You can refer to a formula in the cell that contains that formula; this type of reference is called a circular reference. This is done typically to recurse on a function to approach an optimum value by iterating on the same function.

This topic explains the following tasks:

1. **Iterative Calculations in a Formula**
2. **Locate Circular References in a Formula**

Iterative Calculations in a Formula

You can select how many times a function iterates on itself (recurses) by setting the recalculation iteration count property using the **MaximumIterations ('MaximumIterations Property' in the on-line documentation)** property. You can set the amount of change allowed with the **MaximumChange ('MaximumChange Property' in the on-line documentation)** property.

By default, if the formula "=COLUMNS(A1:C5)" is in cell C4, no result is returned. In other words, if both the last column and row index of the array are greater than the column and row index of the cell in which the formula resides, the formula cannot be calculated. In this case, the cell C4 is in the range A1:C5. This is a circular reference in a formula

and so Spread does not evaluate the formula unless iterations are turned on.

As with most spreadsheet products (including Excel and OpenOffice), Spread solves circular formulas via iterations. During each recalculation cycle, a specified number of iterations are performed. During each iteration, every circular formula is evaluated exactly once.

For information on using the **Formula Editor** to enter a formula at design time, refer to **Entering a Formula in Spread Designer (on-line documentation)**. For details on the functions and operators that can be used to create a formula, refer to the [Formula Reference](#).

Using Code

1. Set the cell types for the cells with the formulas.
2. Set the recalculation iteration count by setting the **MaximumIterations ('MaximumIterations Property' in the on-line documentation)** property for the sheet.
3. Specify the maximum amount of change that can occur with each iteration by setting the **MaximumChange ('MaximumChange Property' in the on-line documentation)** property for the sheet.
4. If needed, set the reference style for the sheet with the **ReferenceStyle ('ReferenceStyle Property' in the on-line documentation)** property.
5. Define the formulas with the circular reference(s) in the cells.

Example

This example sets formulas.

C#

```
fpSpread1.ActiveSheet.Iteration = true;
fpSpread1.ActiveSheet.SetValue(0, 1, 20);
fpSpread1.ActiveSheet.MaximumChange = 5;
fpSpread1.ActiveSheet.MaximumIterations = 5;
fpSpread1.ActiveSheet.SetFormula(0, 2, "A1*3");
fpSpread1.ActiveSheet.SetFormula(0, 0, "B1+C1");
```

VB

```
fpSpread1.ActiveSheet.Iteration = True
fpSpread1.ActiveSheet.SetValue(0, 1, 20)
fpSpread1.ActiveSheet.MaximumChange = 5
fpSpread1.ActiveSheet.MaximumIterations = 5
fpSpread1.ActiveSheet.SetFormula(0, 0, "B1+C1")
fpSpread1.ActiveSheet.SetFormula(0, 2, "A1*3")
```

Locate Circular References in a Formula

In Spread for Winforms, you can use the **CircularFormula ('CircularFormula Event' in the on-line documentation)** event to detect circular references and eliminate them in order to avoid calculation errors in the formulas used in spreadsheets.

Using Code

1. Create a new Circular Formula event.
2. Run a for loop to find all circular references in the spreadsheet.
3. Eliminate circular references from the spreadsheet.

Example

This example detects circular references in a formula.

C#

```
fpSpread1.CircularFormula += delegate (object sender1, CircularFormulaEventArgs e1)
{
    for (int i = 0; i < e1.CircularCells.Count; i++)
    {
        Console.WriteLine("Circular formula detected at cell [{0:d}, {0:d}]!",
e1.CircularCells[i].Row, e1.CircularCells[i].Column);
    }
};
fpSpread1.ActiveSheet.Cells[3, 3].Formula = "A1";
fpSpread1.ActiveSheet.Cells[0, 0].Formula = "A2";
fpSpread1.ActiveSheet.Cells[1, 0].Formula = "D4";
```

VB

```
fpSpread1.CircularFormula += Sub(sender1 As Object, e1 As CircularFormulaEventArgs) For
i As Integer = 0 To e1.CircularCells.Count - 1
    Debug.WriteLine("Circular formula detected at cell [{0:d}, {0:d}]!",
e1.CircularCells(i).Row, e1.CircularCells(i).Column)
Next
fpSpread1.ActiveSheet.Cells(3, 3).Formula = "A1"
fpSpread1.ActiveSheet.Cells(0, 0).Formula = "A2"
fpSpread1.ActiveSheet.Cells(1, 0).Formula = "D4"
```

Using DataTable Formula

A **DataTable** is a range in which you can modify the values in some of the cells and come up with different solutions to a problem. The **DataTable** feature helps in performing what-if analysis in worksheets, a process in which changing the values in cells affects the outcome of the worksheet formulas. It can be used in financial data analysis, for example, to experiment with different values and observe the corresponding variation in results. A good example of a data table uses the **PMT** function with different loan amounts and interest rates to calculate the affordable amount on a mortgage loan.

The **DataTable** formula can be applied to a range of cells using the **DataTable** method of **IRange** interface, which creates a data table based on the input values and formulas that you define on a worksheet.

The **DataTable** method accepts the following parameters:

- **rowInput**: A single cell to use as the row input for your table.
- **columnInput**: A single cell to use as the column input for your table.

Depending on the number of variables and formulas that you need to test, there are one-variable and two-variable data tables.

One-Variable DataTable

To see how different values of one variable in one or more formulas impact the results of those formulas, you can use a one-variable data table.

In the following image, cell F2 contains the payments formula **PMT(B3/12,B2,-B1)**, which refers to the input cell B3. The **DataTable** is set for Range E2:F7. Here, the **DataTable** method creates a data table based on column input value in cell B3 and the formula defined in the cell F2. Since the data table is column-oriented, that is, the variable values are in a column, the formula in the cell is entered one row above and one cell to the right of the column of values. The image below depicts the one-variable data table that is column-oriented, and the formula is contained in cell F2. Likewise, if the data table is row-oriented, that is, the variable values are arranged in a row, the formula is entered in a cell one column to the left of the first value and one cell below the row of values.

	A	B	C	D	E	F	G
1	Loan amount	100000					
2	Term in Months	180				1014.267	
3	Interest Rate	0.09	← Input cell B3		0.05	790.7936	
4	Payment	1014.267			0.055	817.0835	
5			Cell values substituted by Spread in place of input cell →		0.06	843.8568	
6					0.07	898.8283	
7					0.08	955.6521	
8							
9							

As observed, substituting values in cells, E3, E4, E5, E6 and E7, automatically updates the possible outcomes of Payment values in cells F3, F4, F5, F6 and F7 using the payments formula.

Using Code

C#

```
// One-variable data table
TestWorkBook.WorkbookSet.CalculationEngine.CalcFeatures = CalcFeatures.All;
fpSpread1.Features.ExcelCompatibleKeyboardShortcuts = true;
fpSpread1.ActiveSheet.SetClip(0, 0, 4, 1, "Loan amount\nTerm in Months\nInterest Rate\nPayment");
TestActiveSheet.Cells[0, 1].Value = 100000;
TestActiveSheet.Cells[1, 1].Value = 180;
TestActiveSheet.Cells[2, 1].Value = 0.09;
TestActiveSheet.Cells[3, 1].Formula = "PMT(B3/12,B2,-B1)";
TestActiveSheet.Cells["F2"].Formula = "B4";
TestActiveSheet.SetValue(2, 4, new object[,] { { 0.05 }, { 0.055 }, { 0.06 }, { 0.07 }, { 0.08 } });
TestActiveSheet.Range("E2:F7").DataTable("", "B3");
```

VB

```
TestWorkBook.WorkbookSet.CalculationEngine.CalcFeatures = CalcFeatures.All
fpSpread1.Features.ExcelCompatibleKeyboardShortcuts = True
fpSpread1.ActiveSheet.SetClip(0, 0, 4, 1, "Loan amount" & vbLf & "Term in Months" &
vbLf & "Interest Rate" & vbLf & "Payment")
TestActiveSheet.Cells(0, 1).Value = 100000
TestActiveSheet.Cells(1, 1).Value = 180
TestActiveSheet.Cells(2, 1).Value = 0.09
TestActiveSheet.Cells(3, 1).Formula = "PMT(B3/12,B2,-B1)"
TestActiveSheet.Cells("F2").Formula = "B4"
TestActiveSheet.SetValue(2, 4, New Object(,) {{0.05},{0.055},{0.06},{0.07},{0.08}})
TestActiveSheet.Range("E2:F7").DataTable("", "B3")
```

Two-Variable DataTable

A two-variable DataTable uses a formula that contains two lists of input values, column and row. So, the formula must refer to two different input cells.

A two-variable DataTable depict how different combinations of interest rates and loan terms can affect the payment. In

the image here, cell E2 contains the payment formula, =PMT(B3/12,B2,-B1), which uses two input cells, B1 and B3. The DataTable method creates data table for the Range E2:J7.

Cell values substituted by Spread in place of input row cell

	A	B	C	D	E	F	G	H	I	J
1	Loan amount	100000								
2	Term in Months	180			1014.267	20000	30000	40000	50000	60000
3	Interest Rate	0.09			0.05	158.1587	237.2381	316.3175	395.3968	474.4762
4	Payment	1014.267			0.055	163.4167	245.125	326.8334	408.5417	490.2501
5					0.06	168.7714	253.157	337.5427	421.9284	506.3141
6					0.07	179.7657	269.6485	359.5313	449.4141	539.297
7					0.08	191.1304	286.6956	382.2608	477.826	573.3913

Cell values substituted by Spread in place of input column cell

As observed, the variable values are substituted in rows as well as columns. The column input cell B3 substitutes values in cells E3 through E7, while the row input cell B1 substitutes values in cells F2 through J2.

Using Code

C#

```
// Two variable data table
fpSpread1.AsWorkbook().WorkbookSet.CalculationEngine.CalcFeatures =
GrapeCity.Spreadsheet.CalcFeatures.All;
fpSpread1.ActiveSheet.SetClip(0, 0, 4, 1, "Loan amount\nTerm in Months\nInterest
Rate\nPayment");
TestActiveSheet.Cells[0, 1].Value = 100000;
TestActiveSheet.Cells[1, 1].Value = 180;
TestActiveSheet.Cells[2, 1].Value = 0.09;
TestActiveSheet.Cells[3, 1].Formula = "PMT(B3/12,B2,-B1)";
TestActiveSheet.Cells["E2"].Formula = "B4";
TestActiveSheet.SetValue(2, 4, new object[,] { { 0.05 }, { 0.055 }, { 0.06 }, { 0.07 },
{ 0.08 } });
TestActiveSheet.SetValue(1, 5, new object[,] { { 20000, 30000, 40000, 50000, 60000 }
});
TestActiveSheet.Range("E2:J7").DataTable("B1", "B3");
```

VB

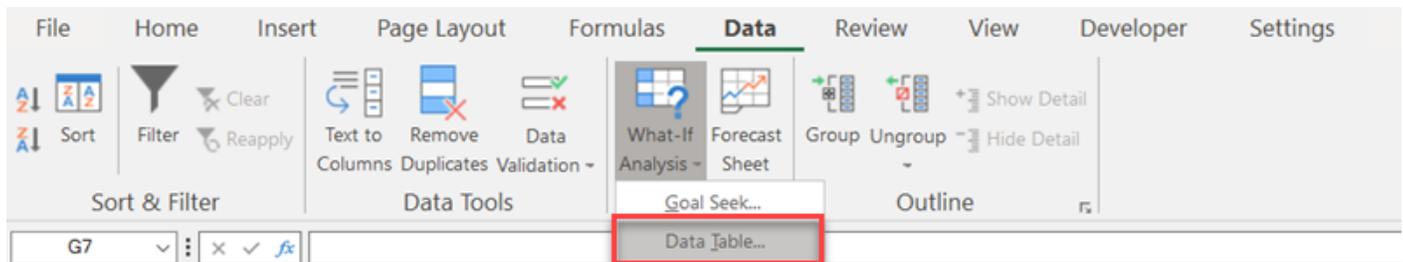
```
' Two variable data table
fpSpread1.AsWorkbook().WorkbookSet.CalculationEngine.CalcFeatures =
GrapeCity.Spreadsheet.CalcFeatures.All
fpSpread1.ActiveSheet.SetClip(0, 0, 4, 1, "Loan amount\nTerm in Months\nInterest
```

```

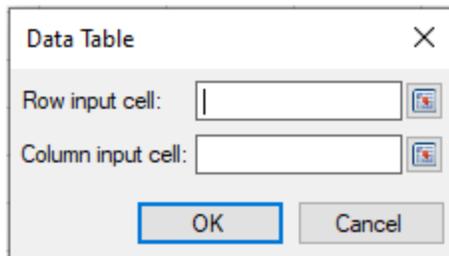
Rate\nPayment")
TestActiveSheet.Cells(0, 1).Value = 100000
TestActiveSheet.Cells(1, 1).Value = 180
TestActiveSheet.Cells(2, 1).Value = 0.09
TestActiveSheet.Cells(3, 1).Formula = "PMT(B3/12,B2,-B1)"
TestActiveSheet.Cells("E2").Formula = "B4"
TestActiveSheet.SetValue(2, 4, New Object(,) {{0.05},{0.055},{0.06},{0.07},{0.08}})
TestActiveSheet.SetValue(1, 5, New Object(,) {{20000, 30000, 40000, 50000, 60000}})
TestActiveSheet.Range("E2:J7").DataTable("B1", "B3")
    
```

Using Spread Designer

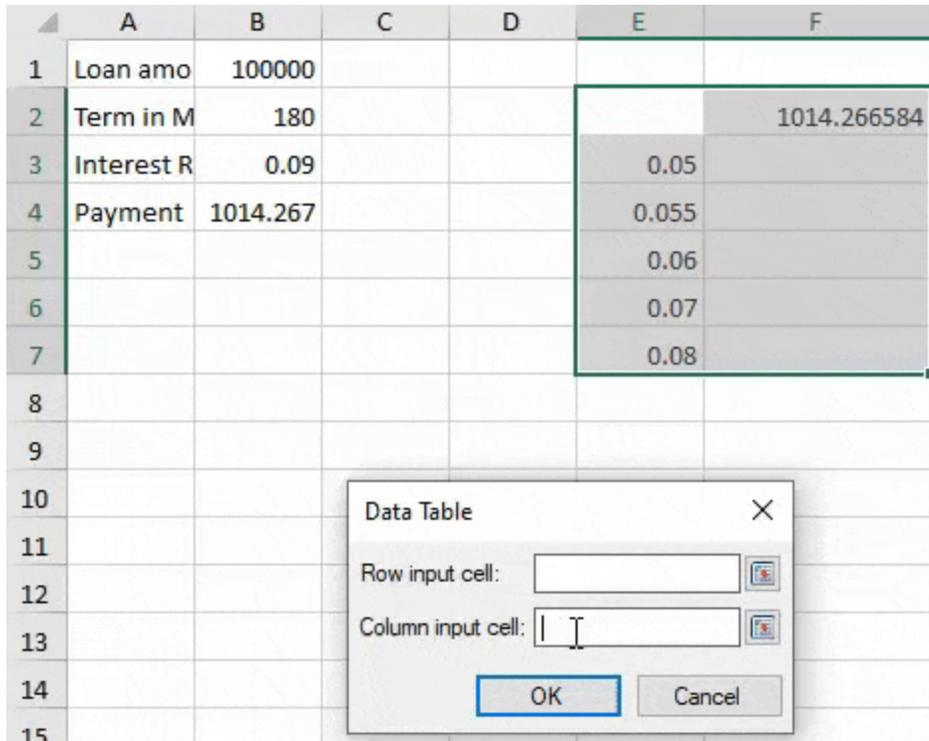
The Data Table dialog can be invoked by navigating to the **Data Table** option in the **What-If Analysis** menu of the Spread ribbon.



Click the **Data Table** option to invoke the dialog.



The dialog lets the user enter the desired row input and column input cells to create a data table.



 **Note:** The row or column input cell must be a single cell in the current worksheet and there should be at least one value inputted to create a data table in the worksheet.

Users can also invoke the **Data Table** dialog at runtime using the **DataTable** method in **BuiltInDialogs** class. For more information, see **Working with BuiltInDialogs**.

Keyboard Shortcut

The Keyboard shortcut to Data Table dialog is Alt + A + W + T.

Nesting Functions in a Formula

You can nest a function within another function in a formula.

For information on using the Formula Editor to enter a formula at design time, refer to **Entering a Formula in Spread Designer (on-line documentation)**. For details on the functions and operators that can be used to create a formula, refer to the [Formula Reference](#).

Using Code

1. If needed, set the reference style for the sheet with the **ReferenceStyle ('ReferenceStyle Property' in the on-line documentation)** property.
2. Use a function within another function in a formula.

Example

In this example the sum of the value in two cells (found by using the SUM function) is embedded in a PRODUCT formula. First the cell types are set and the values of the cells are set.

C#

```
fpSpread1.Sheets[0].Cells[3, 1].Formula = "PRODUCT(A1, SUM(A2,A3))";
```

VB

```
fpSpread1.Sheets(0).Cells(3, 1).Formula = "PRODUCT(A1, SUM(A2,A3))"
```

Recalculating and Updating Formulas Automatically

By default, the spreadsheet recalculates formulas in the spreadsheet when the contents of dependent cells change. You can turn this recalculation off. You can also recalculate an individual cell.

Also by default, the spreadsheet updates formulas when you add, insert, or remove columns or rows or when you move or swap blocks of cells. You can turn off these automatic formula updates, but generally you probably want the spreadsheet to update formulas in these cases. Keep in mind how turning off automatic formula updating might impact the spreadsheet if the user moves data, adds rows or columns, or performs other actions that affect the location of data.

When automatic formula updating is on, the spreadsheet updates absolute and relative cell references, as follows:

- When the spreadsheet is updating formulas, it updates absolute cell references when the cell referenced by the formula is part of the block that has changed.
For example, if you have a formula in cell C3 that references cell A1, which uses an absolute reference, and then add a row to the top of the spreadsheet, you now want the formula to reference cell A2, because cell A1 is empty. If the spreadsheet did not update the formula, your formula would be referencing different data.
- When the spreadsheet is updating formulas, it updates relative cell references when the cell referenced by the formula is not part of the block that has changed.
For example, if you have a formula in cell C3 that references cell C1 as a relative reference, it references cell C1 as the cell that is two cells above it. If you add a row between row 2 and row 3, cell C3 is now C4, and the relative address references cell C2, the cell two cells above it. Therefore, to use the same data in the formula, the spreadsheet updates the cell reference to the cell three cells above it, C1.

Use the **AutoCalculation** ('**AutoCalculation Property**' in the **on-line documentation**) property to turn on or off the automatic recalculation of formulas. Use the **Recalculate** ('**Recalculate Method**' in the **on-line documentation**) method for recalculating formulas.

Using Code

1. If needed, set the reference style for the sheet with the **ReferenceStyle** ('**ReferenceStyle Property**' in the **on-line documentation**) property.
2. Add a formula to the cell with the **SetFormula** ('**SetFormula Method**' in the **on-line documentation**) method.
3. Set the **Recalculate** ('**RecalculateAll Method**' in the **on-line documentation**) method to recalculate.

Example

This example recalculates all the formulas.

C#

```
fpSpread1.ActiveSheet.SetValue(0, 0, 20);  
fpSpread1.ActiveSheet.SetValue(0, 1, 10);  
fpSpread1.ActiveSheet.SetFormula(3, 0, "SUM(A1,B1)");  
fpSpread1.ActiveSheet.SetFormula(4, 0, "A1*B1");  
fpSpread1.ActiveSheet.SetValue(0, 1, 100);  
fpSpread1.ActiveSheet.Recalculate();
```

VB

```
fpSpread1.ActiveSheet.SetValue(0, 0, 20)
fpSpread1.ActiveSheet.SetValue(0, 1, 10)
fpSpread1.ActiveSheet.SetFormula(3, 0, "SUM(A1,B1)")
fpSpread1.ActiveSheet.SetFormula(4, 0, "A1*B1")
fpSpread1.ActiveSheet.SetValue(0, 1, 100)
fpSpread1.ActiveSheet.Recalculate()
```

Finding a Value Using GoalSeek

You can use the **GoalSeek ('GoalSeek Method' in the on-line documentation)** method to find a value that will produce the desired result for a formula. An approximation is acceptable. You can set the starting and intended goal of the calculation.

Using Code

Use the **GoalSeek** method to find the required result.

Example

In this example the formula is in cell (1,1). The result that you want to see in the formula cell is 32. The value in C1 is what is required to get a result of 32.

C#

```
fpSpread1.Sheets[0].Cells[1, 1].Formula = "C1+D1";
fpSpread1.Sheets[0].Cells[0, 3].Value = 2;
fpSpread1.GoalSeek(0, 0, 2, 0, 1, 1, 32);<
```

VB

```
fpSpread1.Sheets(0).Cells(1, 1).Formula = "C1+D1"
fpSpread1.Sheets(0).Cells(0, 3).Value = 2
fpSpread1.GoalSeek(0, 0, 2, 0, 1, 1, 32)
```

Allowing the User to Enter Formulas

Spread for WinForms supports the ability to type in a formula by simply starting with an equal's sign (=). This is the default behavior in the worksheet where the displayed result depends on the cell type.

For example, an integer cell type displays the result as an integer, even if the result of the formula is not an integer. In this case, a number may appear rounded.

	A	B	C	D
1	1			
2				
3				
4	+			
5				
6				

You can control whether to allow the users to enter formulas in a worksheet by setting the boolean value in `FpSpread.AllowUserFormulas` ('AllowUserFormulas Property' in the on-line documentation) property.

C#

```
fpSpread1.AllowUserFormulas = true;
```

VB

```
fpSpread1.AllowUserFormulas = True
```

To define specific formatting for currency and date cells or setting various cell types, refer to **Working with Editable Cell Types**.

For information on the floating formula bar, where users can type in formulas and select cells dynamically, refer to **Working with the Formula Text Box**.

Using Plus or Minus Sign

You can also input a formula in a worksheet by starting with a plus sign (+) or a minus sign (-). These signs function the same as the equal sign (=) and Excel users who are accustomed to using the plus or minus sign don't have to change their behavior when using Spread.

The equal sign is still automatically added before, so that formulas are converted to work as expected. For example, if you enter "+SUM(1,2)" when editing, it will be "=+SUM(1,2)" when displayed again.

	A	B	C	D
1	1			
2				
3				
4	+			
5				
6				

When using the plus or minus sign to enter a formula, the users should be aware of the following behavior:

- If the text cannot be parsed as a formula, it'll be parsed as a simple value.
- If the text can be parsed as a formula, then:

- If there are only numeric values ("+1"), percentage values ("+10%"), or percentage for scientific formula ("+1e-20%") then it will be parsed as a simple value along with its corresponding format.
- Otherwise, the formula is set to the cell input.

 **Note:** This behavior only works with default cell type in flat style mode.

Using the Properties Window

1. At design time, in the **Properties** window (in the **Behavior** category), select the **AllowUserFormulas** property.
2. Select **True** from the drop-down list to allow users to enter formulas or select **False** to prohibit them.

Using the Spread Designer

1. Select the Spread component (or select Spread from the pull-down menu).
2. In the property list for the component (in the **Behavior** category), select the **AllowUserFormulas** property and select the value **True**.

From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Creating and Using a Custom Name

Custom, user-defined names are identifiers to represent information in the spreadsheet, used mostly in formulas. A custom name can refer to a cell, a range of cells, a computed value, or a formula. You can define a custom name and then use the name in formulas. When the formula is evaluated, the custom name's value is referenced and evaluated.

You can create sheet level or workbook level custom names. The scope of the sheet level custom name is limited to the sheet for which it was created. This allows you to use the same name on several sheets. Formulas in a sheet will ignore sheet level custom names on other sheets.

Use the **AddCustomName ('AddCustomName Method' in the on-line documentation)** method in the **SheetView ('SheetView Class' in the on-line documentation)** class to add workbook or sheet level custom names. The *sheetViewScope* parameter in the **AddCustomName** method can be set to true for a sheet level custom name and false for a workbook level custom name.

Avoid using custom names that start with C# or R# patterns (# stands for any number).

It also supports the hidden custom Name.

If a customer creates a custom name without specifying the sheet name (using reference like A1, A1:A2 etc.), the reference will automatically refer to the active sheet. In other words, if you want to use the current sheet with the custom name's formula, you must use special syntax. For instance, if a user provides "!A1" as an input, the final formula of custom name will be changed to "Sheet1!A1".

Using Code

Define the custom name using the **AddCustomName ('AddCustomName Method' in the on-line documentation)** method for the workbook or sheet.

Example

To add a custom name for a cell specified with A1 notation, use the **AddCustomName ('AddCustomName Method' in the on-line documentation)** method as shown in the following example, which creates a workbook level custom name.

C#

```
fpSpread1.Sheets[0].AddCustomName("test", "$B$1", 0, 0);
```

VB

```
fpSpread1.Sheets[0].AddCustomName("test", "$B$1", 0, 0);
```

To add a custom name for a computed value, use the **AddCustomName** ('AddCustomName Method' in the **on-line documentation**) method as shown in this code:

C#

```
fpSpread1.Sheets[0].AddCustomName("alpha", "101", 0, 0);
```

VB

```
fpSpread1.Sheets[0].AddCustomName("alpha", "101", 0, 0);
```

The following example adds a name that is a range reference.

C#

```
fpSpread1.Sheets[0].AddCustomName("Sales", "Sheet1!$F$20:$F$50", 0, 0);
```

VB

```
fpSpread1.Sheets[0].AddCustomName("Sales", "Sheet1!$F$20:$F$50", 0, 0);
```

Using the Spread Designer

1. Select the **Data** menu in the Spread Designer.
2. Select the **Name Manager** icon.
3. Use the **New** button to add custom names and click the **Close** button when finished.
4. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Creating and Using a Custom Function

If you have functions that you use on a regular basis that are not in the built-in functions or if you wish to combine some of the built-in functions into a single function, you can do so by defining your own custom functions. They can be called in the same way as you would call any of the built-in functions.

A custom function can have the same name as a built-in function. The custom function takes priority over the built-in function. Custom functions are dynamically linked at evaluation time. Thus, the application can redefine an existing built-in function, if the custom function uses the same name and is added before the formula is parsed.

If a formula attempts to call a custom function with a parameter count outside of the range indicated by the **MinArgs** ('MinArgs Property' in the **on-line documentation**) property and **MaxArgs** ('MaxArgs Property' in the **on-line documentation**) property of the function, then the **Evaluate** ('Evaluate Method' in the **on-line documentation**) method of the function is skipped and the #VALUE! error value is used as the result.

Also, if a formula attempts to call a custom function with a parameter that is an error value (for example, #NUM!, #VALUE!, #REF!) and the **GetError()** ('GetError Method' in the **on-line documentation**) method of the **IReadOnlyPrimitiveValue** ('IReadOnlyPrimitiveValue Interface' in the **on-line documentation**) interface returns False for that parameter, then the **Evaluate()** method of the **Function** ('Function Class' in the **on-line documentation**) class is skipped and the error value is used as the result.

The **Evaluate()** method evaluates the custom function based on the specified arguments and assigns the evaluated value to the result.

Using Code

1. Define the custom function(s).
2. Register the function(s) in the sheet.
3. Use the custom function(s).

Example

The first step is to create the custom functions. In this example, we will create a Tax Value function that evaluates the tax and returns a numeric value in the specified cells in the spreadsheet.

The following code defines the Tax Value custom function.

C#

```
public class TaxValueFunction : GrapeCity.CalcEngine.Function
{
    public TaxValueFunction() : base("TAXVALUE", 1, 2, FunctionAttributes.SingleCell |
FunctionAttributes.Number) { }
    protected override void Evaluate(IArguments arguments, IValue result)
    {
        IEvaluationContext context = arguments.EvaluationContext;
        double num = arguments[0].GetNumber(context);
        double taxrate = arguments.Count > 1 ? arguments[1].GetNumber() : 0.15;
        result.SetValue(null, num - (num * taxrate));
    }
}
```

The following code registers the custom functions.

C#

```
fpSpread1.AddCustomFunction(new TaxValueFunction());
```

The following code implements the custom functions in formulas.

C#

```
fpSpread1.ActiveSheet.Cells[1,1].Formula = "TAXVALUE (A1)";
```

Parameters in Custom Functions

There are two ways to specify arguments in custom functions- passing parameters by value and passing parameters by reference.

By default, parameters are passed by value (if you're using a single cell). A single empty cell is passed as null (Nothing in Visual Basic). A single non-empty cell is passed as a boxed primitive (for example, double, boolean, string, and so on).

If you're using a cell range, parameters are passed by reference.

To work with parameters in custom functions, you can access the methods and properties of the **Function ('Function Class' in the on-line documentation)** class from within a derived class.

The **GetValue()** ('GetValue Method' in the on-line documentation) method of the **IReferenceSource ('IReferenceSource Interface' in the on-line documentation)** interface and the **SetValue()** ('SetValue Method' in the on-line documentation) method of the **IPrimitiveValue ('IPrimitiveValue Interface' in the on-line documentation)** interface can be used to get or set a single value from the reference. The row and the column indexes for the **GetValue** method and the **SetValue** method start at the row and the column.

Example

In this example, a function counts the number of cells in a range that are less than a given criteria.

C#

```
class CountIfLessThanFunction : GrapeCity.CalcEngine.Function
{
    public CountIfLessThanFunction() : base("COUNTIFLESSTHAN", 2, 2,
    GrapeCity.CalcEngine.FunctionAttributes.Number) { }
    protected override void Evaluate(IArguments arguments, IValue result)
    {
        IValue range = arguments[0];
        if (range.ValueType != GrapeCity.CalcEngine.ValueType.Reference)
        {
            arguments.EvaluationContext.Error = CalcError.Value;
            result.SetValue(CalcError.Value);
        }
        else
        {
            IEvaluationContext evaluationContext = arguments.EvaluationContext;
            double criteria = arguments[1].GetNumber(evaluationContext);
            IReferenceSource referenceSource =
            range.GetReferenceSource(evaluationContext);
            RangeReference rangeRef = range.GetReference(evaluationContext, 0);
            int count = 0;
            for (int c = rangeRef.Column; c <= rangeRef.Column2; c++)
            {
                for (int r = rangeRef.Row; r <= rangeRef.Row2; r++)
                {
                    referenceSource.GetValue(evaluationContext, r, c, result);
                    double cellValue = result.GetNumber(evaluationContext);
                    if (cellValue < criteria)
                    {
                        count++;
                    }
                }
            }
            result.SetValue(evaluationContext, count);
        }
    }
}
```

Creating and Using a Visual Function

Spread for Winforms provides support for working with data visualization functions in the spreadsheet.

Data Visualization functions are custom functions that allows users to add custom logic (either painting the logic or applying the style format logic) to create a visual in a cell that can be used in a formula like custom functions.

The **IsVisual ('IsVisual Property' in the on-line documentation)** property of the **FunctionVisualizer ('FunctionVisualizer Class' in the on-line documentation)** class can be used to indicate whether the function used in the spreadsheet is a data visualization function or not.

Creating a Data Visualization Function

For creating a data visualization function in the spreadsheet, users need to first create a custom Data Visualizer class that inherits from the FunctionVisualizer class. Further, the **ApplyFormat() ('ApplyFormat Method' in the on-line documentation)** method and the **IsShowCell() ('IsShowCell Method' in the on-line documentation)**

method of the **FunctionVisualizer** class can be used to work with data visualization functions in the spreadsheet. Refer to the following example code to create a data visualization function.

C#

```
public class ErrorFunctionVisualizer : FunctionVisualizer
{
    protected override void PaintCell(System.Drawing.Graphics graphics,
    System.Drawing.Rectangle rect, object cellValue, ref StyleFormat styleFormat,
    IWorksheet worksheet)
    {
        rect.Width--;
        rect.Height--;

        if (cellValue is VisualizationData visualizationData)
        {
            IEvaluationContext context =
            worksheet.Workbook.WorkbookSet.
            CalculationEngine.EvaluationContext;
            if (visualizationData.Value.GetValue(context) is
            GrapeCity.CalcEngine.CalcError)
            {
                System.Drawing.Pen pen =
                new System.Drawing.Pen(System.Drawing.ColorTranslator.
                FromHtml(visualizationData.Parameters[1].GetText(context)));
                pen.Width = 2;
                graphics.DrawEllipse(pen, rect);
                pen.Dispose();
            }
            else if (cellValue is GrapeCity.CalcEngine.CalcError)
            {
                graphics.DrawRectangle(System.Drawing.Pens.Red, rect);
            }
        }
    }
}
```

Using a Data Visualization Function

While using a data visualization function, users can assign any name for the function to use it inside the formula but it is important to remember that the prefix "VF" must be used before the name of the data visualization function.

Refer to the following example code to use the existing data visualization function in the spreadsheet.

C#

```
private void DataVisualizationFunction_Load(object sender, EventArgs e)
{
    /* Using a Data Visualization Function
    To use the FunctionVisualizer, you must create a new
    instance of VisualFunction and pass a new instance of
    the FunctionVisualizer in the constructor */

    fpSpread1.AddCustomFunction(new VisualFunction("ERROR",
    1, 2, FunctionAttributes.Variant, new ErrorFunctionVisualizer()));

    /* Prefix "VF." is required before the VisualFunction's name
    for using the custom VisualFunction in a cell formula */
}
```

```
fpSpread1.AsWorkbook().ActiveSheet.Cells["B2"].Formula = "VF.ERROR(1/0, A1)";
fpSpread1.AsWorkbook().ActiveSheet.Cells["A1"].Value =
System.Drawing.ColorTranslator.ToHtml(System.Drawing.Color.Red);
}
```

Creating and Using External Variable

Spread for WinForms provides support for creating and using external variables in spreadsheets.

An external variable refers to a logical cell defined by a symbol. The name of an external variable is case-sensitive. External variables don't possess a cell context and are evaluated distinctly than a normal cell.

- If a reference in the formula for the external variable is missing a reference to the worksheet, then that reference will refer to the active sheet.
- Formulas for external variables are always evaluated using absolute references. If the formula for the external reference uses relative references, then those references will be converted to absolute references when evaluating the formula.
- The external variable formula can result in a range reference when it is evaluated; in this case, the external variable can be used in an array formula. If such a formula is evaluated for a single cell, which requires a primitive value result, then the value in the top-left cell in the range is used as the context cell, and that cell's value will be returned as the evaluated result.
- The external variable can specify a class instance which inherits from `ExternalVariable`, which enables the application to implement custom logic to bind external objects with the calculation engine. For a detailed example showing how to inherit from `ExternalVariable`, refer to **Using External Variables with TextBox Control**.

The `AddExternalVariable` method of the `INames` (**'INames Interface' in the on-line documentation**) interface can be used to add an external variable to the workbook. While using this method, users need to provide the name of the external variable and the formula or the value of the variable.

Using Code

Define the external variable using the `AddExternalVariable` method for the workbook.

Example

The following example code creates two external variables "x" and "y" and uses them within the formulas defined in a particular worksheet of the workbook.

C#

```
// Add a new sheet to your workbook.
FarPoint.Win.Spread.SheetView newsheet = new FarPoint.Win.Spread.SheetView();
fpSpread1.Sheets.Add(newsheet);
fpSpread1.AllowUserFormulas = true;

// Set the values of A1 cell and A2 cell of the active sheet.
fpSpread1.ActiveSheet.Cells[0, 0].Value = "Hello";
fpSpread1.ActiveSheet.Cells[1, 0].Value = "World!";

// Add external variables to the workbook that refer to the value of the cells.
fpSpread1.AsWorkbook().Names.AddExternalVariable("x", "Sheet1!A1");
fpSpread1.AsWorkbook().Names.AddExternalVariable("y", "Sheet1!A2");

// Use the external variables in different cell.
fpSpread1.AsWorkbook().Worksheets[1].Cells["B1"].Formula = "\"The concat name is \" & x
```

```
& y";
```

VB

```
Add a new sheet to your workbook.
Dim newsheet As FarPoint.Win.Spread.SheetView = New FarPoint.Win.Spread.SheetView()
fpSpread1.Sheets.Add(newsheet)
fpSpread1.AllowUserFormulas = true

' Set the values of A1 cell and A2 cell of the active sheet.
fpSpread1.ActiveSheet.Cells(0, 0).Value = "Hello"
fpSpread1.ActiveSheet.Cells(1, 0).Value = "World!"

' Add external variables to the workbook that refer to the value of the cells.
fpSpread1.AsWorkbook().Names.AddExternalVariable("x", "Sheet1!A1")
fpSpread1.AsWorkbook().Names.AddExternalVariable("y", "Sheet1!A2")

' Use the external variables in different cell.
fpSpread1.AsWorkbook().Worksheets(1).Cells("B1").Formula = ""The concat name is "" & x
& y"
```

Using External Variables with Text Box Control

The following example code shows how to create and use external variables with the standard textbox control.

In the example:

- There are two external variables : "x" and "y" and two textbox controls : "textBox1" and "textBox2"
- Cell B1 and textBox1 will provide the arguments for the formula. The arguments can be modified to display the updated result.
- The result will be displayed in textBox2.

C#

```
// Creating and defining the external variable
public class TextBoxExternalVariable : ExternalVariable
{
    private TextBox _textBox;
    private bool _asInput;
    public TextBoxExternalVariable(TextBox textBox, bool asInput)
    {
        _textBox = textBox;
        if (asInput)
        {
            textBox.TextChanged += TextBox_TextChanged;
        }
    }
    private void TextBox_TextChanged(object sender, EventArgs e)
    {
        Dirty();
    }

    protected override bool OnDirtying()
    {
        return !_asInput;
    }
}
```

```

protected override void OnDirtied()
{
    Refresh();
}

protected override void EvaluateCore(IEvaluationContext context,
IValue result)
{
    string text = _textBox.Text;
    if (!string.IsNullOrEmpty(text) && double.TryParse(text,
        out double dblValue))
    {
        result.SetValue(dblValue);
    }
    else
    {
        result.SetValue(text);
    }
}

public void Refresh()
{
    if (!_asInput)
    {
        _textBox.Text = this.Value.GetText();
    }
}
}
// Using the external variable with text box control
private void Form2_Load(object sender, EventArgs e)
{
    var workbook = fpSpread1.AsWorkbook();
    var activeSheet = workbook.ActiveSheet;
    activeSheet.Cells["A1"].Value = "Factor";
    activeSheet.Cells["B1"].Value = 2;

    // Adding Ext. Variable - x referring to textbox1
    workbook.Names.AddExternalVariable("x",
    new TextBoxExternalVariable(textBox1, true));

    // Adding Ext. Variable "y" referring to textbox2 with formula "Sheet1!B1 * x"
    var extVariable2 = new TextBoxExternalVariable(textBox2, false);
    workbook.Names.AddExternalVariable("y", extVariable2, "Sheet1!B1 * x");
    extVariable2.Refresh();
}

```

VB

```

'Creating and defining the external variable
Public Class TextBoxExternalVariable Inherits ExternalVariable
Private _textBox As TextBox
Private _asInput As Boolean
Public Sub New(ByVal textBox As TextBox, ByVal asInput As Boolean)
    _textBox = textBox
    If asInput Then
        textBox.TextChanged += AddressOf TextBox_TextChanged
    End If
End Sub

```

```

End If
End Sub
Private Sub TextBox_TextChanged(ByVal sender As Object, ByVal e As EventArgs)
    Dirty()
End Sub
Protected Overrides Function OnDirtying() As Boolean
    Return Not _asInput
End Function
Protected Overrides Sub OnDirtied()
    Refresh()
End Sub
Protected Overrides Sub EvaluateCore(ByVal context As IEvaluationContext, ByVal result
As IValue)
    Dim text As String = _textBox.Text
    Dim dblValue As Double = Nothing
    If Not String.IsNullOrEmpty(text) AndAlso Double.TryParse(text, dblValue) Then
        result.SetValue(dblValue)
    Else
        result.SetValue(text)
    End If
End Sub
Public Sub Refresh()
    If Not _asInput Then
        _textBox.Text = Me.Value.GetText()
    End If
End Sub
End Class

'Using the external variable with text box control
Private Sub Form2_Load(ByVal sender As Object, ByVal e As EventArgs)
    Dim workbook = fpSpread1.AsWorkbook()
    Dim activeSheet = workbook.ActiveSheet
    activeSheet.Cells("A1").Value = "Factor"
    activeSheet.Cells("B1").Value = 2
    workbook.Names.AddExternalVariable("x", New TextBoxExternalVariable(textBox1, True))
    Dim extVariable2 = New TextBoxExternalVariable(textBox2, False)
    workbook.Names.AddExternalVariable("y", extVariable2, "Sheet1!B1 * x")
    extVariable2.Refresh()
End Sub

```

Using the Array Formula

You can use array formulas in Spread.

An array formula is a formula that can perform multiple calculations on one or more items in an array. Array formulas can return multiple results or a single result.

Use Ctrl + Shift + Enter to create an array formula after entering the formula at run time if the users are allowed to create formulas, or you can use the **SetFormulaArray** method.

The following image displays an array formula in the Total column.

	Units	Price	Total
1	2	40	80
2	5	100	500
3	1	25	25
4	9	80	720

Using Code

The following example creates an array formula.

C#

```
fpSpread1.ActiveSheet.ColumnHeader.Cells[0, 0].Text = "Units";
fpSpread1.ActiveSheet.ColumnHeader.Cells[0, 1].Text = "Price";
fpSpread1.ActiveSheet.ColumnHeader.Cells[0, 2].Text = "Total";
fpSpread1.ActiveSheet.Cells[0, 0].Value = 2;
fpSpread1.ActiveSheet.Cells[1, 0].Value = 5;
fpSpread1.ActiveSheet.Cells[2, 0].Value = 1;
fpSpread1.ActiveSheet.Cells[3, 0].Value = 9;
fpSpread1.ActiveSheet.Cells[0, 1].Value = 40;
fpSpread1.ActiveSheet.Cells[1, 1].Value = 100;
fpSpread1.ActiveSheet.Cells[2, 1].Value = 25;
fpSpread1.ActiveSheet.Cells[3, 1].Value = 80;
fpSpread1.AllowUserFormulas = true;
fpSpread1.ActiveSheet.Cells[0, 2, 3, 2].FormulaArray = "A1:A4*B1:B4";
```

VB

```
fpSpread1.ActiveSheet.ColumnHeader.Cells(0, 0).Text = "Units"
fpSpread1.ActiveSheet.ColumnHeader.Cells(0, 1).Text = "Price"
fpSpread1.ActiveSheet.ColumnHeader.Cells(0, 2).Text = "Total"
fpSpread1.ActiveSheet.Cells(0, 0).Value = 2
fpSpread1.ActiveSheet.Cells(1, 0).Value = 5
fpSpread1.ActiveSheet.Cells(2, 0).Value = 1
fpSpread1.ActiveSheet.Cells(3, 0).Value = 9
fpSpread1.ActiveSheet.Cells(0, 1).Value = 40
fpSpread1.ActiveSheet.Cells(1, 1).Value = 100
fpSpread1.ActiveSheet.Cells(2, 1).Value = 25
fpSpread1.ActiveSheet.Cells(3, 1).Value = 80
fpSpread1.AllowUserFormulas = True
fpSpread1.ActiveSheet.Cells[0, 2, 3, 2].FormulaArray = "A1:A4*B1:B4"
```

Working With Dynamic Array Formulas

Spread for WinForms provides extensive support for using dynamic array formulas in the spreadsheets. When a cell contains a dynamic array formula, multiple values are returned because the elements of the array spill into the adjacent empty cells. Unlike generic arrays, dynamic arrays automatically resize when the data is inserted or removed from the source range.

The aim of introducing dynamic array formulas is to gradually replace generic array formulas that were entered previously using Ctrl+Shift+Enter (CSE). Earlier, working with formulas in spreadsheets was an extremely cumbersome and time-consuming task because users need to copy the formulas to every cell manually where they want the result to

be calculated. But with the introduction of Dynamic Array Formulas, multiple results are returned as output via automatic spilling and spanning to the cell range. This reduces the overall overhead to a great extent and makes it much easier and quicker to work with array formulas in the spreadsheets.

Dynamic arrays are useful especially when you're looking for good locality of reference and want to implement effective data cache utilization in the spreadsheets. Further, they also facilitate random access with low memory footprints (in terms of compactness). Generally, this is possible because dynamic arrays have only a small fixed additional overhead for storing information about the size and capacity. Hence, dynamic arrays work wonders as a powerful tool for building cache-friendly data structures while working with spreadsheets.

Spilled Array Formulas

Dynamic array formulas that return more than one result and spill successfully to the nearby cells are known as Spilled array formulas. Note that Spilled array formulas are not supported in Tables. However, while working with dynamic array formulas that spill to several rows and columns, the cell ranges used in the spreadsheets can be formatted explicitly to appear like tables.

A range containing the results of dynamic array formulas spanned over multiple cells is called a Spilled range. Using the **HasSpill** property of **IRange** Interface, you can check whether a cell or a range has a spilled cell within it. It returns True if all the cells in the range are part of a spilled range, False if none of the cells in the range are part of a spilled range, and null, otherwise.

When you select any cell in the range of the generated Spilled array, the entire range is highlighted with a blue color border by default. However, you can hide this blue border by setting the **DynamicArrayRenderer** and **DynamicArrayErrorRenderer** properties of the **ISelectionRenderer** interface to null.

The following example code hides the border of a spilled dynamic array.

C#

```
fpSpread1.AsWorkbook().WorkbookSet.CalculationEngine.CalcFeatures = CalcFeatures.All;
IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;
SpreadSkin spreadSkin = fpSpread1.DefaultSkin;
spreadSkin.DynamicArrayRenderer = null;
spreadSkin.DynamicArrayErrorRenderer = null;
TestActiveSheet.Cells["C1"].Formula2 = "A1:A4";
TestActiveSheet.Cells["C1"].Select();
```

VB

```
fpSpread1.AsWorkbook().WorkbookSet.CalculationEngine.CalcFeatures = CalcFeatures.All
Dim TestActiveSheet As IWorksheet = fpSpread1.AsWorkbook().ActiveSheet
Dim spreadSkin As SpreadSkin = fpSpread1.DefaultSkin
spreadSkin.DynamicArrayRenderer = Nothing
spreadSkin.DynamicArrayErrorRenderer = Nothing
TestActiveSheet.Cells("C1").Formula2 = "A1:A4"
TestActiveSheet.Cells("C1").Select()
```

Examples of Dynamic Array Formulas

1. **UNIQUE**
2. **SORT**
3. **SORTBY**
4. **SEQUENCE**
5. **RANDARRAY**
6. **SINGLE**
7. **FILTER**

UNIQUE

The Unique function returns a list of all the unique values in a cell range.

For instance - The cell C4 in the following image contains the formula "`=UNIQUE(A4:A15)`" and returns only the unique customer names from the values in cell range A4 to A15. Based on the number of unique values, the dynamic array formula spills to the cell range C5 to C8 automatically.

	A	B	C
1	Dynamic Array Functions		
2			
3	Customer's Name	Age	Unique List
4	Larry	32	Larry
5	Safeway	23	Safeway
6	Safeway	23	Raley
7	Raley	39	Vallarta
8	Vallarta	18	Gilbert
9	Safeway	23	
10	Raley	39	
11	Larry	32	
12	Gilbert	19	
13	Larry	32	
14	Larry	32	
15	Raley	39	
16			

SORT

The SORT function sorts the data in a cell range or an array. The results of this function spill into the resultant range with a dynamic array of values arranged in the ascending (increasing) or descending (decreasing) order. If the sort order is not specified, then by default, the values are sorted alphabetically in the ascending order.

For instance - The cell D4 in the following image contains the formula "`=SORT(A4:A15)`" and returns the customer names sorted in the increasing order.

	A	B	C	D
1	Dynamic Array Functions			
2				
3	Customer's Name	Age	Unique List	Sort
4	Larry	32	Larry	Gilbert
5	Safeway	23	Safeway	Larry
6	Safeway	23	Raley	Larry
7	Raley	39	Vallarta	Larry
8	Vallarta	18	Gilbert	Larry
9	Safeway	23		Raley
10	Raley	39		Raley
11	Larry	32		Raley
12	Gilbert	19		Safeway
13	Larry	32		Safeway
14	Larry	32		Safeway
15	Raley	39		Vallarta
16				

In case you want to sort all the unique values in the range A4 to A15, you can either apply the sort function on the unique list displayed in the column C4 or you can also combine both the functions SORT and UNIQUE into a single formula.

For instance, the cell E4 in the following image contains the formula `"=SORT(C4#)"` where # indicates a list. This formula will sort the list of values in column C (where cell C4 already contains the UNIQUE formula `"=UNIQUE(A4:A15)"`) and displays the results in column E.

Alternatively, you can also combine both the functions SORT and UNIQUE. For instance, the cell F4 in the following image contains the formula `"=SORT(UNIQUE(A4:A15))"` which returns all the unique values in the range A4:A15 sorted alphabetically.

	A	B	C	D	E	F
1	Dynamic Array Functions					
2						
3	Customer's Name	Age	Unique List	Sort	Sort Unique	Sort Unique
4	Larry	32	Larry	Gilbert	Gilbert	Gilbert
5	Safeway	23	Safeway	Larry	Larry	Larry
6	Safeway	23	Raley	Larry	Raley	Raley
7	Raley	39	Vallarta	Larry	Safeway	Safeway
8	Vallarta	18	Gilbert	Larry	Vallarta	Vallarta
9	Safeway	23		Raley		
10	Raley	39		Raley		
11	Larry	32		Raley		
12	Gilbert	19		Safeway		
13	Larry	32		Safeway		
14	Larry	32		Safeway		
15	Raley	39		Vallarta		
16						

SORTBY

The SORTBY function sorts the contents of a cell range or an array on the basis of the values present in a corresponding range or array.

For instance - The cell G4 in the following image contains the formula "`=SORTBY(A4:B15,B4:B15)`". This function sorts the cell range A4 to B15 based on another cell range B4 to B15 and returns the customer names displayed along with their ages sorted in the increasing order.

	A	B	C	D	E	F	G	H
1	Dynamic Array Functions							
2								
3	Customer's Name	Age	Unique List	Sort	Sort Unique	Sort Unique	SortBy	
4	Larry	32	Larry	Gilbert	Gilbert	Gilbert	Vallarta	18
5	Safeway	23	Safeway	Larry	Larry	Larry	Gilbert	19
6	Safeway	23	Raley	Larry	Raley	Raley	Safeway	23
7	Raley	39	Vallarta	Larry	Safeway	Safeway	Safeway	23
8	Vallarta	18	Gilbert	Larry	Vallarta	Vallarta	Safeway	23
9	Safeway	23		Raley			Larry	32
10	Raley	39		Raley			Larry	32
11	Larry	32		Raley			Larry	32
12	Gilbert	19		Safeway			Larry	32
13	Larry	32		Safeway			Raley	39
14	Larry	32		Safeway			Raley	39
15	Raley	39		Vallarta			Raley	39
16								

SEQUENCE

The SEQUENCE function returns a list of sequential numbers in an array in the ascending order.

For instance - The cell A2 in the following image contains the formula "`=SEQUENCE(4,5)`" and returns an array with values spilled to a cell range containing four rows and five columns displaying numbers in the sequence 1, 2, 3, 4 upto 20.

	A	B	C	D	E
1	SEQUENCE(4,5) Function				
2	1	2	3	4	5
3	6	7	8	9	10
4	11	12	13	14	15
5	16	17	18	19	20

RANDARRAY

The RANDARRAY function returns an array of random numeric values. Users can specify the number of rows and columns, minimum and maximum values and indicate whether to return integers or decimal values.

For instance - The cell A8 in the following image contains the formula "`=RANDARRAY(5,3)`" and returns a random set of values between 0 and 1.

7	RANDARRAY(5,3) Function				
8	0.301296713654076	0.968978906944218	0.702538246962628		
9	0.167834041222712	0.528294371528925	0.620410664378921		
10	0.884531817060312	0.516241429963227	0.622292071832181		
11	0.65774892066046	0.208537678811478	0.751867999807693		
12	0.387976196850592	0.0638986509434175	0.534441710355955		
13					

SINGLE

The SINGLE function returns a single value, a single cell range or an error using the implicit intersection logic.

For instance - The cell A15 in the following image contains the formula "`=SINGLE(A15:E15)`" and returns the result "C" in the cell C16 by evaluating the intersection of the rows and columns in the cell range A15 to E15.

13					
14	SINGLE(A15:E15) Function				
15	A	B	C	D	E
16			C		
17					

FILTER

The FILTER function allows users to filter a cell range on the basis of the defined criteria. The Filter operation can be performed based on a single criterion or multiple criteria. In order to combine two or more filter conditions, users can use the "*" operator.

For instance - The cell F5 in the following image contains the formula "`=FILTER(A5:D17, C5:C17=F1)`". This formula filters the cell range A5 to D17 based on one filter criteria (when the cell range C5 to C17 matches the Product value in cell F1 i.e. Apple). As a result, all the values in the cell range A5 to D17 containing product as "Apple" will be displayed.

In another example, the cell F14 in the following image contains the formula "`=FILTER(A5:D17, (C5:C17=F1)*(A5:A17=F2))`". This formula filters the cell range A5 to D17 based on two filter conditions that are specified by the multiplication (*) operator. The first condition is the cell range C5 to C17 should match the Product value in cell F1 i.e. Apple and the second condition is the cell range A5 to A17 should match the region "East". As a result, all the values in the cell range A5 to D17 containing Product as "Apple" and Region as "East" will be displayed.

	A	B	C	D	E	F	G	H	I	J
1					Product:	Apple				
2					Region:	East				
3						Filtering performed on one Criteria				
4	Region	Sales Rep	Product	Units		Region	Sales Rep	Product	Units	
5	East	Tom	Apple	6380		East	Tom	Apple	6380	
6	North	Fred	Grape	2344		East	Hector	Apple	2341	
7	West	Amy	Pear	3434		North	Fred	Apple	2334	
8	South	Sal	Banana	5461		South	Sravan	Apple	7682	
9	East	Hector	Apple	2341						
10	East	Xi	Banana	3234						
11	West	Amy	Banana	6532						
12	South	Sal	Pear	7323		Filtering performed on two Criteria				
13	North	Fred	Apple	2334		Region	Sales Rep	Product	Units	
14	North	Tom	Grape	8734		East	Tom	Apple	6380	
15	East	Hector	Grape	1932		East	Hector	Apple	2341	
16	South	Sravan	Apple	7682						
17	West	Xi	Grape	3293						

Using Code

The following example code demonstrates how the dynamic array functions are used in the spreadsheet.

C#

```
// For enabling Dynamic Array, you need to set CalcFeatures enumeration to DynamicArray
fpSpread1.AsWorkbook().WorkbookSet.CalculationEngine.CalcFeatures |=
CalcFeatures.DynamicArray;
fpSpread1.Sheets[0].FrozenRowCount = 1;
fpSpread1.Sheets[0].Cells[0, 0].Text = "Dynamic Array Functions";
fpSpread1.Sheets[0].Cells[0, 0].BackColor = System.Drawing.Color.LightGray;
fpSpread1.Sheets[0].AddSpanCell(0, 0, 1, 3);
// Setting Data in Cells of Sheet[0]
fpSpread1.Sheets[0].Cells[2, 0].Text = "Customer's Name";
fpSpread1.Sheets[0].Cells[2, 0].BackColor = System.Drawing.Color.LightGray;
fpSpread1.Sheets[0].Cells[3, 0].Text = "Larry";
fpSpread1.Sheets[0].Cells[4, 0].Text = "Safeway";
fpSpread1.Sheets[0].Cells[5, 0].Text = "Safeway";
fpSpread1.Sheets[0].Cells[6, 0].Text = "Raley";
fpSpread1.Sheets[0].Cells[7, 0].Text = "Vallarta";
fpSpread1.Sheets[0].Cells[8, 0].Text = "Safeway";
fpSpread1.Sheets[0].Cells[9, 0].Text = "Raley";
fpSpread1.Sheets[0].Cells[10, 0].Text = "Larry";
fpSpread1.Sheets[0].Cells[11, 0].Text = "Gilbert";
fpSpread1.Sheets[0].Cells[12, 0].Text = "Larry";
fpSpread1.Sheets[0].Cells[13, 0].Text = "Larry";
fpSpread1.Sheets[0].Cells[14, 0].Text = "Raley";
fpSpread1.Sheets[0].Columns[0].Width = 120;
fpSpread1.Sheets[0].Cells[2, 1].Text = "Age";
fpSpread1.Sheets[0].Cells[2, 1].BackColor = System.Drawing.Color.LightGray;
fpSpread1.Sheets[0].Cells[3, 1].Text = "32";
fpSpread1.Sheets[0].Cells[4, 1].Text = "23";
```

```
fpSpread1.Sheets[0].Cells[5, 1].Text = "23";
fpSpread1.Sheets[0].Cells[6, 1].Text = "39";
fpSpread1.Sheets[0].Cells[7, 1].Text = "18";
fpSpread1.Sheets[0].Cells[8, 1].Text = "23";
fpSpread1.Sheets[0].Cells[9, 1].Text = "39";
fpSpread1.Sheets[0].Cells[10, 1].Text = "32";
fpSpread1.Sheets[0].Cells[11, 1].Text = "19";
fpSpread1.Sheets[0].Cells[12, 1].Text = "32";
fpSpread1.Sheets[0].Cells[13, 1].Text = "32";
fpSpread1.Sheets[0].Cells[14, 1].Text = "39";
fpSpread1.Sheets[0].Columns[1].Width = 50;
// Setting "Unique" Formula
fpSpread1.Sheets[0].Cells[2, 2].Text = "Unique List";
fpSpread1.Sheets[0].Cells[2, 2].BackColor = System.Drawing.Color.LightBlue;
fpSpread1.Sheets[0].Cells[3, 2].Formula = "UNIQUE(A4:A15)";
fpSpread1.Sheets[0].Columns[2].Width = 90;
// Setting "Sort" Formula
fpSpread1.Sheets[0].Cells[2, 3].Text = "Sort";
fpSpread1.Sheets[0].Cells[2, 3].BackColor = System.Drawing.Color.LightBlue;
fpSpread1.Sheets[0].Cells[3, 3].Formula = "SORT(A4:A15)";
fpSpread1.Sheets[0].Columns[3].Width = 90;
// Setting "Sort" Formula for Unique list
fpSpread1.Sheets[0].Cells[2, 4].Text = "Sort Unique";
fpSpread1.Sheets[0].Cells[2, 4].BackColor = System.Drawing.Color.LightBlue;
fpSpread1.Sheets[0].Cells[3, 4].Formula = "SORT(C4#)";
fpSpread1.Sheets[0].Columns[4].Width = 90;
// Setting "Sort+Unique" Formula together
fpSpread1.Sheets[0].Cells[2, 5].Text = "Sort Unique";
fpSpread1.Sheets[0].Cells[2, 5].BackColor = System.Drawing.Color.LightBlue;
fpSpread1.Sheets[0].Cells[3, 5].Formula = "SORT(UNIQUE(A4:A15))";
fpSpread1.Sheets[0].Columns[5].Width = 90;
// Setting "SortBy" Formula wherein we sort Range A4:B15 based on the values in a
corresponding range B4:B15
fpSpread1.Sheets[0].Cells[2, 6].Text = "SortBy";
fpSpread1.Sheets[0].Cells[2, 6].BackColor = System.Drawing.Color.LightBlue;
fpSpread1.Sheets[0].Cells[3, 6].Formula = "SORTBY(A4:B15, B4:B15)";
fpSpread1.Sheets[0].Columns[6].Width = 90;
// Setting Data in Cells of Sheet[1]
fpSpread1.Sheets[1].Columns[0, 9].Width = 70;
fpSpread1.Sheets[1].Cells[3, 0].Text = "Region";
fpSpread1.Sheets[1].Cells[3, 0].BackColor = System.Drawing.Color.LightGray;
fpSpread1.Sheets[1].Cells[4, 0].Text = "East";
fpSpread1.Sheets[1].Cells[5, 0].Text = "North";
fpSpread1.Sheets[1].Cells[6, 0].Text = "Wast";
fpSpread1.Sheets[1].Cells[7, 0].Text = "Sast";
fpSpread1.Sheets[1].Cells[8, 0].Text = "East";
fpSpread1.Sheets[1].Cells[9, 0].Text = "East";
fpSpread1.Sheets[1].Cells[10, 0].Text = "West";
fpSpread1.Sheets[1].Cells[11, 0].Text = "South";
fpSpread1.Sheets[1].Cells[12, 0].Text = "North";
fpSpread1.Sheets[1].Cells[13, 0].Text = "North";
fpSpread1.Sheets[1].Cells[14, 0].Text = "East";
fpSpread1.Sheets[1].Cells[15, 0].Text = "South";
fpSpread1.Sheets[1].Cells[16, 0].Text = "West";
fpSpread1.Sheets[1].Cells[3, 1].Text = "Sales Rep";
fpSpread1.Sheets[1].Cells[3, 1].BackColor = System.Drawing.Color.LightGray;
fpSpread1.Sheets[1].Cells[4, 1].Text = "Tom";
```

```
fpSpread1.Sheets[1].Cells[5, 1].Text = "Fred";
fpSpread1.Sheets[1].Cells[6, 1].Text = "Amy";
fpSpread1.Sheets[1].Cells[7, 1].Text = "Sal";
fpSpread1.Sheets[1].Cells[8, 1].Text = "Hector";
fpSpread1.Sheets[1].Cells[9, 1].Text = "Xi";
fpSpread1.Sheets[1].Cells[10, 1].Text = "Amy";
fpSpread1.Sheets[1].Cells[11, 1].Text = "Sal";
fpSpread1.Sheets[1].Cells[12, 1].Text = "Fred";
fpSpread1.Sheets[1].Cells[13, 1].Text = "Tom";
fpSpread1.Sheets[1].Cells[14, 1].Text = "Hector";
fpSpread1.Sheets[1].Cells[15, 1].Text = "Sravan";
fpSpread1.Sheets[1].Cells[16, 1].Text = "Xi";
fpSpread1.Sheets[1].Cells[3, 2].Text = "Product";
fpSpread1.Sheets[1].Cells[3, 2].BackColor = System.Drawing.Color.LightGray;
fpSpread1.Sheets[1].Cells[4, 2].Text = "Apple";
fpSpread1.Sheets[1].Cells[5, 2].Text = "Grape";
fpSpread1.Sheets[1].Cells[6, 2].Text = "Pear";
fpSpread1.Sheets[1].Cells[7, 2].Text = "Banana";
fpSpread1.Sheets[1].Cells[8, 2].Text = "Apple";
fpSpread1.Sheets[1].Cells[9, 2].Text = "Banana";
fpSpread1.Sheets[1].Cells[10, 2].Text = "Banana";
fpSpread1.Sheets[1].Cells[11, 2].Text = "Pear";
fpSpread1.Sheets[1].Cells[12, 2].Text = "Apple";
fpSpread1.Sheets[1].Cells[13, 2].Text = "Grape";
fpSpread1.Sheets[1].Cells[14, 2].Text = "Grape";
fpSpread1.Sheets[1].Cells[15, 2].Text = "Apple";
fpSpread1.Sheets[1].Cells[16, 2].Text = "Grape";
fpSpread1.Sheets[1].Cells[3, 3].Text = "Units";
fpSpread1.Sheets[1].Cells[3, 3].BackColor = System.Drawing.Color.LightGray;
fpSpread1.Sheets[1].Cells[4, 3].Text = "6380";
fpSpread1.Sheets[1].Cells[5, 3].Text = "2344";
fpSpread1.Sheets[1].Cells[6, 3].Text = "3434";
fpSpread1.Sheets[1].Cells[7, 3].Text = "5461";
fpSpread1.Sheets[1].Cells[8, 3].Text = "2341";
fpSpread1.Sheets[1].Cells[9, 3].Text = "3234";
fpSpread1.Sheets[1].Cells[10, 3].Text = "6532";
fpSpread1.Sheets[1].Cells[11, 3].Text = "7323";
fpSpread1.Sheets[1].Cells[12, 3].Text = "2334";
fpSpread1.Sheets[1].Cells[13, 3].Text = "8734";
fpSpread1.Sheets[1].Cells[14, 3].Text = "1932";
fpSpread1.Sheets[1].Cells[15, 3].Text = "7682";
fpSpread1.Sheets[1].Cells[16, 3].Text = "3293";
fpSpread1.Sheets[1].Cells[0, 4].Text = "Product:";
fpSpread1.Sheets[1].Cells[0, 4].BackColor = System.Drawing.Color.LightGray;
fpSpread1.Sheets[1].Cells[0, 5].Text = "Apple";
fpSpread1.Sheets[1].Cells[1, 4].Text = "Region:";
fpSpread1.Sheets[1].Cells[1, 4].BackColor = System.Drawing.Color.LightGray;
fpSpread1.Sheets[1].Cells[1, 5].Text = "East";
fpSpread1.Sheets[1].Cells[2, 5].Text = "Filtering performed on one Criteria";
fpSpread1.Sheets[1].Cells[2, 5].BackColor = System.Drawing.Color.LightBlue;
fpSpread1.Sheets[1].AddSpanCell(2, 5, 1, 4);
fpSpread1.Sheets[1].Cells[3, 5].Text = "Region";
fpSpread1.Sheets[1].Cells[3, 5].BackColor = System.Drawing.Color.LightGray;
fpSpread1.Sheets[1].Cells[3, 6].Text = "Sales Rep";
fpSpread1.Sheets[1].Cells[3, 6].BackColor = System.Drawing.Color.LightGray;
fpSpread1.Sheets[1].Cells[3, 7].Text = "Product";
fpSpread1.Sheets[1].Cells[3, 7].BackColor = System.Drawing.Color.LightGray;
```

```

fpSpread1.Sheets[1].Cells[3, 8].Text = "Units";
fpSpread1.Sheets[1].Cells[3, 8].BackColor = System.Drawing.Color.LightGray;
/* Setting "Filter" Formula( with one condition) wherein we filter range A5:D17 based
upon criteria wherein range C5:C17 is equal to value in cell F1 */
fpSpread1.Sheets[1].Cells[4, 5].Formula = "FILTER(A5:D17, C5:C17=F1)";
fpSpread1.Sheets[1].Cells[12, 5].Text = "Region";
fpSpread1.Sheets[1].Cells[12, 5].BackColor = System.Drawing.Color.LightGray;
fpSpread1.Sheets[1].Cells[12, 6].Text = "Sales Rep";
fpSpread1.Sheets[1].Cells[12, 6].BackColor = System.Drawing.Color.LightGray;
fpSpread1.Sheets[1].Cells[12, 7].Text = "Product";
fpSpread1.Sheets[1].Cells[12, 7].BackColor = System.Drawing.Color.LightGray;
fpSpread1.Sheets[1].Cells[12, 8].Text = "Units";
fpSpread1.Sheets[1].Cells[12, 8].BackColor = System.Drawing.Color.LightGray;
fpSpread1.Sheets[1].Cells[11, 5].Text = "Filtering performed on two Criteria";
fpSpread1.Sheets[1].Cells[11, 5].BackColor = System.Drawing.Color.LightBlue;
fpSpread1.Sheets[1].AddSpanCell(11, 5, 1, 4);
/* Setting "Filter" Formula( with two conditions) wherein we filter range A5:D17 based
upon criteria wherein range C5:C17 is equal to value in cell F1
and range A5:A17 is equal to value in cell F2 */
fpSpread1.Sheets[1].Cells[13, 5].Formula = "FILTER(A5:D17, (C5:C17=F1)*(A5:A17=F2))";
fpSpread1.Sheets[2].Columns[0, 7].Width = 130;
// Setting "Sequence" Formula
fpSpread1.Sheets[2].Columns[0, 7].Width = 130;
fpSpread1.Sheets[2].Cells[0, 0].Text = "SEQUENCE(4,5) Function";
fpSpread1.Sheets[2].AddSpanCell(0, 0, 1, 2);
fpSpread1.Sheets[2].Cells[0, 0].BackColor = System.Drawing.Color.SkyBlue;
fpSpread1.Sheets[2].Cells[1, 0].Formula = "SEQUENCE(4,5)";
// Setting "RandArray" Formula
fpSpread1.Sheets[2].Cells[6, 0].Text = "RANDARRAY(5,3) Function";
fpSpread1.Sheets[2].AddSpanCell(6, 0, 1, 2);
fpSpread1.Sheets[2].Cells[6, 0].BackColor = System.Drawing.Color.SkyBlue;
fpSpread1.Sheets[2].Cells[7, 0].Formula = "RANDARRAY(5,3)";
// Setting "Single" Formula
fpSpread1.Sheets[2].Cells[13, 0].Text = "SINGLE(A15:E15) Function";
fpSpread1.Sheets[2].AddSpanCell(13, 0, 1, 2);
fpSpread1.Sheets[2].Cells[13, 0].BackColor = System.Drawing.Color.SkyBlue;
fpSpread1.Sheets[2].Cells[14, 0].Value = "A";
fpSpread1.Sheets[2].Cells[14, 1].Value = "B";
fpSpread1.Sheets[2].Cells[14, 2].Value = "C";
fpSpread1.Sheets[2].Cells[14, 3].Value = "D";
fpSpread1.Sheets[2].Cells[14, 4].Value = "E";
fpSpread1.Sheets[2].Cells[15, 2].Formula = "SINGLE(A15:E15)";

```

VB

```

' For enabling Dynamic Array, you need to set CalcFeatures enumeration to DynamicArray
FpSpread1.AsWorkbook().WorkbookSet.CalculationEngine.CalcFeatures =
FpSpread1.AsWorkbook().WorkbookSet.CalculationEngine.CalcFeatures Or
CalcFeatures.DynamicArray
FpSpread1.Sheets(0).FrozenRowCount = 1
FpSpread1.Sheets(0).Cells(0, 0).Text = "Dynamic Array Functions"
FpSpread1.Sheets(0).Cells(0, 0).BackColor = System.Drawing.Color.LightGray
FpSpread1.Sheets(0).AddSpanCell(0, 0, 1, 3)
' Setting Data in Cells of Sheets(0)
FpSpread1.Sheets(0).Cells(2, 0).Text = "Customer's Name"
FpSpread1.Sheets(0).Cells(2, 0).BackColor = System.Drawing.Color.LightGray

```

```

FpSpread1.Sheets(0).Cells(3, 0).Text = "Larry"
FpSpread1.Sheets(0).Cells(4, 0).Text = "Safeway"
FpSpread1.Sheets(0).Cells(5, 0).Text = "Safeway"
FpSpread1.Sheets(0).Cells(6, 0).Text = "Raley"
FpSpread1.Sheets(0).Cells(7, 0).Text = "Vallarta"
FpSpread1.Sheets(0).Cells(8, 0).Text = "Safeway"
FpSpread1.Sheets(0).Cells(9, 0).Text = "Raley"
FpSpread1.Sheets(0).Cells(10, 0).Text = "Larry"
FpSpread1.Sheets(0).Cells(11, 0).Text = "Gilbert"
FpSpread1.Sheets(0).Cells(12, 0).Text = "Larry"
FpSpread1.Sheets(0).Cells(13, 0).Text = "Larry"
FpSpread1.Sheets(0).Cells(14, 0).Text = "Raley"
FpSpread1.Sheets(0).Columns(0).Width = 120
FpSpread1.Sheets(0).Cells(2, 1).Text = "Age"
FpSpread1.Sheets(0).Cells(2, 1).BackColor = System.Drawing.Color.LightGray
FpSpread1.Sheets(0).Cells(3, 1).Text = "32"
FpSpread1.Sheets(0).Cells(4, 1).Text = "23"
FpSpread1.Sheets(0).Cells(5, 1).Text = "23"
FpSpread1.Sheets(0).Cells(6, 1).Text = "39"
FpSpread1.Sheets(0).Cells(7, 1).Text = "18"
FpSpread1.Sheets(0).Cells(8, 1).Text = "23"
FpSpread1.Sheets(0).Cells(9, 1).Text = "39"
FpSpread1.Sheets(0).Cells(10, 1).Text = "32"
FpSpread1.Sheets(0).Cells(11, 1).Text = "19"
FpSpread1.Sheets(0).Cells(12, 1).Text = "32"
FpSpread1.Sheets(0).Cells(13, 1).Text = "32"
FpSpread1.Sheets(0).Cells(14, 1).Text = "39"
FpSpread1.Sheets(0).Columns(1).Width = 50
' Setting "Unique" Formula
FpSpread1.Sheets(0).Cells(2, 2).Text = "Unique List"
FpSpread1.Sheets(0).Cells(2, 2).BackColor = System.Drawing.Color.LightBlue
FpSpread1.Sheets(0).Cells(3, 2).Formula = "UNIQUE(A4:A15)"
FpSpread1.Sheets(0).Columns(2).Width = 90
' Using "Sort" Formula
FpSpread1.Sheets(0).Cells(2, 3).Text = "Sort"
FpSpread1.Sheets(0).Cells(2, 3).BackColor = System.Drawing.Color.LightBlue
FpSpread1.Sheets(0).Cells(3, 3).Formula = "SORT(A4:A15)"
FpSpread1.Sheets(0).Columns(3).Width = 90
' Setting "Sort" Formula for Unique list
FpSpread1.Sheets(0).Cells(2, 4).Text = "Sort Unique"
FpSpread1.Sheets(0).Cells(2, 4).BackColor = System.Drawing.Color.LightBlue
FpSpread1.Sheets(0).Cells(3, 4).Formula = "SORT(C4#)"
FpSpread1.Sheets(0).Columns(4).Width = 90
' Setting "Sort+Unique" Formula together
FpSpread1.Sheets(0).Cells(2, 5).Text = "Sort Unique"
FpSpread1.Sheets(0).Cells(2, 5).BackColor = System.Drawing.Color.LightBlue
FpSpread1.Sheets(0).Cells(3, 5).Formula = "SORT(UNIQUE(A4:A15))"
FpSpread1.Sheets(0).Columns(5).Width = 90
' Setting "SortBy" Formula wherein we sort Range A4:B15 based on the values in a
corresponding range B4:B15
FpSpread1.Sheets(0).Cells(2, 6).Text = "SortBy"
FpSpread1.Sheets(0).Cells(2, 6).BackColor = System.Drawing.Color.LightBlue
FpSpread1.Sheets(0).Cells(3, 6).Formula = "SORTBY(A4:B15, B4:B15)"
FpSpread1.Sheets(0).Columns(6).Width = 90
FpSpread1.Sheets(1).Columns(0, 9).Width = 70
FpSpread1.Sheets(1).Cells(3, 0).Text = "Region"
FpSpread1.Sheets(1).Cells(3, 0).BackColor = System.Drawing.Color.LightGray

```

```
FpSpread1.Sheets(1).Cells(4, 0).Text = "East"
FpSpread1.Sheets(1).Cells(5, 0).Text = "North"
FpSpread1.Sheets(1).Cells(6, 0).Text = "West"
FpSpread1.Sheets(1).Cells(7, 0).Text = "Sast"
FpSpread1.Sheets(1).Cells(8, 0).Text = "East"
FpSpread1.Sheets(1).Cells(9, 0).Text = "East"
FpSpread1.Sheets(1).Cells(10, 0).Text = "West"
FpSpread1.Sheets(1).Cells(11, 0).Text = "South"
FpSpread1.Sheets(1).Cells(12, 0).Text = "North"
FpSpread1.Sheets(1).Cells(13, 0).Text = "North"
FpSpread1.Sheets(1).Cells(14, 0).Text = "East"
FpSpread1.Sheets(1).Cells(15, 0).Text = "South"
FpSpread1.Sheets(1).Cells(16, 0).Text = "West"
FpSpread1.Sheets(1).Cells(3, 1).Text = "Sales Rep"
FpSpread1.Sheets(1).Cells(3, 1).BackColor = System.Drawing.Color.LightGray
FpSpread1.Sheets(1).Cells(4, 1).Text = "Tom"
FpSpread1.Sheets(1).Cells(5, 1).Text = "Fred"
FpSpread1.Sheets(1).Cells(6, 1).Text = "Amy"
FpSpread1.Sheets(1).Cells(7, 1).Text = "Sal"
FpSpread1.Sheets(1).Cells(8, 1).Text = "Hector"
FpSpread1.Sheets(1).Cells(9, 1).Text = "Xi"
FpSpread1.Sheets(1).Cells(10, 1).Text = "Amy"
FpSpread1.Sheets(1).Cells(11, 1).Text = "Sal"
FpSpread1.Sheets(1).Cells(12, 1).Text = "Fred"
FpSpread1.Sheets(1).Cells(13, 1).Text = "Tom"
FpSpread1.Sheets(1).Cells(14, 1).Text = "Hector"
FpSpread1.Sheets(1).Cells(15, 1).Text = "Sravan"
FpSpread1.Sheets(1).Cells(16, 1).Text = "Xi"
FpSpread1.Sheets(1).Cells(3, 2).Text = "Product"
FpSpread1.Sheets(1).Cells(3, 2).BackColor = System.Drawing.Color.LightGray
FpSpread1.Sheets(1).Cells(4, 2).Text = "Apple"
FpSpread1.Sheets(1).Cells(5, 2).Text = "Grape"
FpSpread1.Sheets(1).Cells(6, 2).Text = "Pear"
FpSpread1.Sheets(1).Cells(7, 2).Text = "Banana"
FpSpread1.Sheets(1).Cells(8, 2).Text = "Apple"
FpSpread1.Sheets(1).Cells(9, 2).Text = "Banana"
FpSpread1.Sheets(1).Cells(10, 2).Text = "Banana"
FpSpread1.Sheets(1).Cells(11, 2).Text = "Pear"
FpSpread1.Sheets(1).Cells(12, 2).Text = "Apple"
FpSpread1.Sheets(1).Cells(13, 2).Text = "Grape"
FpSpread1.Sheets(1).Cells(14, 2).Text = "Grape"
FpSpread1.Sheets(1).Cells(15, 2).Text = "Apple"
FpSpread1.Sheets(1).Cells(16, 2).Text = "Grape"
FpSpread1.Sheets(1).Cells(3, 3).Text = "Units"
FpSpread1.Sheets(1).Cells(3, 3).BackColor = System.Drawing.Color.LightGray
FpSpread1.Sheets(1).Cells(4, 3).Text = "6380"
FpSpread1.Sheets(1).Cells(5, 3).Text = "2344"
FpSpread1.Sheets(1).Cells(6, 3).Text = "3434"
FpSpread1.Sheets(1).Cells(7, 3).Text = "5461"
FpSpread1.Sheets(1).Cells(8, 3).Text = "2341"
FpSpread1.Sheets(1).Cells(9, 3).Text = "3234"
FpSpread1.Sheets(1).Cells(10, 3).Text = "6532"
FpSpread1.Sheets(1).Cells(11, 3).Text = "7323"
FpSpread1.Sheets(1).Cells(12, 3).Text = "2334"
FpSpread1.Sheets(1).Cells(13, 3).Text = "8734"
FpSpread1.Sheets(1).Cells(14, 3).Text = "1932"
FpSpread1.Sheets(1).Cells(15, 3).Text = "7682"
```

```

FpSpread1.Sheets(1).Cells(16, 3).Text = "3293"
FpSpread1.Sheets(1).Cells(0, 4).Text = "Product:"
FpSpread1.Sheets(1).Cells(0, 4).BackColor = System.Drawing.Color.LightGray
FpSpread1.Sheets(1).Cells(0, 5).Text = "Apple"
FpSpread1.Sheets(1).Cells(1, 4).Text = "Region:"
FpSpread1.Sheets(1).Cells(1, 4).BackColor = System.Drawing.Color.LightGray
FpSpread1.Sheets(1).Cells(1, 5).Text = "East"
FpSpread1.Sheets(1).Cells(2, 5).Text = "Filtering performed on one Criteria"
FpSpread1.Sheets(1).Cells(2, 5).BackColor = System.Drawing.Color.LightBlue
FpSpread1.Sheets(1).AddSpanCell(2, 5, 1, 4)
FpSpread1.Sheets(1).Cells(3, 5).Text = "Region"
FpSpread1.Sheets(1).Cells(3, 5).BackColor = System.Drawing.Color.LightGray
FpSpread1.Sheets(1).Cells(3, 6).Text = "Sales Rep"
FpSpread1.Sheets(1).Cells(3, 6).BackColor = System.Drawing.Color.LightGray
FpSpread1.Sheets(1).Cells(3, 7).Text = "Product"
FpSpread1.Sheets(1).Cells(3, 7).BackColor = System.Drawing.Color.LightGray
FpSpread1.Sheets(1).Cells(3, 8).Text = "Units"
FpSpread1.Sheets(1).Cells(3, 8).BackColor = System.Drawing.Color.LightGray

```

```

' Setting "Filter" Formula( with one condition) wherein we filter range A5:D17 based
upon 'criteria wherein Range C5: C17 is equal to value in cell F1

```

```

FpSpread1.Sheets(1).Cells(4, 5).Formula = "FILTER(A5:D17, C5:C17=F1)"
FpSpread1.Sheets(1).Cells(12, 5).Text = "Region"
FpSpread1.Sheets(1).Cells(12, 5).BackColor = System.Drawing.Color.LightGray
FpSpread1.Sheets(1).Cells(12, 6).Text = "Sales Rep"
FpSpread1.Sheets(1).Cells(12, 6).BackColor = System.Drawing.Color.LightGray
FpSpread1.Sheets(1).Cells(12, 7).Text = "Product"
FpSpread1.Sheets(1).Cells(12, 7).BackColor = System.Drawing.Color.LightGray
FpSpread1.Sheets(1).Cells(12, 8).Text = "Units"
FpSpread1.Sheets(1).Cells(12, 8).BackColor = System.Drawing.Color.LightGray
FpSpread1.Sheets(1).Cells(11, 5).Text = "Filtering performed on two Criteria"
FpSpread1.Sheets(1).Cells(11, 5).BackColor = System.Drawing.Color.LightBlue
FpSpread1.Sheets(1).AddSpanCell(11, 5, 1, 4)

```

```

' Setting "Filter" Formula( with two conditions) wherein we filter range A5:D17 based
upon 'criteria wherein Range C5: C17 is equal to value in cell F1 and range A5:A17 is
equal to 'value in cell F2

```

```

FpSpread1.Sheets(1).Cells(13, 5).Formula = "FILTER(A5:D17, (C5:C17=F1)*(A5:A17=F2))"
FpSpread1.Sheets(2).Columns(0, 7).Width = 130

```

```

' Setting "Sequence" Formula

```

```

FpSpread1.Sheets(2).Columns(0, 7).Width = 130
FpSpread1.Sheets(2).Cells(0, 0).Text = "SEQUENCE(4,5) Function"
FpSpread1.Sheets(2).AddSpanCell(0, 0, 1, 2)
FpSpread1.Sheets(2).Cells(0, 0).BackColor = System.Drawing.Color.SkyBlue
FpSpread1.Sheets(2).Cells(1, 0).Formula = "SEQUENCE(4,5)"

```

```

' Setting "RandArray" Formula

```

```

FpSpread1.Sheets(2).Cells(6, 0).Text = "RANDARRAY(5,3) Function"
FpSpread1.Sheets(2).AddSpanCell(6, 0, 1, 2)
FpSpread1.Sheets(2).Cells(6, 0).BackColor = System.Drawing.Color.SkyBlue
FpSpread1.Sheets(2).Cells(7, 0).Formula = "RANDARRAY(5,3)"

```

```

' Setting "Single" Formula

```

```

FpSpread1.Sheets(2).Cells(13, 0).Text = "SINGLE(A15:E15) Function"

```

```

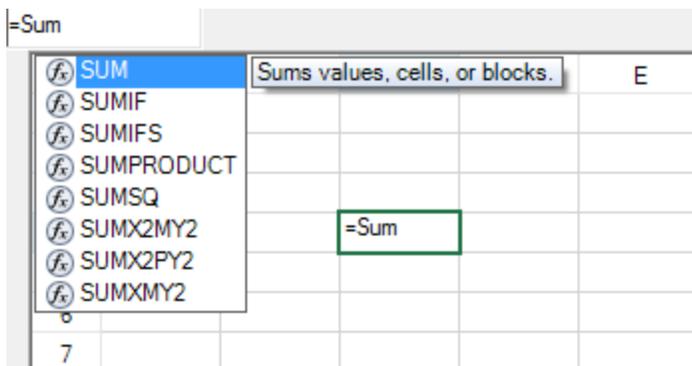
FpSpread1.Sheets(2).AddSpanCell(13, 0, 1, 2)
FpSpread1.Sheets(2).Cells(13, 0).BackColor = System.Drawing.Color.SkyBlue
FpSpread1.Sheets(2).Cells(14, 0).Value = "A"
FpSpread1.Sheets(2).Cells(14, 1).Value = "B"
FpSpread1.Sheets(2).Cells(14, 2).Value = "C"
FpSpread1.Sheets(2).Cells(14, 3).Value = "D"
FpSpread1.Sheets(2).Cells(14, 4).Value = "E"
FpSpread1.Sheets(2).Cells(15, 2).Formula = "SINGLE (A15:E15) "

```

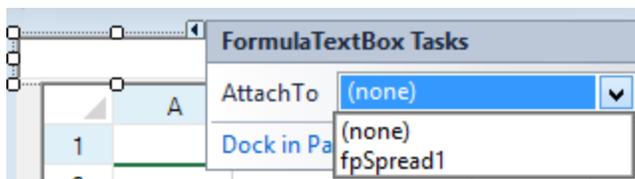
Working with the Formula Text Box

Setting up the Formula Text Box

You can set up a floating formula bar that can be used to add formulas. The formula bar is similar to the formula editor available to the developer and has the appearance of a text box. The formula bar not only renders a list of calculation functions but also provides a visual method of selecting cell ranges for the formula.



In order to set up the formula bar at run time, you can use the **FormulaTextBox** ('**FormulaTextBox Class**' in the **on-line documentation**) class. You can also draw the formula text box on the form and assign it to Spread at design time. Select the formula text box icon in the **Toolbox** and drag it to the form. Select the formula text box verb and attach it to Spread.

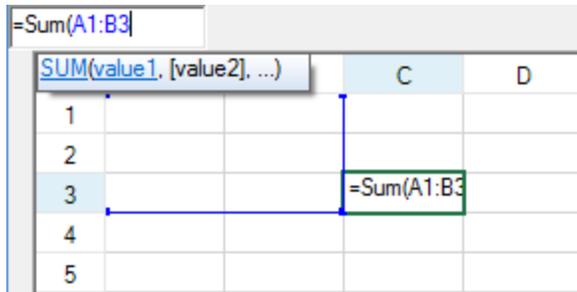


The **AllowUserFormulas** ('**AllowUserFormulas Property**' in the **on-line documentation**) property allows the user to type formulas in the cell in the Spread control.

If you set the **AllowUserFormulas** property to True, then the formulas that are typed in a cell will show up in the formula bar.

Using the Formula Text Box

To use the formula text box, type the equal sign (=) and then start typing the name of the formula. This brings up a list of functions that start with that letter. You can then type the left parenthesis and either select a block of cells by dragging the mouse over that range or type cell values by absolute or relative reference. The figure below shows the selection of a range of cells from A1 to B3.



Using Code

Create the formula editor and attach it to the control.

Example

This example code creates the floating formula bar.

C#

```
FarPoint.Win.Spread.FormulaTextBox editor = new FarPoint.Win.Spread.FormulaTextBox();
editor.Location = new Point(0, 0);
editor.Size = new Size(80, 20);
this.Controls.Add(editor);
editor.Attach(fpSpread1);
// This line will disconnect the formula bar from the control
// editor.Detach();
```

VB

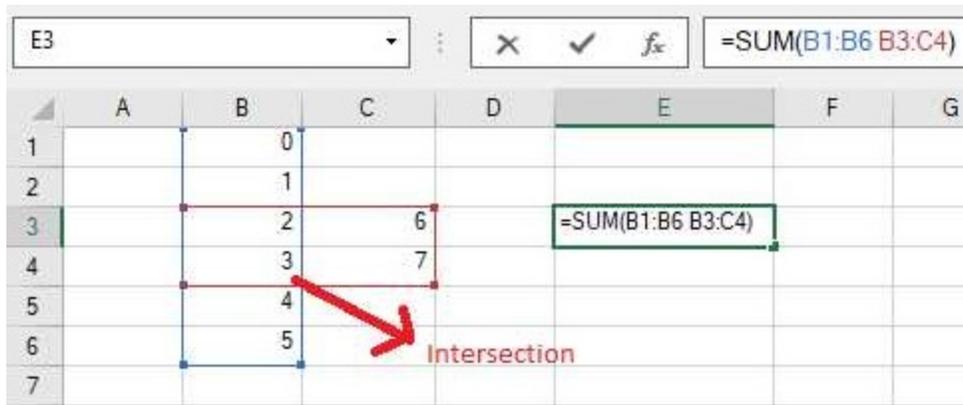
```
Dim editor As New FarPoint.Win.Spread.FormulaTextBox
editor.Location = New Point(0, 0)
editor.Size = New Size(80, 20)
Controls.Add(editor)
editor.Attach(fpSpread1)
` This line will disconnect the formula bar from the control
` editor.Detach()
```

Using Intersect Formula and Mixed Reference Formula

You can use the intersect formula and the mixed reference formula while working with formula text box in the spreadsheets.

In order to create an intersect formula in a worksheet, users need to select or provide two cell ranges separated by spaces as parameters of the calculation function that is being used.

An example screenshot shared below depicts the intersection formula used in a formula text box for SUM function containing two cell ranges - B1:B6 and B3:C4 separated by the space character. When the formula is calculated, it returns the evaluated sum of all the values appearing in the intersection area (an area where rows and columns intersect as highlighted in the image) of the two cell ranges.



A mixed reference formula refers to the combination of relative and absolute cell references (absolute column and relative row or absolute row and relative column) used in a worksheet. The absolute cell references are also known as fixed references and are represented by the cells with the dollar symbol (\$) placed in front of them. The relative cell references change when the formula is dragged or copied across rows and columns in the worksheet.

For more information on formulas, refer to **Managing Formulas in Cells** and the [Formula Reference](#).

Using Code

You can use the intersect formula and the mixed reference formula in the formula text box in the spreadsheet.

Example

This example code shows how to work with intersect formula and mixed reference formula in the spreadsheet.

C#

```
// Using intersect formula
fpSpread1.Sheets[0].Cells[0, 1].Value = 0;
fpSpread1.Sheets[0].Cells[1, 1].Value = 1;
fpSpread1.Sheets[0].Cells[2, 1].Value = 2;
fpSpread1.Sheets[0].Cells[3, 1].Value = 3;
fpSpread1.Sheets[0].Cells[4, 1].Value = 4;
fpSpread1.Sheets[0].Cells[5, 1].Value = 5;
fpSpread1.Sheets[0].Cells[2, 2].Value = 6;
fpSpread1.Sheets[0].Cells[3, 2].Value = 7;
fpSpread1.Sheets[0].Cells[2, 4].Formula = "SUM(B1:B6 B3:C4)";

// Using mixed reference formula
fpSpread1.Sheets[0].Cells[5, 5].Formula = "SUM($B1, $B$2, B$3, B4)";
```

VB

```
'Using intersect formula
fpSpread1.Sheets(0).Cells(0, 1).Value = 0
fpSpread1.Sheets(0).Cells(1, 1).Value = 1
fpSpread1.Sheets(0).Cells(2, 1).Value = 2
fpSpread1.Sheets(0).Cells(3, 1).Value = 3
fpSpread1.Sheets(0).Cells(4, 1).Value = 4
fpSpread1.Sheets(0).Cells(5, 1).Value = 5
fpSpread1.Sheets(0).Cells(2, 2).Value = 6
fpSpread1.Sheets(0).Cells(3, 2).Value = 7
fpSpread1.Sheets(0).Cells(2, 4).Formula = "SUM(B1:B6 B3:C4)"
```

```
'Using mixed reference formula
fpSpread1.Sheets(0).Cells(5, 5).Formula = "SUM($B1, $B$2, B$3, B4)"
```

Selecting Table Formula using Structured References

Spread Winforms provides support for inserting structured reference formulas in table cells. The structured reference formula uses keywords and the column name of the table to refer to cell ranges in the table.

The components of a structured reference in a table formula are as follows:

1. Table Name - A table name is a meaningful name that you provide to reference the actual table data (excluding the headers and totals row, if any).
2. Column Specifier - This is derived from the column header and is enclosed in brackets. The column specifier references the column data (excluding the column header and total, if any).
3. Special Item Specifier - This can be used to refer to specific portions of the table, such as the Totals row.
4. Table specifier - This is the outer portion of the structured reference that is enclosed in square brackets following the table name.
5. Structured Reference - A structured reference is the entire string beginning with the table name and ending with the table specifier.

The following image depicts how to select table formula using structured references while working in Spread Designer.

	A	B	C	D	E	F	G	H
1								
2	Company Name	Q1	Q2	Q3				
3	Gadlco	4343	3434	43543				
4	Sam	3443	3331	3434				
5	Tim Pvt. Ltd.	6531	2324	4334				
6								
7								
8								
9								

Setting up the Formula Provider

You can add a formula provider control to the form. The provider control can be added to the **Toolbox**. Find the formula provider control in the list of .NET controls that can be added to the **Toolbox** and select it.



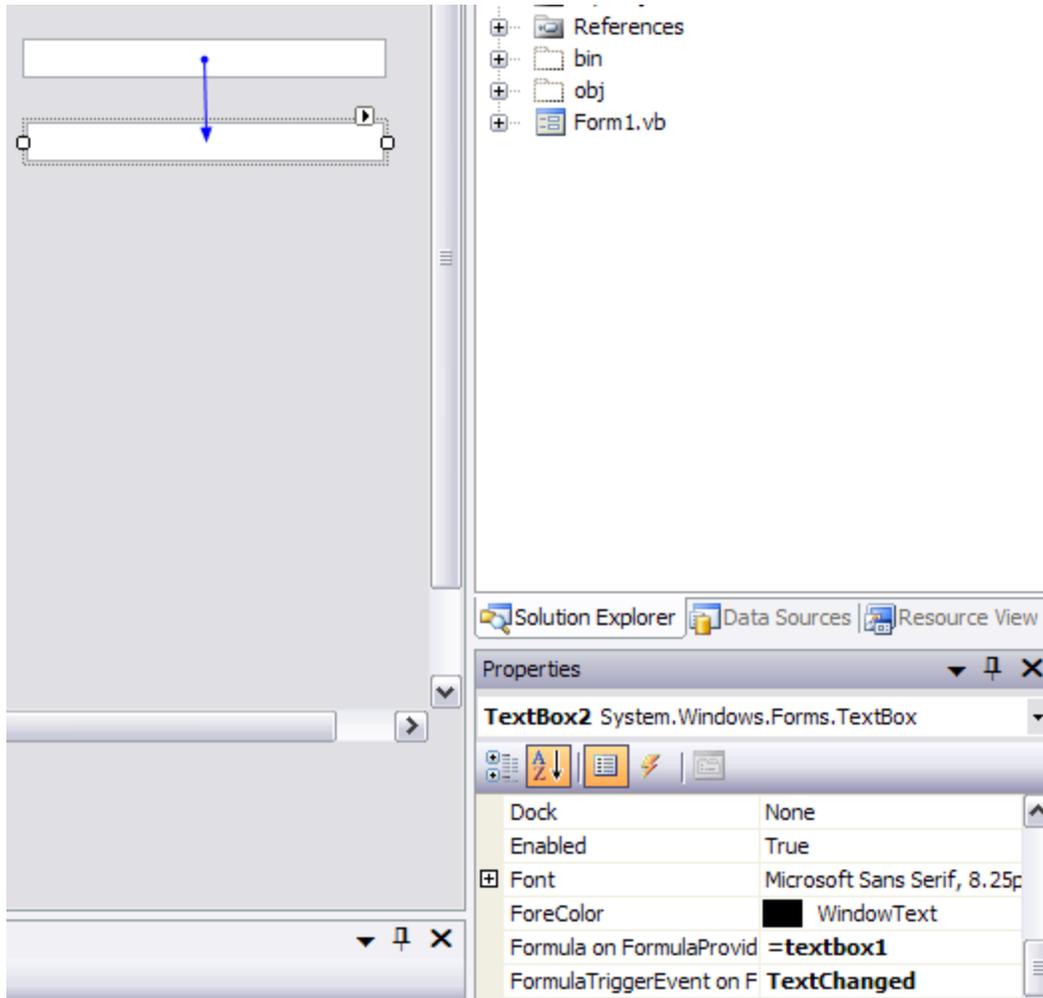
Setting up the Formula Provider Control

Double-click the formula provider control after selecting it from the **Toolbox**.

Two design properties will be added to each control on the form. The **Formula on Formula Provider** property is the property for setting a formula. The **Formula Trigger Event on Formula Provider** property allows you to determine which event will cause the formula to update.

Using the Formula Provider

The following basic example shows how to use the formula provider at design time. This example uses two text box controls and the formula provider. The second text box displays the value from the first text box. The formula for the second text box (**Formula on Formula Provider** property) has been set to be equal to the first text box. The trigger event (**FormulaTriggerEvent**) has been set to the **TextChanged** event.



For more information on formulas, refer to **Formulas in Cells** and the [Formula Reference](#).

Using Code

Add the formula provider and two text box controls to the form and set the **SetFormula** (**SetFormula Method** in **the on-line documentation**) and **TextChanged** Event methods for the formula provider.

Example

This example gets the typed data from text box 1 and puts it in text box 2.

C#

```
FormulaProvider1.SetFormula(TextBox2, "=TextBox1");  
FormulaProvider1.SetFormulaTriggerEvent(TextBox1, "TextChanged");
```

VB

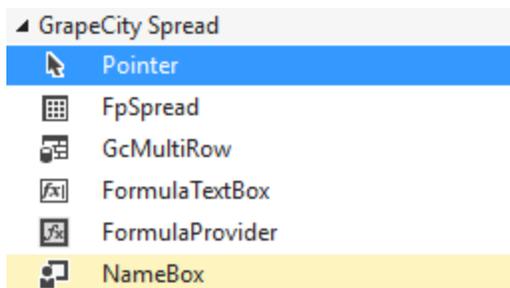
```
FormulaProvider1.SetFormula(TextBox2, "=TextBox1")  
FormulaProvider1.SetFormulaTriggerEvent(TextBox1, "TextChanged")
```

Setting up the Name Box

You can use the name box control to display or create custom names at run time. Custom names that are created by the name box can only refer to a cell or a range of cells.

Setting up the Name Box

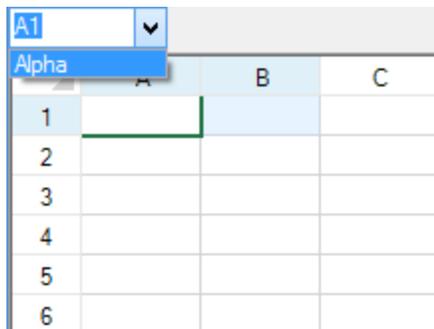
Select the name box control from the **Toolbox** and draw it on the form or use the **NameBox ('NameBox Class' in the on-line documentation)** class to create the control at runtime. Then attach the control to Spread. The following image displays the NameBox control in the toolbox.



Using the Name Box

To create a custom name, select a cell or range of cells in the Spread control, type a custom name in the name box control, and then press the equal key to create the custom name.

Use the drop-down list in the name box control to display the custom names. You can select one of the names to see the cell or cell range that the name refers to.



Using Code

Create the name box and attach it to the control.

Example

This example code creates a name box control and a custom name.

C#

```
fpSpread1.ActiveSheet.AddCustomName("Alpha", "A1:B1", 0, 0);

        FarPoint.Win.Spread.NameBox namebox1 = new
FarPoint.Win.Spread.NameBox();
        namebox1.Location = new Point(0, 0);
        namebox1.Size = new Size(80, 20);
        this.Controls.Add(namebox1);
        namebox1.Attach(fpSpread1);
```

VB

```
fpSpread1.ActiveSheet.AddCustomName("Alpha", "A1:B1", 0, 0)

        Dim namebox1 As New FarPoint.Win.Spread.NameBox()
        namebox1.Location = New Point(0, 0)
        namebox1.Size = New Size(80, 20)
        Controls.Add(namebox1)
        namebox1.Attach(fpSpread1)
```

Using Language Package

Spread for WinForms provides additional language packages in order to enable users to set language preferences while working with spreadsheets. Besides this, it facilitates users to localize functions names and formula keywords via creating function alias in their native languages and managing worksheets without any hassle.

In order to use language package, refer to the following tasks:

- **Available Language Packages for WinForms**
- **Creating and Using a Custom Language Package**

Available Language Packages for WinForms

While working with spreadsheets, users can make use of the eighteen available built-in language packages in order to localize function names and formula keywords; set up the display of the worksheet and load screentips in their preferred language.

A custom function name created for a new language can also be referred to as function alias. Each language has a specific language package corresponding to it. By default, Spread uses the English language package. If users want to use the Spread component in their native language, they can either choose from the available list of built-in language packages or create their own language package with function alias representing the same formula logic but in another language of their choice.

Spread for WinForms supports localization for the following languages:

1. Chinese
2. Czech
3. Danish
4. Dutch
5. Finnish
6. French
7. German
8. Hungarian
9. Italian
10. Japanese

11. Korean
12. Norwegian
13. Polish
14. Portuguese
15. Russian
16. Spanish
17. Swedish
18. Turkish

 **Note:** The default language package is in English language and this will be used as the fallback scenario.

Using Code

Refer to the following example code in order to use the available language package in a spreadsheet.

C#

```
// Assigning Formula

fpSpread1.Sheets[0].Cells[0, 0].Formula = "Sum(1)";

// Setting Spanish language package
fpSpread1.AsWorkbook().WorkbookSet.LanguagePackage =
LanguagePackage.Spanish;

// Now we get formula in Spanish Language
MessageBox.Show(fpSpread1.Sheets[0].Cells[0,
0].Formula.ToString());
```

VB

```
'Assigning Formula

fpSpread1.Sheets(0).Cells(0, 0).Formula = "Sum(1)"

'Setting Spanish language package
fpSpread1.AsWorkbook().WorkbookSet.LanguagePackage =
LanguagePackage.Spanish

'Now we get formula in Spanish Language
MessageBox.Show(fpSpread1.Sheets(0).Cells(0, 0).Formula.ToString())
```

Creating and Using a Custom Language Package

Spread for WinForms provides a set of eighteen language packages including support for Chinese, Czech, Danish, Dutch, Finnish, French, German, Hungarian, Italian, Japanese, Korean, Norwegian, Polish, Portuguese, Russian, Spanish, Swedish and Turkish languages.

In case, your preferred language doesn't come under the list of the available built-in packages, you can create a custom language package as per your requirements.

Using Code

In order to create and use a custom language package in your spreadsheet, you can follow the steps shared below:

Step 1 - Create the custom language package with function alias.

C#

```
// Creating a custom French language package with functions alias, and mapping
BuiltinFunction and
StructuredItemSpecifier.
private LanguagePackage CreateCustomFrenchLanguagePackage()
{
    LanguagePackage languagePackage = new LanguagePackage("CustomFrench", "français");
    languagePackage.CreateFunctionAlias(BuiltinFunction.SUM, "SOMME");
    languagePackage.CreateFunctionAlias(BuiltinFunction.SUBTOTAL, "SOUS.TOTAL");
    languagePackage.MapAlias(StructuredItemSpecifiers.Headers, "En-têtes");
    languagePackage.MapAlias(StructuredItemSpecifiers.Totals, "Totaux");
    return languagePackage;
}
```

VB

```
Private Function CreateCustomFrenchLanguagePackage() As LanguagePackage
' Creating a custom French language package with functions alias, and mapping
BuiltinFunction and StructuredItemSpecifier.

    LanguagePackage languagePackage = new LanguagePackage("CustomFrench", "français")
    languagePackage.CreateFunctionAlias(BuiltinFunction.SUM, "SOMME")
    languagePackage.CreateFunctionAlias(BuiltinFunction.SUBTOTAL, "SOUS.TOTAL")
    languagePackage.MapAlias(StructuredItemSpecifiers.Headers, "En-têtes")
    languagePackage.MapAlias(StructuredItemSpecifiers.Totals, "Totaux")
    return languagePackage

End Function
```

Step 2 - Assign the custom language using LanguagePackage property.

C#

```
// Assigning custom language using LanguagePackage property.
fpSpread1.AsWorkbook().WorkbookSet.LanguagePackage =
CreateCustomFrenchLanguagePackage();
```

VB

```
'Assigning custom language using LanguagePackage property.
fpSpread1.AsWorkbook().WorkbookSet.LanguagePackage =
CreateCustomFrenchLanguagePackage()
```

Step 3 - Use the custom language package, set the formulas in the spreadsheet and calculate cell values.

C#

```
//Setting cell texts and formulas.
fpSpread1.Sheets[0].Cells[1, 1].Text = "Last Name";
fpSpread1.Sheets[0].Cells[1, 2].Text = "Value";
fpSpread1.Sheets[0].Cells[2, 1].Text = "Smith";
fpSpread1.Sheets[0].Cells[2, 2].Value = 50;
```

```

fpSpread1.Sheets[0].Cells[3, 1].Text = "Vil";
fpSpread1.Sheets[0].Cells[3, 2].Value = 10;
fpSpread1.Sheets[0].Cells[4, 1].Text = "Press";
fpSpread1.Sheets[0].Cells[4, 2].Value = 78;

// Adding a table.
TableView t = fpSpread1.Sheets[0].AddTable("table", 1, 1, 5, 2);
t.TotalRowVisible = true;
fpSpread1.ActiveSheet.Cells[0, 1].Formula = "\"Total of \"&table[[#En-têtes],[Value]]\"";
//Use localized keyword
fpSpread1.ActiveSheet.Cells[0, 1].ColumnSpan = 2;
fpSpread1.ActiveSheet.Cells[0, 3].Formula = "table[[#Totals],[Value]]"; //Use English
keyword
fpSpread1.ActiveSheet.Cells[7, 1].ColumnSpan = 2;
fpSpread1.ActiveSheet.SetFormula(0, 0, "SUM(2,120)");
fpSpread1.ActiveSheet.SetActiveCell(0, 3);

```

VB

```

'Setting cell texts and formulas.
fpSpread1.Sheets[0].Cells[1, 1].Text = "Last Name"
fpSpread1.Sheets[0].Cells[1, 2].Text = "Value"
fpSpread1.Sheets[0].Cells[2, 1].Text = "Smith"
fpSpread1.Sheets[0].Cells[2, 2].Value = 50
fpSpread1.Sheets[0].Cells[3, 1].Text = "Vil"
fpSpread1.Sheets[0].Cells[3, 2].Value = 10
fpSpread1.Sheets[0].Cells[4, 1].Text = "Press"
fpSpread1.Sheets[0].Cells[4, 2].Value = 78

'Adding a table.
TableView t = fpSpread1.Sheets[0].AddTable("table", 1, 1, 5, 2)
t.TotalRowVisible = true
fpSpread1.ActiveSheet.Cells[0, 1].Formula = "\"Total of \"&table[[#En-têtes],[Value]]\"";
'Use localized keyword
fpSpread1.ActiveSheet.Cells[0, 1].ColumnSpan = 2
fpSpread1.ActiveSheet.Cells[0, 3].Formula = "table[[#Totals],[Value]]"; 'Use English
keyword
fpSpread1.ActiveSheet.Cells[7, 1].ColumnSpan = 2
fpSpread1.ActiveSheet.SetFormula(0, 0, "SUM(2,120)");
fpSpread1.ActiveSheet.SetActiveCell(0, 3)

```

Accessing Data from Header or Footer

You can add formulas to headers and footers while performing spreadsheet calculations. These formulas can contain references to a cell, cell range, other worksheets, or to itself. In addition to this, the data available in the header or footer can be accessed from any worksheet within the workbook in order to ensure it can be used anywhere in the formulas as and when required.

Different keywords are supported for different regions of a worksheet. These keywords are used in formulas to retrieve data available in specific regions. The table shared below lists the keywords that are supported for a valid formula:

Keyword	Description
#Headers	This keyword refers to the column header rows area.
#Data	This keyword refers to the spreadsheet rows area.

#Totals	This keyword refers to the column footer rows area.
#RowHeaders	This keyword refers to the row header rows area.

The **Formula ('Formula Property' in the on-line documentation)** property can be used to add formulas for header and footer. When values of referenced cells in source worksheets are modified (using the copy, move, insert or delete operations), the formulas are automatically updated along with the calculated values in the worksheet.

After applying formulas, you can save the workbook to an excel file along with the flags.

Refer to the following table for the detailed list and description of flags that can be applied while saving to an excel file.

SaveCustomRowHeaders flag	Row header is changed to a new range, and the row header formula is saved.
SaveCustomColumnHeaders flag	Column header is changed to a new range, and the column header formula is saved.
Other flags	Only the data area is saved.

Users can also make use of structured reference syntax in order to refer to the header or footer area. For more information, refer to the topic **Using Structured References** in the documentation.

Using Code

You can add formula by specifying the **Formula ('Formula Property' in the on-line documentation)** property for header or footer.

Example

This example shows how to set a formula in different scenarios.

C#

```
// Sheets refer to headers or footers
fpSpread1.Sheets[1].Cells[0, 0].Formula = "SUM(Sheet1[[#Headers], $A$1:$B$2])";
fpSpread1.Sheets[1].Cells[0, 2].Formula = "Sheet2[[#Totals], $A$1]";

// Headers or footers refer to itself
fpSpread1.Sheets[2].ColumnHeader.Cells[0, 3].Formula = "Sheet1[[#Headers], [$B$2]]";
fpSpread1.Sheets[2].ColumnFooter.Cells[0, 5].Formula = "Sheet1[[#Totals], [A3]]";

// ColumnFooter refers to sheets
fpSpread1.ActiveSheet.ColumnFooter.Cells[0, 3].Formula = "$A$1";
```

VB

```
' Sheets refer to headers or footers
fpSpread1.Sheets(1).Cells(0, 0).Formula = "SUM(Sheet1[[#Headers], $A$1:$B$2])"
fpSpread1.Sheets(1).Cells(0, 2).Formula = "Sheet2[[#Totals], $A$1]"

' Headers or footers refer to itself
fpSpread1.Sheets(2).ColumnHeader.Cells(0, 3).Formula = "Sheet1[[#Headers], [$B$2]]"
fpSpread1.Sheets(2).ColumnFooter.Cells(0, 5).Formula = "Sheet1[[#Totals], [A3]]"

' ColumnFooter refers to sheets
fpSpread1.ActiveSheet.ColumnFooter.Cells(0, 3).Formula = "$A$1"
```

Example

This example shows how to set a formula in RowHeader in order to refer to the data available in a table.

C#

```
// RowHeader refers to table
fpSpread1.ActiveSheet.RowHeader.Cells[9, 0].Formula = "SUM(Table1[ [#Totals] ] )";
```

VB

```
' RowHeader refers to table
fpSpread1.ActiveSheet.RowHeader.Cells(9, 0).Formula = "SUM(Table1[ [#Totals] ] )"
```

Example

This example shows how to use a custom name for the formula.

C#

```
fpSpread1.AsWorkbook().Names.Add("name_1", "Sheet2[ [#Headers] , $A$1:$B$5 ] ");
fpSpread1.Sheets[1].ColumnHeader.Cells[0, 3].Formula = "name_1 + 2";
```

VB

```
fpSpread1.AsWorkbook().Names.Add("name_1", "Sheet2[ [#Headers] , $A$1:$B$5 ] ")
fpSpread1.Sheets(1).ColumnHeader.Cells(0, 3).Formula = "name_1 + 2"
```

 **Note:** The worksheet header or footer formulas doesn't support external reference i.e. reference to a cell or a cell range of another workbook is not supported. Also, the references to column footer cannot be exported to XLSX format.

Auto Format Formulas

Spread for Winforms provides support for automatic formatting of formula values based on the appropriate data type returned by the function in the same way as in Excel 2019 and Excel for Office 365. This feature can be used when users want the Spread control to automatically format the cells based on the formula return value.

The following image depicts how the formulas are automatically formatted when the Auto Formatting feature is enabled in the spreadsheet.

 AutoFormatting

	A	B	C
1	AutoFormatting is set to False		
2	\$0.20	AVERAGE(A2:A6)	0.21
3	\$0.10	MEDIAN(A2:A6)	0.2
4	\$0.25	Today()	43763
5	\$0.20	RATE(60,-5,150)	0.0263203630115447
6	\$0.30		
7			

	A	B	C
1	AutoFormatting is set to True		
2	\$0.20	AVERAGE(A2:A6)	\$0.21
3	\$0.10	MEDIAN(A2:A6)	\$0.20
4	\$0.25	Today()	10/25/2019
5	\$0.20	RATE(60,-5,150)	3%
6	\$0.30		
7			

 **Note:** The auto formatting feature works only when the user inputs the formula directly into the cell in flat style mode (LegacyBehaviors.Style is not used). If users edit a cell in flat style mode, the detected cell format is always applied regardless of the **AutoFormatting** ('AutoFormatting Property' in the on-line **documentation**) property.

Date Format - Users can use the formula functions that return date values (e.g. TODAY, DATE, etc.) and date

formatting will be automatically applied in the cells of the spreadsheet just like in Excel if the **AutoFormatting ('AutoFormatting Property' in the on-line documentation)** property is set to "True".

Number Format - Users can use the formula functions that return number values (e.g. SUM, SUBTOTAL, RATE, IRR, DB, FV etc.) and number formatting will be automatically applied to the cells of the spreadsheet just like in Excel if the **AutoFormatting ('AutoFormatting Property' in the on-line documentation)** property is set to "True".

Cell Reference Format - Users can use a cell reference in the formula and cell reference format will be automatically applied to the cells of the spreadsheet in the same way as in Excel if the **AutoFormatting ('AutoFormatting Property' in the on-line documentation)** property is set to "True".

Example - For example, in the above image, when the **AutoFormatting** property is set to False, cell C4 shows the date as an integer value (incorrect format). But when the **AutoFormatting** property is enabled i.e. the value of this property is set to true, the date is displayed in the correct format (MM/DD/YY).

Similarly, the cell C5 containing RATE formula doesn't display the data in percentage format but once the Auto formatting feature is enabled, cell C5 displays the correct output of RATE formula in the percentage format.

Also, the cells C2 and C3 containing the AVERAGE formula and the MEDIAN formula of the cell range (A2:A6) respectively take up the format of the cells on which they are dependent.

Enable Auto Formatting

Users can enable the Auto Formatting feature by setting the value of the **AutoFormatting ('AutoFormatting Property' in the on-line documentation)** property to "true", as shown in the following code snippet.

C#

```
// Enable AutoFormatting
fpSpread1.AsWorkbook().Features.AutoFormatting = true;
```

Using Code

This example code shows how formulas can be automatically formatted in the worksheet.

C#

```
// Enable AutoFormatting
fpSpread1.AsWorkbook().Features.AutoFormatting = true;

fpSpread1.AsWorkbook().ActiveSheet.Cells[0, 0].Text = "AutoFormatting is set to True";
fpSpread1.AsWorkbook().ActiveSheet.Cells[0, 0].BackColor = Color.Thistle;
fpSpread1.AsWorkbook().ActiveSheet.AddSpanCell(0, 0, 1, 3);
fpSpread1.AsWorkbook().ActiveSheet.Cells[1, 0].Text = "$0.20";
fpSpread1.AsWorkbook().ActiveSheet.Cells[2, 0].Text = "$0.10";
fpSpread1.AsWorkbook().ActiveSheet.Cells[3, 0].Text = "$0.25";
fpSpread1.AsWorkbook().ActiveSheet.Cells[4, 0].Text = "$0.20";
fpSpread1.AsWorkbook().ActiveSheet.Cells[5, 0].Text = "$0.30";
fpSpread1.AsWorkbook().ActiveSheet.Cells[1, 1].Text = "AVERAGE (A2:A6) ";
fpSpread1.AsWorkbook().ActiveSheet.Cells[2, 1].Text = "MEDIAN (A2:A6) ";
fpSpread1.AsWorkbook().ActiveSheet.Cells[3, 1].Text = "Today() ";
fpSpread1.AsWorkbook().ActiveSheet.Columns[1, 3].Width = 130;
// Formula cells gets formatted dependent on cells they are referring to
// 1. Example of Cell Reference Format
fpSpread1.AsWorkbook().ActiveSheet.Cells["C2"].Formula = "AVERAGE (A2:A6) ";
fpSpread1.AsWorkbook().ActiveSheet.Cells["C3"].Formula = "MEDIAN (A2:A6) ";
// 2. Example of Number Format
fpSpread1.AsWorkbook().ActiveSheet.Cells["B5"].Text = "RATE (60, -5, 150) ";
```

```
fpSpread1.AsWorkbook().ActiveSheet.Cells["C5"].Formula = "RATE(60,-5,150)";  
// 3. Example of Date Format  
fpSpread1.AsWorkbook().ActiveSheet.Cells["C4"].Formula = "Today()";
```

Managing External Reference

Spread for WinForms supports interaction with other Spread instances in order to facilitate data interchange with the help of external references. An external reference in a spreadsheet refers to the contents of a cell or a range of cells lying in another workbook.

While using external references, if the referred workbook is available in the memory, then the data is fetched directly from the external workbook; else the required information is fetched from the cached data. Usually, an external workbook caches only the cell references and not the complete data in the spreadsheet. In such a scenario, all the remaining cells in the external workbook will remain empty.

Apart from another workbook, you can also create external reference to an XLSX file. While referring to an XLSX file, if the workbook isn't available, data is imported from XLSX to cached data storage of external book.

Refer to the following scenarios for more information regarding external reference:

- **Creating a new workbook in existing workbook set**
- **Adding existing workbook to the workbook set**
- **Opening external file in the workbook and displaying it in Spread control**
- **Creating external references between workbooks of same workbook set**
- **Updating values in source workbook**
- **Setting break links**

Spread for WinForms also supports ExternalReference flag for saving an Excel file. If ExternalReference flag is turned on, behavior of the workbook will be same as Excel, else saving the workbook will not change anything. To save external link values with the workbook when exporting to Excel, user needs to set the SaveLinkValues property to true.

 External reference feature is available only for files saved in OpenXML format. XML and binary formatted files do not support this feature.

Creating a new workbook in existing workbook set

Refer to the following code snippet to create a new workbook in existing workbook set.

C#

```
// Create a workbookSet  
IWorkbookSet workbookSet;  
workbookSet = GrapeCity.Spreadsheet.Win.Factory.CreateWorkbookSet();  
  
// Create new workbook in same workbookSet  
IWorkbook workbook1;  
workbook1 = workbookSet.Workbooks.Add();  
  
// Attaching workbook1 to Spread control fpSpread1  
fpSpread1.Attach(workbook1);
```

VB

```
' Create a workbookSet  
Dim workbookSet As IWorkbookSet
```

```
workbookSet = GrapeCity.Spreadsheet.Win.Factory.CreateWorkbookSet()  
  
' Create new workbook in same workbookSet  
Dim workbook1 As IWorkbook  
workbook1 = workbookSet.Workbooks.Add()  
  
' Attaching workbook1 to Spread control fpSpread1  
fpSpread1.Attach(workbook1)
```

Adding existing workbook to the workbook set

Before adding an existing workbook to workbook set, you must make sure that the name of the workbook is unique and that the new workbook contains no data. There can be one worksheet in the new workbook, but that worksheet should be empty too. Refer to the following code snippet for the implementation.

C#

```
// Add an existing workbook to workbookSet  
IWorkbook workbook2;  
workbook2 = fpSpread2.AsWorkbook();  
workbook2.Name = "Book2"; // Default name is "fpSpread2"  
workbookSet.Workbooks.Add(workbook2);
```

VB

```
' Add an existing workbook to workbookSet  
Dim workbook2 As IWorkbook  
workbook2 = fpSpread2.AsWorkbook()  
workbook2.Name = "Book2" ' Default name is "fpSpread2"  
workbookSet.Workbooks.Add(workbook2)
```

Opening external file in the workbook and displaying it in Spread control

Refer to the following code snippet to open and display external file in the Spread control.

C#

```
// Open new workbook from a file  
IWorkbook workbook3;  
workbook3 = workbookSet.Workbooks.Open(@"Test.xlsx");  
workbook3.Name = "Book3"; // Default name is "TEST.xlsx"  
  
// Use an opened workbook to display in spread control  
fpSpread3.Attach(workbook3);
```

VB

```
' Open new workbook from a file  
Dim workbook3 As IWorkbook  
workbook3 = workbookSet.Workbooks.Open("Test.xlsx")  
workbook3.Name = "Book3" ' Default name is "TEST.xlsx"  
  
' Use an opened workbook to display in spread control  
fpSpread3.Attach(workbook3)
```

Creating external references between workbooks of same workbook set

Refer to the following code snippet to create external references between workbooks of same workbook set.

C#

```
// Assigning external cell reference formula in workbook2 referring to cell in
workbook1
fpSpread2.Sheets[0].Cells[1, 1].Formula = "[Book1]Sheet1!$B$2";

//Assigning cell formula in workbook1 referring to range in workbook2 & cell formula in
workbook2 referring to range in Workbook1
fpSpread1.Sheets[0].SetFormula(1, 3, "SUM([Book2]Sheet1!C3:C4)");
fpSpread2.Sheets[0].SetFormula(1, 3, "SUM([Book1]Sheet1!C3:C4)");

//Create custom name in workbook1 referring to range in workbook2
fpSpread1.ActiveSheet.AddCustomName("Alpha", "Sum([Book2]Sheet1!C3:C4)", 2, 2);
fpSpread1.ActiveSheet.SetFormula(2, 2, "Alpha");
fpSpread2.ActiveSheet.SetValue(2, 2, 10);
fpSpread2.ActiveSheet.SetValue(3, 2, 20);

//Add cell formula in workbook1 referring to custom name of workbook2
fpSpread2.ActiveSheet.AddCustomName("Beta", "$C$3:$C$4", 1, 1);
fpSpread2.ActiveSheet.SetValue(2, 2, 10);
fpSpread2.ActiveSheet.SetValue(3, 2, 20);
fpSpread1.ActiveSheet.SetFormula(1, 4, "SUM(Book2!Beta)");
```

VB

```
' Assigning external cell reference formula in workbook2 referring to cell in workbook1
fpSpread2.Sheets(0).Cells(1, 1).Formula = "[Book1]Sheet1!$B$2"
'Assigning cell formula in workbook1 referring to range in workbook2 & cell formula in
workbook2 referring to range in Workbook1
fpSpread1.Sheets(0).SetFormula(1, 3, "SUM([Book2]Sheet1!C3:C4)")
fpSpread2.Sheets(0).SetFormula(1, 3, "SUM([Book1]Sheet1!C3:C4)")

'Create custom name in workbook1 referring to range in workbook2
fpSpread1.ActiveSheet.AddCustomName("Alpha", "Sum([Book2]Sheet1!C3:C4)", 2, 2)
fpSpread1.ActiveSheet.SetFormula(2, 2, "Alpha")
fpSpread2.ActiveSheet.SetValue(2, 2, 10)
fpSpread2.ActiveSheet.SetValue(3, 2, 20)

'Add cell formula in workbook1 referring to custom name of workbook2
fpSpread2.ActiveSheet.AddCustomName("Beta", "$C$3:$C$4", 1, 1)
fpSpread2.ActiveSheet.SetValue(2, 2, 10)
fpSpread2.ActiveSheet.SetValue(3, 2, 20)
fpSpread1.ActiveSheet.SetFormula(1, 4, "SUM(Book2!Beta)")
```

Updating values in source workbook

Modifying cell values in source workbook will automatically update formulas in the referred workbook. Before resetting the source workbook, you must make sure that the workbook is closed. Refer to the following code snippet for the implementation.

C#

```
//Change reference cell in source workbook, then formula is automatically updated in
fpSpread2
fpSpread1.Sheets[0].Cells[2, 1].Value = 1000;
```

```
// Reset Source workbook
fpSpread1.AsWorkbook().Close();
fpSpread1.Reset(); // We need to close workbook before reset
```

VB

```
'Change reference cell in source workbook, then formula is updated in fpSpread2
fpSpread1.Sheets(0).Cells(2, 1).Value = 1000

' Reset Source workbook
fpSpread1.AsWorkbook().Close()
Dim ' We need to close workbook before reset As fpSpread1.Reset()
```

Setting break links

Setting break links in the source workbook replaces external references with values. Refer to the following code snippet for the implementation.

C#

```
// Support "break links"
var value1 = workbook1.ActiveSheet.Cells[2, 2].Value;
fpSpread2.ActiveSheet.SetValue(2, 2, 20);
var value2 = workbook1.ActiveSheet.Cells[2, 2].Value;
MessageBox.Show(string.Format("Before break link, Value is changing : value1={0}
value2={1} ", value1, value2));

workbook1.BreakLink(workbook2.Name);
fpSpread2.ActiveSheet.SetValue(2, 2, 30);
var value3 = workbook1.ActiveSheet.Cells[2, 2].Value;
MessageBox.Show(string.Format("After Break link, Value isn't changing : value1={0}
value2={1} ", value2, value3));
```

VB

```
' Support "break links"
Dim value1 As var = workbook1.ActiveSheet.Cells(2,2).Value
fpSpread2.ActiveSheet.SetValue(2, 2, 20)
Dim value2 As var = workbook1.ActiveSheet.Cells(2,2).Value
MessageBox.Show(String.Format("Before break link, Value is changing : value1={0}
value2={1} ", value1, value2))

workbook1.BreakLink(workbook2.Name)
fpSpread2.ActiveSheet.SetValue(2, 2, 30)
Dim value3 As var = workbook1.ActiveSheet.Cells(2,2).Value
MessageBox.Show(String.Format("After Break link, Value isn't changing : value1={0}
value2={1} ", value2, value3))
```

Precedents and Dependents

Spread Winforms allows you to trace precedents and dependents cells in spreadsheets.

- **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

Tracing precedents and dependents cells help you to check calculation issues, debug formulas, and validate the accuracy of results. You can also display cells related to the selected formula cell and observe which cells are impacted if a cell value is modified.

Trace Precedents

You can trace the direct precedents of a cell by using **ShowPrecedents** method of the **IRange** interface. It displays the relational arrows towards the precedents as shown in the below gif:

	A	B	C	D	E
1	1	3	3		
2	2		3		10
3					
4					
5			4		

You can also use the **DirectPrecedents** property of **IRange** interface to fetch a range object that represents the range containing all the direct precedents of a cell.

The following example shows the implementation of **ShowPrecedents** method and **DirectPrecedents** property.

C#

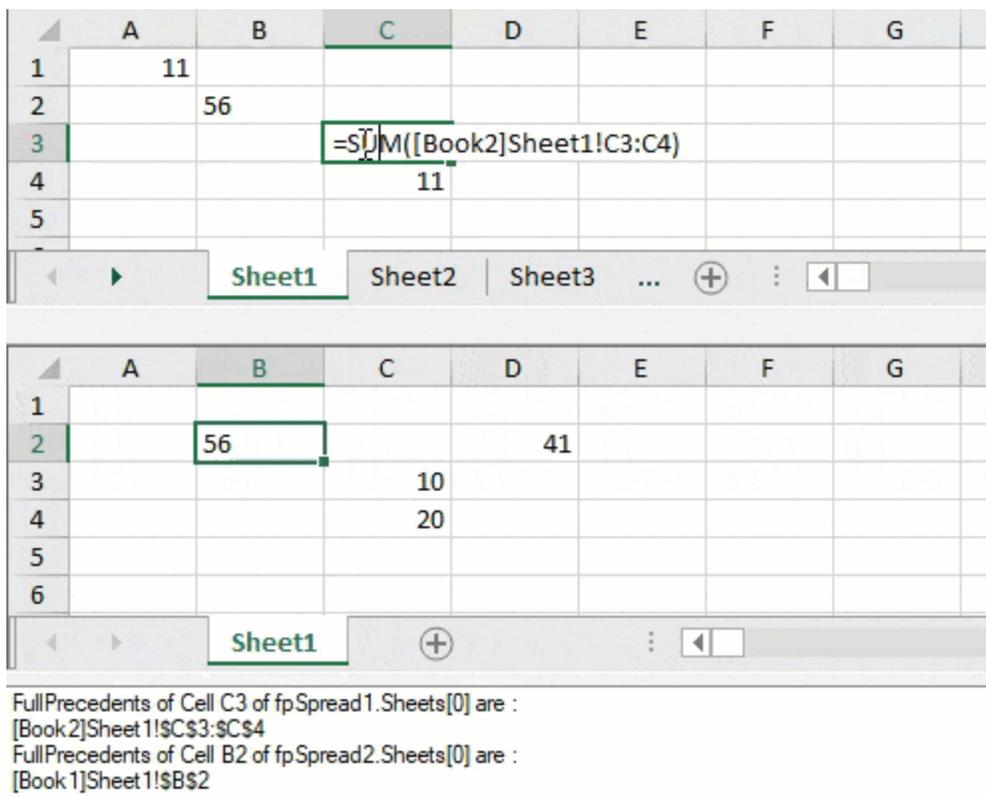
```
// Set Formula in Cell E2
fpSpread1.Sheets[0].Cells["E2"].Formula = "Sum(C1:C2, C5)";
// Set Formula in Cell C1
fpSpread1.Sheets[0].Cells["C1"].Formula = "B1";
// Set Formula in Cell B1
fpSpread1.Sheets[0].Cells["B1"].Formula = "Sum(A1:A2)";
// Set Value of Cells
fpSpread1.Sheets[0].Cells["A1"].Value = 1;
fpSpread1.Sheets[0].Cells["A2"].Value = 2;
fpSpread1.Sheets[0].Cells["C2"].Value = 3;
fpSpread1.Sheets[0].Cells["C5"].Value = 4;
// ShowPrecedents of Cell E2
fpSpread1.ShowPrecedents(0, 1, 4);
// DirectPrecedents of Cell E2
IRange DirectPrecedents =
fpSpread1.AsWorkbook().Worksheets[0].Cells["E2"].DirectPrecedents;
listBox1.Items.Add("DirectPrecedents of Cell E2 are :");
int areas1 = DirectPrecedents.Areas.Count;
for (int i = 0; i < areas1; i++)
{
    listBox1.Items.Add(DirectPrecedents.Areas[i].ToString());
}
// Output
// DirectPrecedents of Cell E2 :
// [fpSpread1]Sheet1!$C$1:$C$2
// [fpSpread1]Sheet1!$C$5
```

VB

```
'Set Formula in Cell E2
FpSpread1.Sheets(0).Cells("E2").Formula = "Sum(C1:C2, C5)"
'Set Formula in Cell C1
FpSpread1.Sheets(0).Cells("C1").Formula = "B1"
```

```
'Set Formula in Cell B1
FpSpread1.Sheets(0).Cells("B1").Formula = "Sum(A1:A2) "
'Set Value of Cells
FpSpread1.Sheets(0).Cells("A1").Value = 1
FpSpread1.Sheets(0).Cells("A2").Value = 2
FpSpread1.Sheets(0).Cells("C2").Value = 3
FpSpread1.Sheets(0).Cells("C5").Value = 4
'ShowPrecedents of Cell E2
FpSpread1.ShowPrecedents(0, 1, 4)
'DirectPrecedents of Cell E2
Dim DirectPrecedents As IRange =
FpSpread1.AsWorkbook().Worksheets(0).Cells("E2").DirectPrecedents
listBox1.Items.Add("DirectPrecedents of Cell E2 are :")
Dim areas1 As Integer = DirectPrecedents.Areas.Count
For i As Integer = 0 To areas1 - 1
    listBox1.Items.Add(DirectPrecedents.Areas(i).ToString())
Next
'Output
'DirectPrecedents of Cell E2 :
'[fpSpread1]Sheet1!$C$1$C$2
'[fpSpread1]Sheet1!$C$5
```

The **GetFullPrecedents** method of **IRange** interface can be used to display all the precedents including external workbook references for the selected cell. The below gif demonstrates the tracing of precedents including external workbook references.



Alternatively, the **Precedents** property of **IRange** interface can be used to return a range object that represents all the precedents of a cell. However, this property does not trace external references.

The following example code shows the implementation of **GetFullPrecedents** method and **Precedents** property.

C#

```
// set value
fpSpread1.Sheets[0].Cells[1, 1].Value = "56";
fpSpread1.Sheets[0].SetValue(3, 2, 11);
fpSpread2.Sheets[0].SetValue(2, 2, 10);
fpSpread2.Sheets[0].SetValue(3, 2, 20);
fpSpread1.Sheets[0].Cells[0, 0].Formula = "C4";
// Assigning external cell reference formula in workbook2 referring to cell in
workbook1
fpSpread2.Sheets[0].Cells[1, 1].Formula = "[Book1]Sheet1!B2";
fpSpread2.Sheets[0].SetFormula(1, 3, "SUM([Book1]Sheet1!C3:C4)");
fpSpread1.Sheets[0].SetFormula(2, 2, "Sum([Book2]Sheet1!C3:C4)");
// cross worksheet formula
fpSpread1.Sheets[1].Cells[0, 0].Formula = "Sheet1!B2";

// cross workbook referencing
// GetFullPrecedents of cell C3 of fpSpread1.Sheets[0]
IRange[] FullPrecedents =
fpSpread1.AsWorkbook().Worksheets[0].Cells["C3"].GetFullPrecedents();
//MessageBox.Show(FullPrecedents.ToString());
listBox1.Items.Add("FullPrecedents of Cell C3 of fpSpread1.Sheets[0] are : ");
listBox1.Items.Add(FullPrecedents.GetValue(0).ToString());

// cross worksheet referencing in same workbook
// GetFullPrecedents of cell B2 of fpSpread2.Sheets[0]
listBox1.Items.Add("FullPrecedents of Cell B2 of fpSpread2.Sheets[0] are : ");
IRange[] FullPrecedents1 =
fpSpread2.AsWorkbook().Worksheets[0].Cells["B2"].GetFullPrecedents();
listBox1.Items.Add(FullPrecedents1.GetValue(0).ToString());
// Precedents of Cell A1
listBox1.Items.Add("Precedents of Cell A1 : " +
fpSpread1.AsWorkbook().Worksheets[0].Cells["A1"].Precedents[0]);
// Output
// Precedents of Cell A1 : [Book1]Sheet1!$C$4
```

VB

```
'set value
FpSpread1.Sheets(0).Cells(1, 1).Value = "56"
FpSpread1.Sheets(0).SetValue(3, 2, 11)
FpSpread2.Sheets(0).SetValue(2, 2, 10)
FpSpread2.Sheets(0).SetValue(3, 2, 20)
FpSpread1.Sheets(0).Cells(0, 0).Formula = "C4"
'Assigning external cell reference formula in workbook2 referring to cell in workbook1
FpSpread2.Sheets(0).Cells(1, 1).Formula = "[Book1]Sheet1!B2"
FpSpread2.Sheets(0).SetFormula(1, 3, "SUM([Book1]Sheet1!C3:C4)")
FpSpread1.Sheets(0).SetFormula(2, 2, "Sum([Book2]Sheet1!C3:C4)")
'cross worksheet formula
FpSpread1.Sheets(1).Cells(0, 0).Formula = "Sheet1!B2"
'cross workbook referencing
'GetFullPrecedents of cell C3 of fpSpread1.Sheets(0)
Dim FullPrecedents As IRange() =
FpSpread1.AsWorkbook().Worksheets(0).Cells("C3").GetFullPrecedents(False)
'MessageBox.Show(FullPrecedents.ToString());
ListBox1.Items.Add("FullPrecedents of Cell C3 of fpSpread1.Sheets(0) are : ")
```

```

ListBox1.Items.Add(FullPrecedents.GetValue(0).ToString())
'cross worksheet referencing in same workbook
'GetFullPrecedents of cell B2 of fpSpread2.Sheets(0)
ListBox1.Items.Add("FullPrecedents of Cell B2 of fpSpread2.Sheets(0) are : ")
Dim FullPrecedents1 As IRange() =
FpSpread2.AsWorkbook().Worksheets(0).Cells("B2").GetFullPrecedents()
ListBox1.Items.Add(FullPrecedents1.GetValue(0).ToString())
'Precedents of Cell A1
ListBox1.Items.Add("Precedents of Cell A1 : ")
ListBox1.Items.Add(FpSpread1.AsWorkbook().Worksheets(0).Cells("A1").Precedents(0))
'Output
'Precedents of Cell A1 :
'[Book2]Sheet1!$C$4

```

The **GetFullPrecedents** method also provides **excludeUnlinkedWorksheet** parameter which excludes all precedents referring to the workbook that are not loaded in the WorkbookSet. Its default value is true. However, it includes precedents from external worksheets in other workbooks that are not currently loaded in the WorkbookSet, when set to false.

C#

```

// Set external Formula in Cell E2 of fpSpread1 i.e. the formula include reference to
fpSpread2
fpSpread1.Sheets[0].Cells["A1"].Value = 2000;
fpSpread1.Sheets[0].Cells["E2"].Formula = "A1 + [fpSpread2]Sheet1!A1";
// set value in fpSpread2 Cell A1
fpSpread2.Sheets[0].Cells["A1"].Value = 1000;
// Add two spread controls in a workbook set
fpSpread1.AsWorkbook().WorkbookSet.Workbooks.Add(fpSpread2.AsWorkbook());
// Unlink fpSpread2 from workbook set
fpSpread2.AsWorkbook().Close();
// Getting precedents by excluding the unlinked worksheet
listBox1.Items.Add("GetFullPrecedents of Cell E2 WITHOUT broken link :");
IRange[] precedents =
fpSpread1.AsWorkbook().Worksheets[0].Cells["E2"].GetFullPrecedents(true);
// Output
// GetFullPrecedents of Cell E2 WITHOUT broken link :
// [fpSpread1]Sheet1!$A$1
// Getting precedents by including the unlinked worksheet
listBox1.Items.Add("GetFullPrecedents of Cell E2 WITH broken link - ExternalReference
item(s) may be listed :");
IRange[] precedents2 =
fpSpread1.AsWorkbook().Worksheets[0].Cells["E2"].GetFullPrecedents(false);
// Output
// GetFullPrecedents of Cell E2 WITH broken link - ExternalReference item(s) may be
listed :
// GrapeCity.Spreadsheet.API.ExternalRange
// [fpSpread1]Sheet1!$A$1

```

VB

```

'Set external Formula in Cell E2 of fpSpread1 i.e. the formula include reference to
fpSpread2
FpSpread1.Sheets(0).Cells("A1").Value = 2000
FpSpread1.Sheets(0).Cells("E2").Formula = "A1 + [fpSpread2]Sheet1!A1"
'set value in fpSpread2 Cell A1
FpSpread2.Sheets(0).Cells("A1").Value = 1000

```

```

'Set two spread controls in a workbook set
FpSpread1.AsWorkbook().WorkbookSet.Workbooks.Add(FpSpread2.AsWorkbook())
'Unlink fpSpread2 from workbook set
FpSpread2.AsWorkbook().Close()
'Getting precedents by excluding the unlinked worksheet
ListBox1.Items.Add("GetFullPrecedents of Cell E2 WITHOUT broken link :")
Dim precedents As IRange() =
FpSpread1.AsWorkbook().Worksheets(0).Cells("E2").GetFullPrecedents(True)
'Output
'GetFullPrecedents of Cell E2 WITHOUT broken link :
'[fpSpread1]Sheet1!$A$1
'Getting precedents by including the unlinked worksheet
ListBox1.Items.Add("GetFullPrecedents of Cell E2 WITH broken link - ExternalReference
item(s) may be listed :")
Dim precedents2 As IRange() =
FpSpread1.AsWorkbook().Worksheets(0).Cells("E2").GetFullPrecedents(False)
'Output
'GetFullPrecedents of Cell E2 WITH broken link - ExternalReference item(s) may be
listed :
'GrapeCity.Spreadsheet.API.ExternalRange
'[fpSpread1]Sheet1!$A$1

```

Trace Dependents

You can trace the direct dependents of a cell by using **ShowDependents** method of the **IRange** interface. It displays the relational arrows towards the dependents as shown in the below gif:

	A	B	C	D	E	F	G
1	3		3	3	3	3	3
2		1	3				
3		2					

You can also use the **DirectDependents** property of **IRange** interface to fetch a range object that represents the range containing all the direct dependents of a cell.

The following example shows the implementation of **ShowDependents** method and **DirectDependents** property.

C#

```

// Set formulas
fpSpread1.Sheets[1].Cells[0, 0].Formula = "B2+B3";
fpSpread1.Sheets[1].Cells["C1"].Formula = "A1";
fpSpread1.Sheets[1].Cells["C2"].Formula = "A1";
fpSpread1.Sheets[1].Cells["D1"].Formula = "A1";
fpSpread1.Sheets[1].Cells["F1"].Formula = "A1";
fpSpread1.Sheets[1].Cells["G1"].Formula = "A1";
fpSpread1.Sheets[1].Cells["E1"].Formula = "D1";
// Set Value of Cells
fpSpread1.Sheets[1].Cells["B2"].Value = 1;
fpSpread1.Sheets[1].Cells["B3"].Value = 2;
// ShowDependents of Cell A1
fpSpread1.ShowDependents(1, 0, 0);
// DirectDependents of Cell A1
IRange DirectDependents =
fpSpread1.AsWorkbook().Worksheets[1].Cells["A1"].DirectDependents;
listBox1.Items.Add("DirectDependents of Cell A1 are :");
int areas1 = DirectDependents.Areas.Count;

```

```

for (int i = 0; i < areas1; i++)
{
    listBox1.Items.Add(DirectDependents.Areas[i].ToString());
}
// Output
// DirectDependents of Cell A1 are :
// [fpSpread1]Sheet2!$C$1:$C$2
// [fpSpread1]Sheet2!$D$1
// [fpSpread1]Sheet2!$F$1:$G$1

```

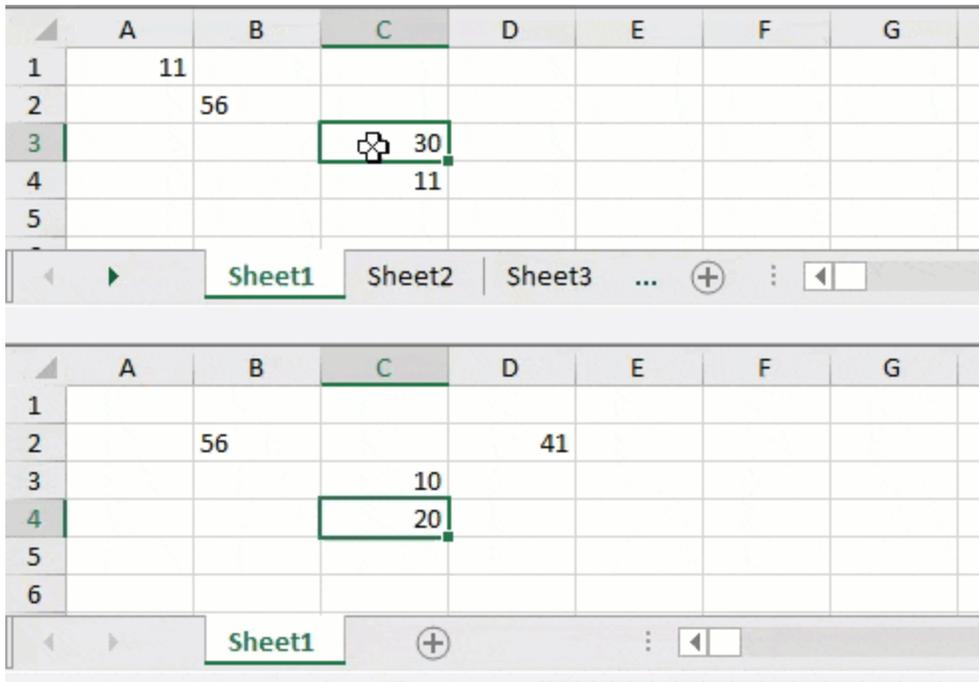
VB

```

'Set Formula in Cells
FpSpread1.Sheets(1).Cells(0, 0).Formula = "B2+B3"
FpSpread1.Sheets(1).Cells("C1").Formula = "A1"
FpSpread1.Sheets(1).Cells("C2").Formula = "A1"
FpSpread1.Sheets(1).Cells("D1").Formula = "A1"
FpSpread1.Sheets(1).Cells("F1").Formula = "A1"
FpSpread1.Sheets(1).Cells("G1").Formula = "A1"
FpSpread1.Sheets(1).Cells("E1").Formula = "D1"
'Set Value of Cells
FpSpread1.Sheets(1).Cells("B2").Value = 1
FpSpread1.Sheets(1).Cells("B3").Value = 2
'ShowDependents of Cell A1
FpSpread1.ShowDependents(1, 0, 0)
'DirectDependents of Cell A1
Dim DirectDependents As IRange =
FpSpread1.AsWorkbook().Worksheets(1).Cells("A1").DirectDependents
ListBox1.Items.Add("DirectDependents of Cell A1 are :")
Dim areas1 As Integer = DirectDependents.Areas.Count
For i As Integer = 0 To areas1 - 1
    ListBox1.Items.Add(DirectDependents.Areas(i).ToString())
Next
'Output
'DirectDependents of Cell A1 are :
'[fpSpread1]Sheet2!$C$1:$C$2
'[fpSpread1]Sheet2!$D$1
'[fpSpread1]Sheet2!$F$1:$G$1

```

The **GetFullDependents** method of IRange interface can be used to display all the dependents including external workbook references for the selected cell. The below gif demonstrates the tracing of dependents including external workbook references.



FullDependents of Cell C4 of fpSpread2.Sheets[0] are :
 [Book1]Sheet1!\$C\$3
 [Book2]Sheet1!\$D\$2

Alternatively, the **Dependents** property of **IRange** interface can be used to return a range object that represents all the dependents of a cell. However, this property does not trace external references.

The following example shows the implementation **GetFullDependents** method and **Dependents** property.

C#

```
// set value
fpSpread1.Sheets[0].Cells[1, 1].Value = "56";
fpSpread1.Sheets[0].SetValue(3, 2, 11);
fpSpread2.Sheets[0].SetValue(2, 2, 10);
fpSpread2.Sheets[0].SetValue(3, 2, 20);
fpSpread1.Sheets[0].Cells[0, 0].Formula = "C4";
// Assigning external cell reference formula in workbook2 referring to cell in
workbook1
fpSpread2.Sheets[0].Cells[1, 1].Formula = "[Book1]Sheet1!B2";
fpSpread2.Sheets[0].SetFormula(1, 3, "SUM([Book1]Sheet1!C3:C4)");
fpSpread1.Sheets[0].SetFormula(2, 2, "Sum([Book2]Sheet1!C3:C4)");
// cross worksheet formula
fpSpread1.Sheets[1].Cells[0, 0].Formula = "Sheet1!B2";

// GetFullDependents of cell C4 of fpSpread2.Sheets[0]
listBox1.Items.Add("FullDependents of Cell C4 of fpSpread2.Sheets[0] are : ");
IRange[] FullDependents =
fpSpread2.AsWorkbook().Worksheets[0].Cells["C4"].GetFullDependents();
listBox1.Items.Add(FullDependents.GetValue(0).ToString());
listBox1.Items.Add(FullDependents.GetValue(1).ToString());
// Dependents of Cell C4
IRange Dependents = fpSpread1.AsWorkbook().Worksheets[1].Cells["C4"].Dependents;
listBox1.Items.Add("Dependents of Cell C4 are :");
int areas = Dependents.Areas.Count;
```

```

for (int i = 0; i < areas; i++)
{
    listBox1.Items.Add(Dependents.Areas[i].ToString());
}
// Output
// Dependents of Cell C4 are :
// [Book1]Sheet1!$A$1

```

VB

```

'set value
FpSpread1.Sheets(0).Cells(1, 1).Value = "56"
FpSpread1.Sheets(0).SetValue(3, 2, 11)
FpSpread2.Sheets(0).SetValue(2, 2, 10)
FpSpread2.Sheets(0).SetValue(3, 2, 20)
FpSpread1.Sheets(0).Cells(0, 0).Formula = "C4"
'Assigning external cell reference formula in workbook2 referring to cell in workbook1
FpSpread2.Sheets(0).Cells(1, 1).Formula = "[Book1]Sheet1!B2"
FpSpread2.Sheets(0).SetFormula(1, 3, "SUM([Book1]Sheet1!C3:C4)")
FpSpread1.Sheets(0).SetFormula(2, 2, "Sum([Book2]Sheet1!C3:C4)")
'cross worksheet formula
FpSpread1.Sheets(1).Cells(0, 0).Formula = "Sheet1!B2"
'GetFullDependents of cell C4 of fpSpread2.Sheets(0)
ListBox1.Items.Add("FullDependents of Cell C4 of fpSpread2.Sheets(0) are : ")
Dim FullDependents As IRange() =
FpSpread2.AsWorkbook().Worksheets(0).Cells("C4").GetFullDependents()
ListBox1.Items.Add(FullDependents.GetValue(0).ToString())
ListBox1.Items.Add(FullDependents.GetValue(1).ToString())
'Dependents of Cell C4
Dim Dependents As IRange = FpSpread1.AsWorkbook().Worksheets(1).Cells("C4").Dependents
ListBox1.Items.Add("Dependents of Cell C4 are :")
Dim areas1 As Integer = Dependents.Areas.Count
For i As Integer = 0 To areas1 - 1
    ListBox1.Items.Add(Dependents.Areas(i).ToString())
Next
'Output
'Dependents of Cell C4 are :
'[Book1]Sheet1!$A$1

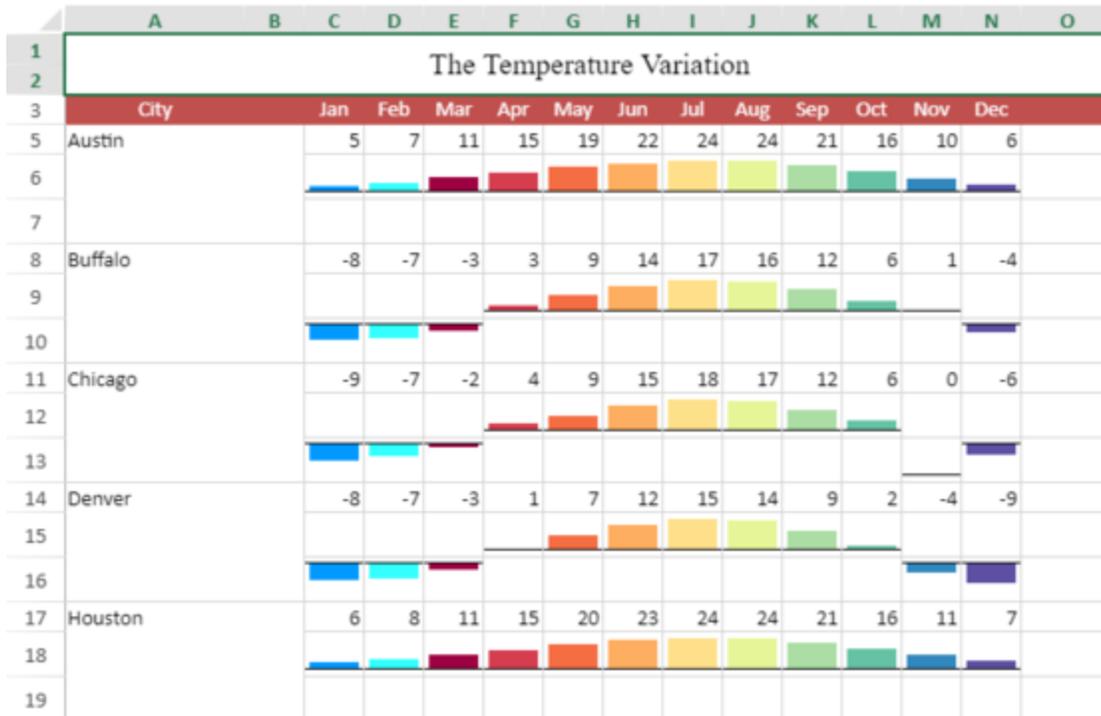
```

Sparklines

A sparkline is a small chart which can be applied to quickly visualize data and transform it in a compact form in a cell. It uses data from a range of cells to help you easily analyze data at the cell level.

You can set the sparkline type to column, line, or winloss by using the **AddSparkline ('AddSparkline Method' in the on-line documentation)** method or use type-specific formulas to create different sparklines available.

The image below shows the **Vbar sparkline** in action. It shows the trends in the temperature levels across a year in different cities and helps to quickly understand the high and low data points.



Refer to the following list of topics while working with sparkline.

Topic

Content

Add Sparklines Using Methods

Add column, line, and winloss sparklines using methods and perform following operations:

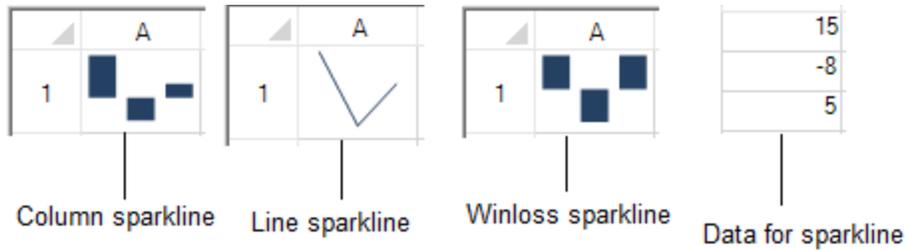
- **Specifying Horizontal and Vertical Axes**
- **Customizing Markers and Points**
- **Working with Sparklines**

Add Sparklines using Formulas

Add different types of sparklines such as pie, histogram, stacked, etc. using formulas.

Add Sparklines Using Methods

You can add a standard column, line, or winloss sparkline to a cell using the **AddSparkline ('AddSparkline Method' in the on-line documentation)** method. As shown in the following figure, these sparklines were created using a minimum axis of -8 and a maximum axis of 15.



The column sparkline draws the values as a column chart. The line sparkline draws the values as a line chart. The winloss sparkline shows the points with the same size. Negative points extend down from the axis and positive points extend up.

These sparklines can be enhanced by adding other styles and technical properties such as displaying negative points or colors for the marker points as well as colors for high, low, negative, first, and last points. You can set all these using the **ExcelSparklineSetting** ('ExcelSparklineSetting Class' in the on-line documentation) class.

Note: You can also add column, loss, or winloss sparklines using formulas. Refer to **Column, Line, and Winloss Sparkline** for more information.

Example

This example creates a column sparkline in a cell and shows negative and series colors.

C#

```
FarPoint.Win.Spread.SheetView sv = new FarPoint.Win.Spread.SheetView();
FarPoint.Win.Spread.Chart.SheetCellRange data = new
FarPoint.Win.Spread.Chart.SheetCellRange(sv, 0,0,1, 5);
FarPoint.Win.Spread.Chart.SheetCellRange data2 = new
FarPoint.Win.Spread.Chart.SheetCellRange(sv, 5,0,1,1);
FarPoint.Win.Spread.ExcelSparklineSetting ex = new
FarPoint.Win.Spread.ExcelSparklineSetting();
ex.ShowMarkers = true;
ex.ShowNegative = true;
ex.NegativeColor = Color.Red;
// Use with a Column or Winloss type
ex.SeriesColor = Color.Olive;
fpSpread1.Sheets[0] = sv;
sv.Cells[0, 0].Value = 2;
sv.Cells[0, 1].Value = 5;
sv.Cells[0, 2].Value = 4;
sv.Cells[0, 3].Value = -1;
sv.Cells[0, 4].Value = 3;
fpSpread1.Sheets[0].AddSparkline(data, data2, FarPoint.Win.Spread.SparklineType.Column,
ex);
```

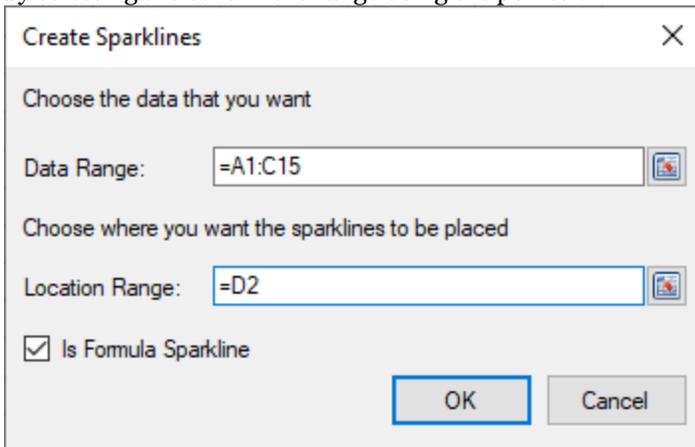
VB

```
Dim sv As New FarPoint.Win.Spread.SheetView()
Dim data As New FarPoint.Win.Spread.Chart.SheetCellRange(sv, 0, 0, 1, 5)
Dim data2 As New FarPoint.Win.Spread.Chart.SheetCellRange(sv, 5, 0, 1, 1)
Dim ex As New FarPoint.Win.Spread.ExcelSparklineSetting()
ex.ShowMarkers = True
ex.ShowNegative = True
ex.NegativeColor = Color.Red
' Use with a Column or Winloss type
```

```
ex.SeriesColor = Color.Olive
fpSpread1.Sheets(0) = sv
sv.Cells(0, 0).Value = 2
sv.Cells(0, 1).Value = 5
sv.Cells(0, 2).Value = 4
sv.Cells(0, 3).Value = -1
sv.Cells(0, 4).Value = 3
fpSpread1.Sheets(0).AddSparkline(data, data2, FarPoint.Win.Spread.SparklineType.Column,
ex)
```

Using the Spread Designer

1. Type data in a cell or a column or row of cells in the designer.
2. Select a cell for the sparkline.
3. Select the **Insert** menu.
4. Select a sparkline type.
5. Set the Data Range in the **Create Sparklines** dialog (such as =Sheet1!\$E\$1:\$E\$3). You can also set the range by selecting the cells in the range using the pointer.



6. Select **OK**.
7. Select **Apply and Exit** from the **File** menu to save your changes and close the designer.

For more information, see the following topics:

- **Specifying Horizontal and Vertical Axes**
- **Customizing Markers and Points**
- **Working with Sparklines**

Specifying Horizontal and Vertical Axes

The horizontal axis has a general and a date type. The general type specifies that all the points are painted along the axis at the same distance. The date type specifies which points are drawn and can have different distances between the points based on the day unit.

Set the **DateAxis** property to true in the **ExcelSparklineSetting** (**'ExcelSparklineSetting Class' in the on-line documentation**) class to use the date horizontal axis and the **DisplayXAxis** property if you wish to display the axis line. If a cell is blank in the date range, then that point is not drawn by default with the date axis. The following image displays all three points on the graph but leaves a larger gap between the second and third points to show the distance between January 3rd and January 5th.

	A	B	C	D
1	1/2/2011	1/3/2011	1/5/2011	
2	2	11	4	

You can specify different minimum and maximum value options for the vertical axis. The automatic option allows each sparkline to have a different minimum and maximum value. The same option uses the same minimum and maximum value for all the sparklines. The custom option allows you to specify the minimum and maximum value for all the sparklines in a group.

If the custom option is used for the vertical axis, and the minimum value is equal to or larger than all data points, points or lines are not drawn. Lines or columns are truncated if they are not completely in the drawing area. If a column sparkline has at least one point drawn completely or partially, then all columns with values less than the minimum are drawn as thin columns that extend down.

See the **ManualMax**, **ManualMin**, **MaxAxisType**, and **MinAxisType** properties in the **ExcelSparklineSetting** (**'ExcelSparklineSetting Class' in the on-line documentation**) class for vertical axis examples.

Using Code

1. Create an `ExcelSparklineSetting` object.
2. Set the **DateAxis** property to use the date for the x-axis.
3. Set the **DisplayXAxis** property to true if you wish to display the axis.
4. Add values and dates to the cells.
5. Add the sparkline to the cell.

Example

This example creates a column sparkline in a cell with a date horizontal axis.

C#

```
FarPoint.Win.Spread.ExcelSparklineSetting ex = new
FarPoint.Win.Spread.ExcelSparklineSetting();
ex.DisplayXAxis = true;
ex.DateAxis = true;
ex.Formula = "Sheet1!$A$1:$C$1";
fpSpread1.Sheets[0].Cells[0, 0].Text = "1/2/2011";
fpSpread1.Sheets[0].Cells[0, 1].Text = "1/3/2011";
fpSpread1.Sheets[0].Cells[0, 2].Text = "1/5/2011";
fpSpread1.Sheets[0].Cells[1, 0].Value = 2;
fpSpread1.Sheets[0].Cells[1, 1].Value = 11;
fpSpread1.Sheets[0].Cells[1, 2].Value = 4;
fpSpread1.Sheets[0].AddSparkline("Sheet1!$A$2:$C$2", "Sheet1!$D$2:$D$2",
FarPoint.Win.Spread.SparklineType.Column, ex);
```

VB

```
Dim ex As New FarPoint.Win.Spread.ExcelSparklineSetting()
ex.DisplayXAxis = True
ex.DateAxis = True
ex.Formula = "Sheet1!$A$1:$C$1"
fpSpread1.Sheets(0).Cells(0, 0).Text = "1/2/2011"
fpSpread1.Sheets(0).Cells(0, 1).Text = "1/3/2011"
fpSpread1.Sheets(0).Cells(0, 2).Text = "1/5/2011"
fpSpread1.Sheets(0).Cells(1, 0).Value = 2
fpSpread1.Sheets(0).Cells(1, 1).Value = 11
```

```
fpSpread1.Sheets(0).Cells(1, 2).Value = 4
fpSpread1.Sheets(0).AddSparkline("Sheet1!$A$2:$C$2", "Sheet1!$D$2:$D$2",
FarPoint.Win.Spread.SparklineType.Column, ex)
```

Using the Spread Designer

1. Type data in a cell or a column or row of cells in the designer.
2. Type dates in a cell or a column or row of cells in the designer.
3. Select a cell for the sparkline.
4. Select the **Insert** menu.
5. Select a sparkline type.
6. Select the data for the graph when the **Create Sparklines** dialog is displayed.
7. Select **OK**.
8. Select the sparkline cell, select the **Axis** icon, and then make any changes to the axis.
9. Select **Apply and Exit** from the **File** menu to save your changes and close the designer.

Customizing Markers and Points

You can show markers or points in the sparkline graphs. The following image displays the points in a line sparkline. You can specify different colors for low or negative, high, first, and last points.

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

The high point is the point for the largest value. The low point is the smallest value. The negative point represents negative values. The first point is the first point that is drawn on the graph. The last point is the last point that is drawn on the graph.

The **SeriesColor** property applies to the line for the line spark type. The **MarkersColor** property is only for the line type sparkline. See the **ExcelSparklineSetting** (**ExcelSparklineSetting Class** in the on-line documentation) class for a list of sparkline properties and additional code samples.

Using Code

1. Specify a cell to create the sparkline in.
2. Specify a range of cells for the data.
3. Set any properties for the sparkline (such as **ShowFirst** and **FirstMarkerColor**).
4. Add the sparkline to the cell.

Example

This example creates a line sparkline in a cell and shows different markers and colors.

C#

```
FarPoint.Win.Spread.SheetView sv = new FarPoint.Win.Spread.SheetView();
```

```
FarPoint.Win.Spread.Chart.SheetCellRange data = new
FarPoint.Win.Spread.Chart.SheetCellRange(sv, 0,0,1, 5);
FarPoint.Win.Spread.Chart.SheetCellRange data2 = new
FarPoint.Win.Spread.Chart.SheetCellRange(sv, 5,0,1,1);
FarPoint.Win.Spread.ExcelSparklineSetting ex = new
FarPoint.Win.Spread.ExcelSparklineSetting();
ex.AxisColor = Color.SaddleBrown;
ex.ShowFirst = true;
ex.ShowHigh = true;
ex.ShowLow = true;
ex.ShowLast = true;
ex.FirstMarkerColor = Color.Blue;
ex.HighMarkerColor = Color.DarkGreen;
ex.MarkersColor = Color.Aquamarine;
ex.LowMarkerColor = Color.Red;
ex.LastMarkerColor = Color.Orange;
ex.ShowMarkers = true;
fpSpread1.Sheets[0] = sv;
sv.Cells[0, 0].Value = 2;
sv.Cells[0, 1].Value = 5;
sv.Cells[0, 2].Value = 4;
sv.Cells[0, 3].Value = 1;
sv.Cells[0, 4].Value = 3;
fpSpread1.Sheets[0].AddSparkline(data, data2, FarPoint.Win.Spread.SparklineType.Line,
ex);
```

VB

```
Dim sv As New FarPoint.Win.Spread.SheetView()
Dim data As New FarPoint.Win.Spread.Chart.SheetCellRange(sv, 0, 0, 1, 5)
Dim data2 As New FarPoint.Win.Spread.Chart.SheetCellRange(sv, 5, 0, 1, 1)
Dim ex As New FarPoint.Win.Spread.ExcelSparklineSetting()
ex.AxisColor = Color.SaddleBrown
ex.ShowFirst = True
ex.ShowHigh = True
ex.ShowLow = True
ex.ShowLast = True
ex.FirstMarkerColor = Color.Blue
ex.HighMarkerColor = Color.DarkGreen
ex.MarkersColor = Color.Aquamarine
ex.LowMarkerColor = Color.Red
ex.LastMarkerColor = Color.Orange
ex.ShowMarkers = True
fpSpread1.Sheets(0) = sv
sv.Cells(0, 0).Value = 2
sv.Cells(0, 1).Value = 5
sv.Cells(0, 2).Value = 4
sv.Cells(0, 3).Value = 1
sv.Cells(0, 4).Value = 3
fpSpread1.Sheets(0).AddSparkline(data, data2, FarPoint.Win.Spread.SparklineType.Line,
ex)
```

Using the Spread Designer

1. Type data in a cell or a column or row of cells in the designer.
2. Select a cell for the sparkline.

3. Select the **Insert** menu.
4. Select a sparkline type.
5. Select the data for the graph when the **Create Sparklines** dialog is displayed.
6. Select **OK**.
7. Select the sparkline cell, select the **Marker Color** or **Sparkline Color** icon, and set the colors.
8. Select **Apply and Exit** from the **File** menu to save your changes and close the designer.

Working with Sparklines

Spread for Winforms provides various sparkline-related operations. You can perform group, delete, cut, copy, and switch rows and columns operations on column, line, and winloss sparklines, which are created by using methods.

Basic operations

You can use standard Windows shortcut keys to cut, copy, or paste sparklines.

When copying or deleting a group of sparklines:

- If you copy a group of sparklines, they will become a new group when they are pasted to a new location.
- Copying a single sparkline from a group, will create a new group with a single sparkline.
- If you delete a group, all sparklines in the group are also deleted.

You can clear a sparkline with the **ClearSparklines ('ClearSparklines Method' in the on-line documentation)** method in the **Sheetview** class.

The range of data used in the sparkline can be switched from row to column or column to row using the **SwitchRowColumn ('SwitchRowColumn Method' in the on-line documentation)** method. The data range should have the same number of rows and columns.

Use the **AddSquareSparkline ('AddSquareSparkline Method' in the on-line documentation)** method in the **SheetView** class to add a sparkline that can be switched.

Group Sparklines

Grouping merges sparklines into a new group and removes them from the old groups. You can use the **GroupSparkline ('GroupSparkline Method' in the on-line documentation)** method to group sparkline cells.

The data and location range of the original sparkline are used in the new groups. The settings of the original group are also used in the new groups.

If the selected sparklines belong to different groups with different types, the new group will have the type of the last selected group. The new group will also have the empty cell, style, and axis settings of the last selected group.

Ungrouping selected sparkline groups separates them into different groups that contain only one sparkline. You can use the **UnGroupSparkline ('UngroupSparkline Method' in the on-line documentation)** method to ungroup sparkline cells.

The following example shows how to use the **GroupSparkline ('GroupSparkline Method' in the on-line documentation)** method or the **UnGroupSparkline ('UngroupSparkline Method' in the on-line documentation)** method.

C#

```
FarPoint.Win.Spread.SheetView sv = new FarPoint.Win.Spread.SheetView();
private void Form1_Load(object sender, EventArgs e)
{
    FarPoint.Win.Spread.Chart.SheetCellRange test = new
    FarPoint.Win.Spread.Chart.SheetCellRange(sv, 0, 0, 2, 4);
```

```
FarPoint.Win.Spread.Model.CellRange data2 = new FarPoint.Win.Spread.Model.CellRange(0,
5, 2, 1);
FarPoint.Win.Spread.ExcelSparklineSetting ex = new
FarPoint.Win.Spread.ExcelSparklineSetting();
ex.ShowFirst = true;
ex.FirstMarkerColor = Color.Violet;
fpSpread1.Sheets[0] = sv;
sv.Cells[0, 0].Value = 2;
sv.Cells[0, 1].Value = 5;
sv.Cells[0, 2].Value = 4;
sv.Cells[0, 3].Value = -1;
sv.Cells[0, 4].Value = 3;
sv.Cells[1, 0].Value = 3;
sv.Cells[1, 1].Value = 1;
sv.Cells[1, 2].Value = 2;
sv.Cells[1, 3].Value = -1;
sv.Cells[1, 4].Value = 5;

sv.Cells[2, 0].Value = 3;
sv.Cells[2, 1].Value = 1;
sv.Cells[2, 2].Value = 2;
sv.Cells[2, 3].Value = -1;
sv.Cells[2, 4].Value = 5;
fpSpread1.Sheets[0].AddSparkline(test, data2, FarPoint.Win.Spread.SparklineType.Column,
ex);

FarPoint.Win.Spread.Chart.SheetCellRange test1 = new
FarPoint.Win.Spread.Chart.SheetCellRange(sv, 2, 0, 1, 4);
FarPoint.Win.Spread.Model.CellRange data3 = new FarPoint.Win.Spread.Model.CellRange(2,
5, 1, 1);
FarPoint.Win.Spread.ExcelSparklineSetting ex1 = new
FarPoint.Win.Spread.ExcelSparklineSetting();
ex1.FirstMarkerColor = Color.Red;
ex1.ShowFirst = true;
fpSpread1.Sheets[0].AddSparkline(test1, data3,
FarPoint.Win.Spread.SparklineType.Column, ex1);

sv.Cells[4, 0].Value = 2;
sv.Cells[4, 1].Value = 1;
sv.Cells[5, 0].Value = 5;
sv.Cells[5, 1].Value = 3;
FarPoint.Win.Spread.Chart.SheetCellRange newdata = new
FarPoint.Win.Spread.Chart.SheetCellRange(sv, 4, 0, 2, 2);
FarPoint.Win.Spread.Model.CellRange newlocation = new
FarPoint.Win.Spread.Model.CellRange(6, 0, 1, 2);
FarPoint.Win.Spread.ExcelSparklineSetting ex2 = new
FarPoint.Win.Spread.ExcelSparklineSetting();
fpSpread1.Sheets[0].AddSquareSparkline(newdata, newlocation,
FarPoint.Win.Spread.SparklineType.Column, ex2, true);
}

private void button1_Click(object sender, EventArgs e)
{

fpSpread1.Sheets[0].GroupSparkline(new FarPoint.Win.Spread.Model.CellRange[] { new
FarPoint.Win.Spread.Model.CellRange(5, 0, 3, 1) });
// fpSpread1.Sheets[0].UnGroupSparkline(new FarPoint.Win.Spread.Model.CellRange[] { new
```

```
FarPoint.Win.Spread.Model.CellRange(5, 0, 3, 1) });  
}
```

VB

```
Dim sv As New FarPoint.Win.Spread.SheetView()  
  
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)  
Handles MyBase.Load  
Dim data2 As New FarPoint.Win.Spread.Chart.SheetCellRange(sv, 0, 0, 2, 4)  
Dim test As New FarPoint.Win.Spread.Model.CellRange(0, 5, 2, 1)  
  
Dim ex As New FarPoint.Win.Spread.ExcelSparklineSetting()  
ex.ShowFirst = True  
ex.FirstMarkerColor = Color.Violet  
fpSpread1.Sheets(0) = sv  
sv.Cells(0, 0).Value = 2  
sv.Cells(0, 1).Value = 5  
sv.Cells(0, 2).Value = 4  
sv.Cells(0, 3).Value = -1  
sv.Cells(0, 4).Value = 3  
sv.Cells(1, 0).Value = 3  
sv.Cells(1, 1).Value = 1  
sv.Cells(1, 2).Value = 2  
sv.Cells(1, 3).Value = -1  
sv.Cells(1, 4).Value = 5  
  
sv.Cells(2, 0).Value = 3  
sv.Cells(2, 1).Value = 1  
sv.Cells(2, 2).Value = 2  
sv.Cells(2, 3).Value = -1  
sv.Cells(2, 4).Value = 5  
fpSpread1.Sheets(0).AddSparkline(data2, test, FarPoint.Win.Spread.SparklineType.Column,  
ex)  
  
Dim data3 = New FarPoint.Win.Spread.Chart.SheetCellRange(sv, 2, 0, 1, 4)  
Dim test1 = New FarPoint.Win.Spread.Model.CellRange(2, 5, 1, 1)  
Dim ex1 As New FarPoint.Win.Spread.ExcelSparklineSetting()  
ex1.FirstMarkerColor = Color.Red  
ex1.ShowFirst = True  
fpSpread1.Sheets(0).AddSparkline(data3, test1,  
FarPoint.Win.Spread.SparklineType.Column, ex1)  
  
sv.Cells(4, 0).Value = 2  
sv.Cells(4, 1).Value = 1  
sv.Cells(5, 0).Value = 5  
sv.Cells(5, 1).Value = 3  
Dim newdata = New FarPoint.Win.Spread.Chart.SheetCellRange(sv, 4, 0, 2, 2)  
Dim newlocation = New FarPoint.Win.Spread.Model.CellRange(6, 0, 1, 2)  
Dim ex2 As New FarPoint.Win.Spread.ExcelSparklineSetting()  
fpSpread1.Sheets(0).AddSquareSparkline(newdata, newlocation,  
FarPoint.Win.Spread.SparklineType.Column, ex2, True)  
End Sub  
  
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)  
Handles Button1.Click  
fpSpread1.Sheets(0).GroupSparkline(New FarPoint.Win.Spread.Model.CellRange() {New
```

```
FarPoint.Win.Spread.Model.CellRange(5, 0, 3, 1))
'fpSpread1.Sheets(0).UnGroupSparkline(New FarPoint.Win.Spread.Model.CellRange() {New
FarPoint.Win.Spread.Model.CellRange(5, 0, 3, 1)})
End Sub
```

Using the Spread Designer

1. Select the sparkline cells.
2. Select **Group** or **Ungroup** in the **Group** section of the toolbar to group or ungroup sparklines.
3. Select **Clear** if you wish to remove any sparklines.
4. Use the **Edit Data** option to edit the data or switch the range of data used in the sparkline.
5. Select **Apply and Exit** from the **File** menu to save your changes and close the designer.

Hide Group Indicator

When you select a sparkline cell present in a group, all the sparklines in that group are highlighted.

Spread for Winforms provides an option to hide the sparkline group indicator when you select a sparkline cell. You can set the **SparklineGroupRenderer** ('**SparklineGroupRenderer Property**' in the on-line documentation) property to null to hide the indicator.

Show Group Indicator

	A	B	C	D
1	1	2	3	
2	4	6	7	
3	10	11	12	
4	15	16	17	

Hide Group Indicator

	A	B	C	D
1	1	2	3	
2	4	6	7	
3	10	11	12	
4	15	16	17	

C#

```
// Hide sparkline group indicator
fpSpread1.DefaultSkin.SparklineGroupRenderer = null;
```

Visual Basic

```
'Hide sparkline group indicator
FpSpread1.DefaultSkin.SparklineGroupRenderer = Nothing
```

Add Sparklines using Formulas

You can add different types of sparklines using their respective formulas in Spread for Winforms. The following table showcases the different sparklines available:

Types of Sparklines			
Column Sparkline	Line Sparkline	Winloss Sparkline	Area Sparkline
Box Plot Sparkline	Bullet Sparkline	Cascade Sparkline	Gauge KPI Sparkline

Hbar Sparkline	Vbar Sparkline	Histogram Sparkline	Image Sparkline
Month Sparkline	Year Sparkline	Pareto Sparkline	Pie Sparkline
Scatter Sparkline	Spread Sparkline	Stacked Sparkline	Vari Sparkline

 **Note:** The GrapeCity.Spreadsheet.DataVisualization.dll file is required to use the above sparkline functions. Users must add the DLL file in the project manually.

Column, Line, and Wireless Sparkline

For an area sparkline, you need a column, line, and wireless sparkline and column, line, and wireless sparkline for adding other data and related properties that are made for your data visualization.

The following table lists the available sparkline functions and their descriptions:

Function	Description
Area	A range of cells that represents a sparkline. For example, A1:A10.
Column	A range of cells that represents a sparkline. For example, A1:A10.
Line	A range of cells that represents a sparkline. For example, A1:A10.
Wireless	A range of cells that represents a sparkline. For example, A1:A10.

The following table lists the available sparkline functions and their descriptions:

Function	Description
Area	A range of cells that represents a sparkline. For example, A1:A10.
Column	A range of cells that represents a sparkline. For example, A1:A10.
Line	A range of cells that represents a sparkline. For example, A1:A10.
Wireless	A range of cells that represents a sparkline. For example, A1:A10.

Area Sparkline

Area sparklines are presented as a line sparkline graph in which the area between the series and the X axis is filled with color. You can create an area sparkline by using the AREASPARKLINE formula.

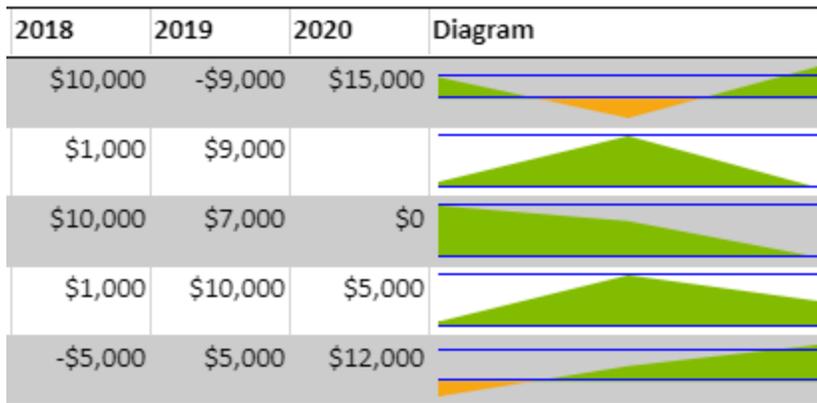
Using the Sparkline Designer

- Click the **Sparkline Designer** button on the ribbon.
- Click the **Area** button.
- Click the **Area** button.
- Click the **Area** button.

The **Sparkline Designer** dialog box is displayed. In the **Area** section, click the **Area** button. In the **Area** section, click the **Area** button. In the **Area** section, click the **Area** button.

Area Sparkline

Area sparklines are presented as a line sparkline graph in which the area between the series and the X axis is filled with color. You can create an area sparkline by using the AREASPARKLINE formula.



The area sparkline formula has the following syntax:

```
=AREASPARKLINE(points, [min, max, line1, line2, colorPositive, colorNegative, lineColor])
```

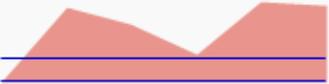
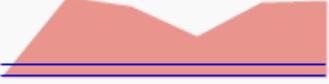
The formula options are described below:

Option	Description
points	A cell range. Invalid values are treated as 0.
min	A number that represents the minimum value of the sparkline.
<i>Optional</i>	
max	A number that represents the maximum value of the sparkline.
<i>Optional</i>	
line1	A number that represents a horizontal line's vertical position.
<i>Optional</i>	
line2	A number that represents a second horizontal line's vertical position.
<i>Optional</i>	
colorPositive	A string that represents the area color of positive values.
<i>Optional</i>	
colorNegative	A string that represents the area color of negative values.
<i>Optional</i>	
lineColor	A string that represents the color of the line.
<i>Optional</i>	

 **Note:** In case value of line is smaller than the actual minimum value, the line will not be painted.

Usage Scenario

Consider a scenario where an entertainment-focused website displays the number of monthly subscribers gained by US-based streaming services. An area sparkline can show the fluctuation throughout the first six months of the year and compare the viewership obtained by those services.

Monthly Subscribers of Streaming Services							
Stream	January	February	March	April	May	June	Diagram
	3,23,123	2,45,631	1,22,398	3,45,341	3,27,831	2,34,234	
	2,14,352	23,454	5,45,656	7,12,433	2,73,911	3,49,034	
	2,34,234	1,23,435	4,32,534	3,43,454	3,45,343	6,36,349	
	7,26,326	6,34,523	3,74,000	6,78,300	7,00,028	6,02,090	
	52,692	32,423	12,383	45,376	30,019	26,001	

C#

```
// Get sheet
var worksheet = fpSpread1_Sheet1.AsWorksheet();

// Set data
fpSpread1_Sheet1.AsWorksheet().SetValue(1, 0, new object[,]
{
    {"Stream", "January", "February", "March", "April", "May", "June", "Diagram"},
    {"Netflix", 323123, 245631, 122398, 345341, 327831, 234234, null},
    {"Prime Video", 214352, 23454, 545656, 712433, 273911, 349034, null},
    {"Disney Hotstar", 234234, 123435, 432534, 343454, 345343, 636349, null},
    {"HBO", 726326, 634523, 374000, 678300, 700028, 602090, null},
    {"Hulu", 52692, 32423, 12383, 45376, 30019, 26001, null}
});

// Set backColor for cells
worksheet.Cells["A1"].Interior.Color =
GrapeCity.Spreadsheet.Color.FromArgb(unchecked((int)0xFFD9A7A7));
worksheet.Cells["A2:H2"].Interior.Color =
GrapeCity.Spreadsheet.Color.FromArgb(unchecked((int)0xFFE3C3C3));
worksheet.Cells["A3:H7"].Interior.Color =
GrapeCity.Spreadsheet.Color.FromArgb(unchecked((int)0xFFFF9F9F9));
fpSpread1_Sheet1.Cells["A1"].ForeColor = Color.FromArgb(checked((int)0xFF3D3131));
fpSpread1_Sheet1.Cells["A2:H2"].ForeColor = Color.FromArgb(checked((int)0xFF3D3131));

// Set AreaSparkline formula
worksheet.Cells["H3:H7"].Formula =
"=AreaSparkline(A3:F3,,,0,100000,\"#E9958E\", \"#f7a711\")";
```

Visual Basic

```
'Get sheet
Dim worksheet = FpSpread1_Sheet1.AsWorksheet()

'Set data
```

```

worksheet.SetValue(1, 0, New Object(,) {
{"Stream", "January", "February", "March", "April", "May", "June", "Diagram"},
{"Netflix", 323123, 245631, 122398, 345341, 327831, 234234, Nothing},
{"Prime Video", 214352, 23454, 545656, 712433, 273911, 349034, Nothing},
{"Disney Hotstar", 234234, 123435, 432534, 343454, 345343, 636349, Nothing},
{"HBO", 726326, 634523, 374000, 678300, 700028, 602090, Nothing},
{"Hulu", 52692, 32423, 12383, 45376, 30019, 26001, Nothing}})

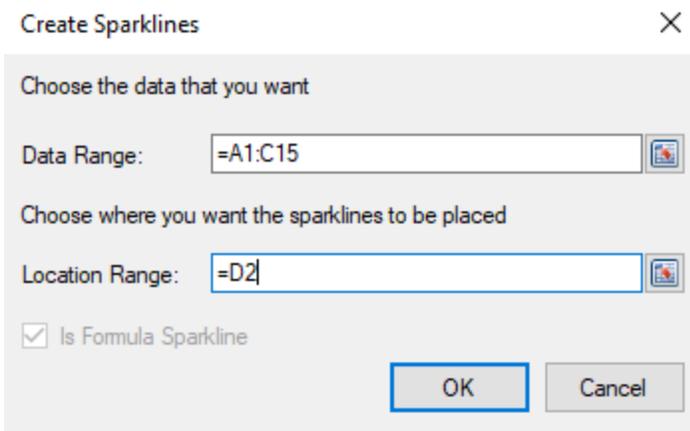
'Set backgroundColor for cells
worksheet.Cells("A1").Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFFD9A7A7)
worksheet.Cells("A2:H2").Interior.Color =
GrapeCity.Spreadsheet.Color.FromArgb(&HFFE3C3C3)
worksheet.Cells("A3:H7").Interior.Color =
GrapeCity.Spreadsheet.Color.FromArgb(&HFFF9F9F9)
FpSpread1_Sheet1.Cells("A1").ForeColor = Color.FromArgb(&HFF3D3131)
FpSpread1_Sheet1.Cells("A2:H2").ForeColor = Color.FromArgb(&HFF3D3131)

'Set AreaSparkline formula
worksheet.Cells("H3:H7").Formula =
"=Areasparkline(A3:F3,,,0,100000,""#E9958E"", ""#f7a711"")"

```

Using the Spread Designer

1. Type data in a cell or a column or row of cells in the designer.
2. Select a cell for the sparkline.
3. Select the Insert menu.
4. Select a sparkline type.
5. Set the Data Range in the **Create Sparklines** dialog (such as =Sheet1!\$E\$1:\$E\$3). Alternatively, set the range by selecting the cells in the range using the pointer.

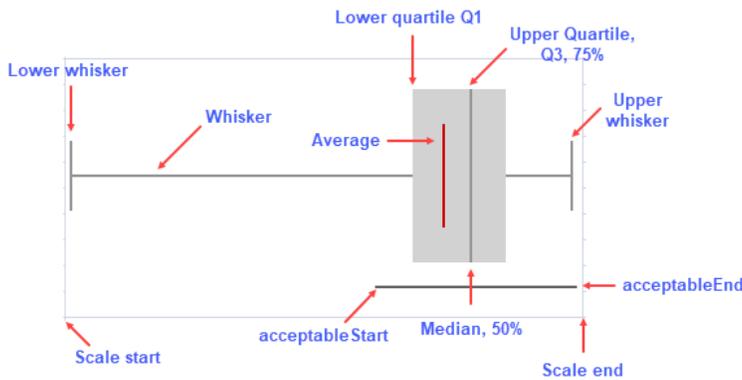


You can also set additional sparkline settings in the dialog if available.

6. Select OK.
7. Select Apply and Exit from the File menu to save your changes and close the designer.

BoxPlot Sparkline

A boxplot sparkline uses quartiles to display data and gives you a good indication of how the values in the data are spread out. It is a quick way of examining data sets graphically. Box plots are useful as they provide a visual summary of the data enabling researchers to quickly identify mean values, the dispersion of the data set, and signs of skews. The following image displays the sparkline areas.



The boxplot sparkline formula has the following syntax:

```
=BOXPLOTSPARKLINE(points, [boxPlotClass, showAverage, scaleStart, scaleEnd, acceptableStart, acceptableEnd, colorScheme, style, vertical])
```

The formula options are described below:

Option	Description
points	A reference that represents the cell range that contains the values, such as "A1:A4".
boxPlotClass	Q1 = 25% percentile, Q3 = 75% percentile, IQR (interquartile range) = Q3 - Q1.
<i>Optional</i>	5ns (default): Whisker ends at minimum and maximum, median, no outliers. 7ns: Whisker ends at 2% percentile and 98% percentile, hatch marks at 9% percentile and 91% percentile, outliers beyond 2% percentile and 98% percentile. tukey: Whisker ends at a value (the minimum of the points between Q1 and $Q1 - 1.5 * IQR$, use the point if it exists or use the minimum) and a value (the maximum of the points between Q3 and $Q3 + 1.5 * IQR$, use the point if it exists or use the maximum), outliers beyond $Q1 - 1.5 * IQR$ and $Q3 + 1.5 * IQR$, and extreme outliers beyond $Q1 - 3 * IQR$ and $Q3 + 3 * IQR$. bowley: Whisker ends at minimum and maximum, hatch marks at 10% percentile and 90% percentile, no outliers. sigma3: Whisker ends at a value ($average - 2 * StDev > scaleStart ? average - 2 * StDev : minimum$) and a value ($average + 2 * StDev < scaleEnd ? average + 2 * StDev : maximum$), box at $average +/- stdev$, outliers beyond $average - 2 * StDev$ and $average + 2 * StDev$, and extreme outliers beyond $average - 3 * StDev$ and $average + 3 * StDev$.
showAverage	A boolean that represents whether to show the average.
<i>Optional</i>	The default value is FALSE.
scaleStart	A number or reference that represents the minimum boundary of the sparkline, such as 1 or "A6".
<i>Optional</i>	The default value is the minimum of all values.
scaleEnd	A number or reference that represents the maximum boundary of the sparkline, such as 8 or "A7".
<i>Optional</i>	The default value is the maximum of all values.
acceptableStart	A number or reference that represents the start of the acceptable line, such as 3 or "A8".
<i>Optional</i>	The default value is None.
acceptableEnd	A number or reference represents the end of the acceptable line, such as 5 or "A9".
<i>Optional</i>	The default value is None.
colorScheme	A string that represents the color of the sparkline's box.
<i>Optional</i>	The default value is "#D2D2D2".
style	A number or reference that represents the sparkline style.
<i>Optional</i>	The style can be 0 or 1: <ul style="list-style-type: none"> 0: the whisker is a line and outlier is a circle. 1: the whisker is a rectangle and outlier is a line. The default value is 0 (Classical).
vertical	A boolean that represents whether to display the sparkline vertically.
<i>Optional</i>	The default value is FALSE.

Usage Scenario

Consider a scenario where a company, for example, Mescius wants to visualize the download count of different Spread products available throughout the year. A boxplot sparkline helps display the visual summary of a fiscal year in terms of the product downloads.

Number of downloads of Spread products													Diagram
Products	Apr' 20	May' 20	Jun' 20	Jul' 20	Aug' 20	Sep' 20	Oct' 20	Nov' 20	Dec' 20	Jan' 21	Feb' 21	Mar' 21	
Spread.NET	43,340	20,200	40,188	83,762	13,112	34,543	56,756	23,434	45,022	70,028	63,098	72,690	
SpreadJS	92,887	73,289	93,876	80,002	93,200	98,867	10,507	63,423	71,881	81,367	60,197	90,012	
Spread COM	1,292	3,411	565	1,002	915	1,301	451	891	505	537	791	618	
DataViewsJS	2,376	1,235	5,241	4,234	5,235	8,234	9,102	7,016	3,432	1,922	1,840	2,560	

C#

```
// Get sheet
var worksheet = fpSpread1.Sheets[0].AsWorksheet();

// Set data
worksheet.SetValue(1, 0, new object[,]
{
    {"Products", "Apr' 20", "May' 20", "Jun' 20", "Jul' 20", "Aug' 20", "Sep' 20", "Oct' 20", "Nov' 20", "Dec' 20", "Jan' 21", "Feb' 21", "Mar' 21", "Diagram"},
    {"Spread.NET", 43340, 20200, 40188, 83762, 13112, 34543, 56756, 23434, 45022, 70028, 63098, 72690, null},
    {"SpreadJS", 92887, 73289, 93876, 80002, 93200, 98867, 10507, 63423, 71881, 81367, 60197, 90012, null},
    {"Spread COM", 1292, 3411, 565, 1002, 915, 1301, 451, 891, 505, 537, 791, 618, null},
    {"DataViewsJS", 2376, 1235, 5241, 4234, 5235, 8234, 9102, 7016, 3432, 1922, 1840, 2560, null}
});

// Set formula for boxplotsparkline
worksheet.Cells["N3"].Formula = "BOXPLOTSPARKLINE (B3:M3, \"7ns\", TRUE, 0, 100000, 5000, 10000, \"#F58624\", 0, FALSE) ";
worksheet.Cells["N4"].Formula = "BOXPLOTSPARKLINE (B4:M4, \"5ns\", TRUE, 0, 100000, 2000, 80000, \"#F58624\", 0, FALSE) ";
worksheet.Cells["N5"].Formula = "BOXPLOTSPARKLINE (B5:M5, \"sigma3\", TRUE, 0, 2000, 100, 1000, \"#F58624\", 0, FALSE) ";
worksheet.Cells["N6"].Formula = "BOXPLOTSPARKLINE (B6:M6, \"bowley\", TRUE, 0, 10000, 100, 5000, \"#F58624\", 0, FALSE) ";
```

Visual Basic

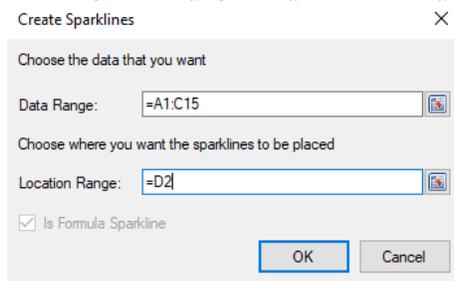
```
'Get sheet
Dim worksheet = FpSpread1_Sheet1.AsWorksheet()

'Set data
worksheet.SetValue(1, 0, New Object() {
    {"Products", "Apr' 20", "May' 20", "Jun' 20", "Jul' 20", "Aug' 20", "Sep' 20", "Oct' 20", "Nov' 20", "Dec' 20", "Jan' 21", "Feb' 21", "Mar' 21", "Diagram"},
    {"Spread.NET", 43340, 20200, 40188, 83762, 13112, 34543, 56756, 23434, 45022, 70028, 63098, 72690, Nothing},
    {"SpreadJS", 92887, 73289, 93876, 80002, 93200, 98867, 10507, 63423, 71881, 81367, 60197, 90012, Nothing},
    {"Spread COM", 1292, 3411, 565, 1002, 915, 1301, 451, 891, 505, 537, 791, 618, Nothing},
    {"DataViewsJS", 2376, 1235, 5241, 4234, 5235, 8234, 9102, 7016, 3432, 1922, 1840, 2560, Nothing}
})

'Set formula for BoxPlotSparkline
worksheet.Cells("N3").Formula = "BOXPLOTSPARKLINE (B3:M3, \"7ns\", TRUE, 0, 100000, 5000, 10000, \"#F58624\", 0, FALSE) "
worksheet.Cells("N4").Formula = "BOXPLOTSPARKLINE (B4:M4, \"5ns\", TRUE, 0, 100000, 2000, 80000, \"#F58624\", 0, FALSE) "
worksheet.Cells("N5").Formula = "BOXPLOTSPARKLINE (B5:M5, \"sigma3\", TRUE, 0, 2000, 100, 1000, \"#F58624\", 0, FALSE) "
worksheet.Cells("N6").Formula = "BOXPLOTSPARKLINE (B6:M6, \"bowley\", TRUE, 0, 10000, 100, 5000, \"#F58624\", 0, FALSE) "
```

Using the Spread Designer

1. Type data in a cell or a column or row of cells in the designer.
2. Select a cell for the sparkline.
3. Select the Insert menu.
4. Select a sparkline type.
5. Set the Data Range in the **Create Sparklines** dialog (such as =Sheet1!\$E\$1:\$E\$3). Alternatively, set the range by selecting the cells in the range using the pointer.

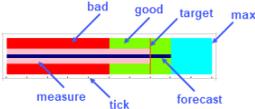


You can also set additional sparkline settings in the dialog if available.

6. Select OK.
7. Select Apply and Exit from the File menu to save your changes and close the designer.

Bullet Sparkline

A bullet sparkline is a variation of a bar graph where it can show a lot of data in a small amount of space. It is typically used when displaying performance data.



The bullet sparkline formula has the following syntax:

```
=BULLETSARKLINE(measure, target, max, [good, bad, forecast, tickunit, colorScheme, vertical, measureColor, targetColor, maxiColor, goodColor, badColor, forecastColor, allowMeasureOverMaxi, barSize])
```

The formula options are described below:

Option	Description
measure	A number or reference that represents the length of the measure bar, such as 5 or "A1".
target	A number or reference that represents the location of the target line, such as 7 or "A2".
max	A number or reference that represents the maximum value of the sparkline, such as 10 or "A3".
good	A number or reference that represents the length of the good bar, such as 3 or "A4".
Optional	The default value is 0.
bad	A number or reference that represents the length of the bad bar, such as 1 or "A5".
Optional	The default value is 0.
forecast	A number or reference that represents the length of the forecast line, such as 8 or "A6".
Optional	The default value is 0.
tickunit	A number or reference that represents the tick unit, such as 1 or "A7".
Optional	The default value is 0.
colorScheme	A string that represents a color scheme for displaying the sparkline.
Optional	The default value is "#A0A0A0".
vertical	A boolean value that indicates whether to display or not the sparkline vertically.
Optional	The default value is false.
measureColor	A color string that indicates the color of measure bar.
Optional	
targetColor	A color string that indicates the color of target line.
Optional	
maxiColor	A color string that indicates the color of the maxi area.
Optional	
goodColor	A color string that indicates the color of the good area.
Optional	
badColor	A color string that indicates the color of the bad area.
Optional	
forecastColor	A color string that indicates the color of the forecast line.
Optional	
allowMeasureOverMaxi	A Boolean value that indicates whether the measure can exceed maxi area.
Optional	The default value of this parameter is false.
barSize	A number value greater than 0 and equal to or less than 1, which indicates the percentage of bar width or height according to the cell width or height.
Optional	

Usage Scenario

Consider a scenario where an electronic store wants to analyze the performance of its product sales through its defined measures. A bullet sparkline can showcase all the required factors, effectively shown in the image below.

Sales (Million \$)						
Product	Actual	Target	Poor	Satisfactory	Excellent	Diagram
Laptops	81	75	40	65	75	
Monitor	56	60	30	50	60	
Keyboard	90	80	50	74	80	

```
C#
// Get sheet
var worksheet = fpSpread1.Sheets[0].AsWorksheet();

// Set data
worksheet.SetValue(1, 0, new object[,]
{
    {"Product", "Actual", "Target", "Poor", "Satisfactory", "Excellent", "Diagram"},
    {"Laptops", 81, 75, 40, 65, 75, null},
    {"Monitor", 56, 60, 30, 50, 60, null},
    {"Keyboard", 90, 80, 50, 74, 80, null}
});

// Set BulletSparkline formula
worksheet.Cells[2, 6, 4, 6].Formula = "BULLETSARKLINE(B3, C3, F3, E3, D3, 100, 1, \"#####\", FALSE, \"#FF00CB\", \"#FF4500\", \"#00cccc\", \"#7FFF00\", \"#00cccc\", \"#000080\", TRUE, 0.5)";
```

Visual Basic

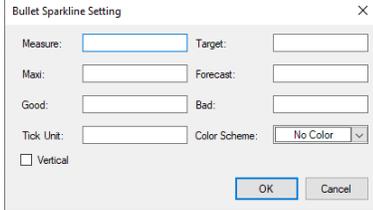
```
Visual Basic
'Get sheet
Dim worksheet = FpSpread1.Sheets(0).AsWorksheet()

'Set data
FpSpread1.Sheets(0).AsWorksheet().SetValue(1, 0, New Object(,) {
    {"Product", "Actual", "Target", "Poor", "Satisfactory", "Excellent", "Diagram"},
    {"Laptops", 81, 75, 40, 65, 75, Nothing},
    {"Monitor", 56, 60, 30, 50, 60, Nothing},
    {"Keyboard", 90, 80, 50, 74, 80, Nothing}
})
```

```
'Set BulletSparkline formula
FpSpread1.Sheets(0).AsWorksheet().Cells(2, 6, 4, 6).Formula = "BULLETPARKLINE(B3, C3, F3, E3, D3, 100, 1, ""#FFFFFF", FALSE, ""#FFC0CB"", ""#FF4500"", ""#00cccc"", ""#7FFF00"", ""#00cccc"", ""#000080"", TRUE, 0.5)"
```

Using the Spread Designer

1. Select a cell for the sparkline.
2. Select the Insert menu.
3. Select a sparkline type.
4. Set the Measure, Target, and Maxi values in the **Sparkline Setting** dialog.

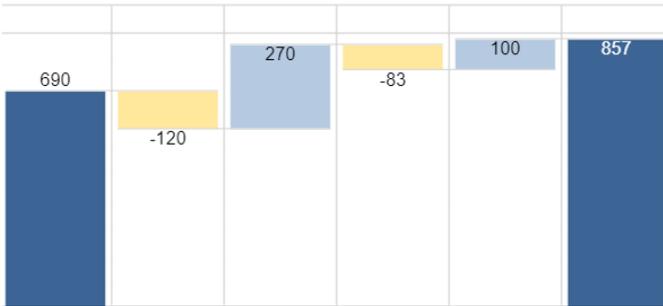


Set the additional sparkline settings as shown in the image above.

5. Select OK.
6. Select Apply and Exit from the File menu to save your changes and close the designer.

Cascade Sparkline

A cascade sparkline is generally used to analyze a value over time and shows the progressive changes between two values like yearly sales, total profit, net tax etc. It is used widely in finance, sales, legal and construction sectors, to name a few.



The cascade sparkline formula is used with following parameters:

```
=CASCADESPARKLINE(pointsRange, [pointIndex, labelsRange, minimum, maximum, colorPositive, colorNegative, vertical, itemTypeRange, colorTotal, colorTransition])
```

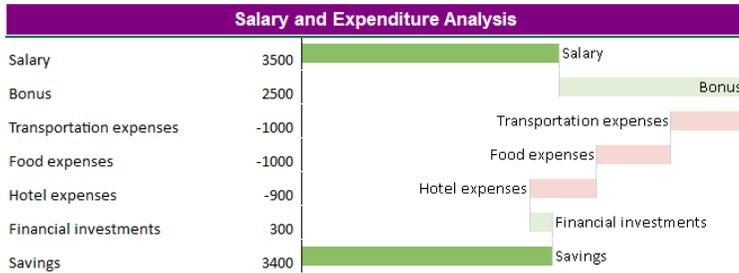
The formula options are described below:

Option	Description
pointsRange	A reference that represents the range of cells that contains values, such as "B2:B8".
pointIndex	A number or reference that represents the points index. The pointIndex is >= 1 such as 1 or "D2".
Optional	
labelsRange	A reference that represents the range of cells that contains the labels, such as "A2:A8".
Optional	The default value is no label.
minimum	A number or reference that represents the minimum values of the display area.
Optional	The default value is the minimum of the sum (the sum of the points' value), such as -2000. The minimum set must be less than the default minimum; otherwise, the default minimum is used.
maximum	A number or reference that represents the maximum values of the display area.
Optional	The default value is the maximum of the sum (the sum of the points' value), such as 6000. The maximum set must be greater than the default maximum; otherwise, the default maximum is used.
colorPositive	A string that represents the color of the first or last positive sparkline's box (this point's value is positive).
Optional	The default value is "#8CBF64". If the first or last box represents a positive value, the box's color is set to colorPositive. The middle positive box is set to a lighter color than colorPositive.
colorNegative	A string that represents the color of the first or last negative sparkline's box (this point's value is negative).
Optional	The default value is "#D6604D". If the first or last box represents the negative value, the box's color is set to colorNegative. The middle negative box is set to a lighter color than colorNegative.
vertical	A boolean that represents whether the box's direction is vertical or horizontal.
Optional	The default value is FALSE. You must set vertical to true or false for a group of formulas, because all the formulas represent the entire sparkline.
itemTypeRange	An array or reference that represents all the item types of the data range.
Optional	The values should be {"-", "+", "="} or "A1:A7" that reference the value of {"+", "-", "="}, where "+" indicates positive change, "-" indicates negative change and "=" indicates total columns.
colorTotal	A string that either represents the color of the last sparkline's box when itemTypeRange does not exist or represents the color of the resulting sparkline's box when itemTypeRange exists. This setting is optional.
Optional	

`colorTransition` Indicates the color transition of middle items. The value is in range of -1 to 1.
Optional The default value is 0.5

Usage Scenario

Consider a scenario where a salesperson wants to analyze their monthly salary and expenditure costs. A Cascade sparkline can depict how their finances go through progressive changes in a month.



```
C#
// Get sheet
var worksheet = fpSpread1_Sheet1.AsWorksheet();

// Set data for sparkline
worksheet.SetValue(1, 0, "Salary");
worksheet.SetValue(2, 0, "Bonus");
worksheet.SetValue(3, 0, "Transportation expenses");
worksheet.SetValue(4, 0, "Food expenses");
worksheet.SetValue(5, 0, "Hotel expenses");
worksheet.SetValue(6, 0, "Financial investments");
worksheet.SetValue(7, 0, "Savings");
worksheet.SetValue(1, 1, 3500);
worksheet.SetValue(2, 1, 2500);
worksheet.SetValue(3, 1, -1000);
worksheet.SetValue(4, 1, -1000);
worksheet.SetValue(5, 1, -900);
worksheet.SetValue(6, 1, 300);
worksheet.Cells[7, 1].Formula = "Sum(B2:B7)";

// Add formula for cascadesparkline
worksheet.Cells[1, 2].Formula2 = "CASCADESPARKLINE(B2:B8,,A2:A8,,\"#8CBF64\", \"#D6604D\", false)";

// Set bgcolor and forecolor for cells
worksheet.Cells["A1"].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(unchecked((int)0xFF800080));
fpSpread1_Sheet1.Cells["A1"].ForeColor = Color.FromArgb(unchecked((int)0xFFFFFFFF));
```

Visual Basic

```
'Get sheet
Dim worksheet = FpSpread1.ActiveSheet.AsWorksheet()

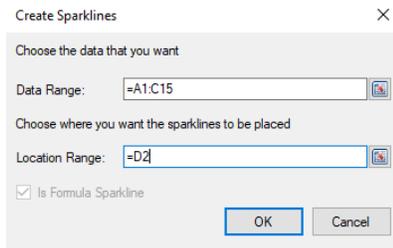
'Set data for sparkline
worksheet.SetValue(1, 0, "Salary")
worksheet.SetValue(2, 0, "Bonus")
worksheet.SetValue(3, 0, "Transportation expenses")
worksheet.SetValue(4, 0, "Food expenses")
worksheet.SetValue(5, 0, "Hotel expenses")
worksheet.SetValue(6, 0, "Financial investments")
worksheet.SetValue(7, 0, "Savings")
worksheet.SetValue(1, 1, 3500)
worksheet.SetValue(2, 1, 2500)
worksheet.SetValue(3, 1, -1000)
worksheet.SetValue(4, 1, -1000)
worksheet.SetValue(5, 1, -900)
worksheet.SetValue(6, 1, 300)
worksheet.Cells(7, 1).Formula = "Sum(B2:B7)"

'Add formula for cascadesparkline
worksheet.Cells(1, 2).Formula2 = "CASCADESPARKLINE(B2:B8,,A2:A8,,\"#8CBF64\", \"#D6604D\", false)"

'Set bgcolor and forecolor for cells
worksheet.Cells("A1").Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFF800080)
FpSpread1_Sheet1.Cells("A1").ForeColor = Color.FromArgb(&HFFFFFFFF)
```

Using the Spread Designer

1. Type data in a cell or a column or row of cells in the designer.
2. Select a cell for the sparkline.
3. Select the Insert menu.
4. Select a sparkline type.
5. Set the Data Range in the **Create Sparklines** dialog (such as =Sheet1!\$E\$1:\$E\$3).
 Alternatively, set the range by selecting the cells in the range using the pointer.

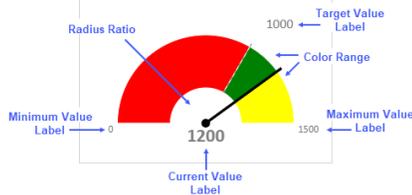


You can also set additional sparkline settings in the dialog if available.

6. Select OK.
7. Select Apply and Exit from the File menu to save your changes and close the designer.

Gauge KPI Sparkline

Spread for Winforms provides gauge KPI sparkline which can be used to visualize the performance of metrics with respect to KPI values. It can be used to indicate the effectiveness of management work or present sales targets etc.

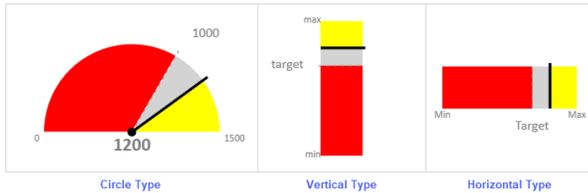


Gauge KPI sparkline uses the following values to display metrics:

- Target Value: Indicates the expected value and displays it at the bottom.
- Current Value: Indicates the current value.
- Minimum Value: Indicates the smallest expected value. It should be lower than the target value and current value.
- Maximum Value: Indicates the biggest expected value. It should be higher than the target value and current value.

Types of Gauge KPI Sparklines

There are three types of Gauge KPI sparklines, namely, Circle, Vertical, and Horizontal which can be set by using the **gaugeType** option in the GaugeKPISparkline function.



The gauge KPI sparkline formula has the following syntax:

```
=GAUGEKPISPARKLINE (targetValue, currentValue, minValue, maxValue, [showLabel, targetValueLabel, currentValueLabel, minValueLabel, maxValueLabel, fontArray, minAngle, maxAngle, radiusRatio, gaugeType, colorRange])
```

The formula options are described below:

Argument	Type	Description
targetValue	Number	The target value of the gauge KPI sparkline. The target value is between minValue and maxValue.
currentValue	Number	The current value of the gauge KPI sparkline. The current value is between minValue and maxValue.
minValue	Number	The minimum value of the gauge KPI sparkline. The minValue is less than maxValue.
maxValue	Number	The maximum value of the gauge KPI sparkline. The maxValue is more than minValue.
showLabel	Boolean	Specifies whether to show the label of all the values provided in the sparkline. If false, it will not show labels.
<i>Optional</i>		If true, it will only show the labels which fit inside the cell width and height. The cell should have enough width and height to show both graph and labels. The default value is true.
targetValueLabel	String	The string to display as target value label.
<i>Optional</i>		The default value is targetValue.
currentValueLabel	String	The string to display as current value label.
<i>Optional</i>		The default value is currentValue.
minValueLabel	String	The string to display as minimum value label.
<i>Optional</i>		The default value is minValue.
maxValueLabel	String	The string to display as maximum value label.
<i>Optional</i>		The default value is maxValue.
fontArray	CalcArray	Array contains font format as string items for the four label types - <ul style="list-style-type: none"> • Target value label: Default value "16px Calibri" • Current value label: Default value "bold 22px Calibri" • Minimum and Maximum value labels: Default value "12px Calibri"
<i>Optional</i>		showLabel option must be true.
minAngle	Number	The minimum angle value of circle type. The minAngle should be less than maxAngle.
<i>Optional</i>		The angle values correspond to the time in the clock. 0 is 12 o'clock, -90 is 9 o'clock, 90 is 3 o'clock, and -180/180 is 6 o'clock. The default value is -90. gaugeType option must be 0 (circle type).
maxAngle	Number	The maximum angle value of circle type. The maxAngle should be bigger than minAngle.
<i>Optional</i>		The angle values correspond to the time in the clock. 0 is 12 o'clock, -90 is 9 o'clock, 90 is 3 o'clock, and -180/180 is 6 o'clock. The default value is 90. gaugeType option must be 0 (circle type).
radiusRatio	Number	The value is calculated as the ratio of inner circle radius and outer circle radius. The outer circle radius value is decided by the cell size.
<i>Optional</i>		The value ranges between 0 and 1. Default is 0. gaugeType option must be 0 (circle type).

Argument	Type	Description
gaugeType	Number	The KPI sparkline type.
Optional		0 - Circle 1 - Vertical Bar 2 - Horizontal Bar The default type is circle.
colorRange	CalcArray	The color range between specified values.
Optional		{startValue, endValue, color_string}
Repeatable		Where startValue is the starting value in a range, endValue is the ending value in a range, and the color_string is the color of the range between the two specified values. The startValue must be less than the endValue and both the values should be between minValue and maxValue. Default color range from minValue to maxValue is filled with light gray color.

Usage Scenario

Consider a scenario where a company wants to monitor its annual sales performance. A gauge KPI sparkline can be used to measure the revenue, profit, and sales metrics against their target and actual values. It helps to visualize the data efficiently and make meaningful deductions.



```

C#
// get sheet
var worksheet1 = fpSpread1.Sheets[0].AsWorksheet();
var worksheet2 = fpSpread1.Sheets[1].AsWorksheet();

// set data
worksheet2.SetValue(0, 0, new object[,]
{
    { "Parameters", "Target", "Current", "Min", "Max" },
    { "Revenue", 900, 1200, 0, 2000 },
    { "Profit", 1200, 1100, 0, 2000 },
    { "Sales", 1600, 1800, 0, 2000 }
});

// set GaugeKPISparkline formula
worksheet1.Cells["A2"].Formula = "GAUGEKPISPARKLINE(Sheet2!B2, Sheet2!C2, Sheet2!D2, Sheet2!E2, TRUE, TEXT(Sheet2!B2/1000, \"\$0.0K\"), Sheet2!A2, TEXT(Sheet2!D2/1000, \"\$0.0K\"), TEXT(Sheet2!E2/1000, \"\$0.0K\"), , -90, 90, 0, 4, 0, {0, 1200, \"#FFB2BD\"}, {1200, 1500, \"#FFDFB0\"}, {1500, 2000, \"#BCEAB4\"})";
worksheet1.Cells["B2"].Formula = "GAUGEKPISPARKLINE(Sheet2!B3, Sheet2!C3, Sheet2!D3, Sheet2!E3, TRUE, TEXT(Sheet2!B3/1000, \"\$0.0K\"), Sheet2!A3, TEXT(Sheet2!D3/1000, \"\$0.0K\"), TEXT(Sheet2!E3/1000, \"\$0.0K\"), , -90, 90, 0, 4, 0, {0, 1200, \"#FFB2BD\"}, {1200, 1500, \"#FFDFB0\"}, {1500, 2000, \"#BCEAB4\"})";
worksheet1.Cells["C2"].Formula = "GAUGEKPISPARKLINE(Sheet2!B4, Sheet2!C4, Sheet2!D4, Sheet2!E4, TRUE, TEXT(Sheet2!B4/1000, \"\$0.0K\"), Sheet2!A4, TEXT(Sheet2!D4/1000, \"\$0.0K\"), TEXT(Sheet2!E4/1000, \"\$0.0K\"), , -90, 90, 0, 4, 0, {0, 1200, \"#FFB2BD\"}, {1200, 1500, \"#FFDFB0\"}, {1500, 2000, \"#BCEAB4\"})";
    
```

Visual Basic

```

'Get sheet
Dim worksheet1 = FpSpread1.Sheets(0).AsWorksheet()
Dim worksheet2 = FpSpread1.Sheets(1).AsWorksheet()

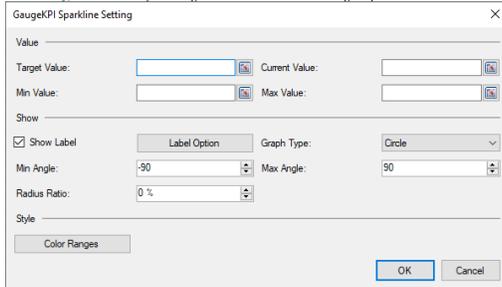
'Set data
worksheet2.SetValue(0, 0, New Object() {
    { "Parameters", "Target", "Current", "Min", "Max" },
    { "Revenue", 900, 1200, 0, 2000 },
    { "Profit", 1200, 1100, 0, 2000 },
    { "Sales", 1600, 1800, 0, 2000 }
})

'Set GaugeKPISparkline formula
worksheet1.Cells("A2").Formula = "GAUGEKPISPARKLINE(Sheet2!B2, Sheet2!C2, Sheet2!D2, Sheet2!E2, TRUE, TEXT(Sheet2!B2/1000, \"\$0.0K\"), Sheet2!A2, TEXT(Sheet2!D2/1000, \"\$0.0K\"), TEXT(Sheet2!E2/1000, \"\$0.0K\"), , -90, 90, 0, 4, 0, {0, 1200, \"#FFB2BD\"}, {1200, 1500, \"#FFDFB0\"}, {1500, 2000, \"#BCEAB4\"})";
worksheet1.Cells("B2").Formula = "GAUGEKPISPARKLINE(Sheet2!B3, Sheet2!C3, Sheet2!D3, Sheet2!E3, True, TEXT(Sheet2!B3/1000, \"\$0.0K\"), Sheet2!A3, TEXT(Sheet2!D3/1000, \"\$0.0K\"), TEXT(Sheet2!E3/1000, \"\$0.0K\"), , -90, 90, 0, 4, 0, {0, 1200, \"#FFB2BD\"}, {1200, 1500, \"#FFDFB0\"}, {1500, 2000, \"#BCEAB4\"})";
worksheet1.Cells("C2").Formula = "GAUGEKPISPARKLINE(Sheet2!B4, Sheet2!C4, Sheet2!D4, Sheet2!E4, True, TEXT(Sheet2!B4/1000, \"\$0.0K\"), Sheet2!A4, TEXT(Sheet2!D4/1000, \"\$0.0K\"), TEXT(Sheet2!E4/1000, \"\$0.0K\"), , -90, 90, 0, 4, 0, {0, 1200, \"#FFB2BD\"}, {1200, 1500, \"#FFDFB0\"}, {1500, 2000, \"#BCEAB4\"})";
    
```

Using the Spread Designer

1. Type the different values in cells or a column or row of cells in the designer.
2. Select a cell for the sparkline.
3. Select the Insert menu.
4. Select a sparkline type.
5. Set the Values in the **Sparkline Setting** dialog.

Alternatively, set the values by selecting the cells in the worksheet using the pointer.



Set the additional sparkline settings as shown in the image above.

6. Select OK.
7. Select Apply and Exit from the File menu to save your changes and close the designer.

Hbar and Vbar Sparkline

Hbar and vbar sparklines present categorical data with rectangular bars with heights or lengths proportional to the values that they represent. These sparklines can be used to show variations or ranges in the given data.



The sparkline starts at the left or bottom of the cell for positive values and the top or right of the cell for negative values. If the value is greater than 100% or smaller than -100%, an arrow is displayed.

The hbar and vbar sparkline formulas have the following syntax:

```
=HBARSARKLINE(value, [colorScheme, axisVisible, barHeight, minimum, maximum, axisValue])
=VBARSARKLINE(value, [colorScheme, axisVisible, barWidth, minimum, maximum, axisValue])
```

The formula options are described below:

Option

- value
colorScheme
Optional
- axisVisible
Optional
- barHeight (Hbar) or barWidth (Vbar)
Optional
- minimum
Optional
- maximum
Optional
- axisValue
Optional

Description

- A number or reference that represents the length of the bar. The value should be between -1 and 1.
- A string that represents the color of the bar.
The default value is "grey".
- A Boolean value that indicates whether or not to show the axis.
The default value is true.
- A number greater than 0 and less than or equal to 1, which indicates the percentage of bar height or bar width according to the cell height or cell width.
Default value is 0.7
- A number that represents the minimum axis value.
The default value is 0 if the value is greater than 0, or -1 if the value is less than 0.
- A number that represents the maximum axis value.
The default value is 1 if the value is greater than 0, or 0 if the value is less than 0.
- A number that represents the axis intercept value (where the axis line is drawn). The value should be between minimum and maximum values.
If the value is out of range, it will be adjusted to the minimum or maximum value.
The default value is 0.

Usage Scenario

Consider a scenario where a company wants to display the employee satisfaction scores against different aspects of work. The hbar and vbar sparklines can show variations between each category provided during the company survey.

Hbar Sparkline



Vbar Sparkline



```

C#
// Get sheet
var worksheet1 = fpSpread1.Sheets[0].AsWorksheet();
var worksheet2 = fpSpread1.Sheets[1].AsWorksheet();

// Set data for HBarSparkline
worksheet1.SetValue(1, 0, new object[] {
    ("Commute", 0.8, null),
    ("Job Security", 0.61, null),
    ("Health Plan", 0.45, null),
    ("Work/ Life Balance", 0.42, null),
    ("Growth Potential", 0.39, null),
    ("Flexible Time Plan", 0.39, null),
    ("Training Programs", 0.36, null),
    ("Promotion Policy", 0.31, null),
    ("Bonus Plan", 0.29, null)
});

// Set number format
worksheet1.Cells["B2:B10"].NumberFormat = "0%";

// Set HBarSparkline formula
worksheet1.Cells["C2:C13"].Formula =
"=IF(A2>=0.8, HBARSARKLINE(B2, ""#092834"", TRUE, B2), IF(B2>=0.6, HBARSARKLINE(B2, ""#B2D732"", TRUE, B2), IF(B2>=0.4, HBARSARKLINE(B2, ""#66B032"", TRUE, B2), IF(B2>=0.2, HBARSARKLINE(B2, ""#B2D732"", TRUE, B2), IF(B2>=0, HBARSARKLINE(B2, ""#8E1963"", TRUE, B2), HBARSARKLINE(B2, ""red""))))))";

// Set data for VBarSparkline
worksheet2.SetValue(1, 0, new object[] {
    ("Commute", "Job Security", "Health Plan", "Work/ Life Balance", "Growth Potential", "Flexible Time Plan", "Training Programs", "Promotion Policy", "Bonus Plan" },
});

worksheet2.SetValue(3, 0, new object[] {
    (0.80, 0.61, 0.45, 0.42, 0.39, 0.39, 0.36, 0.31, 0.29)
});

// Set VBarSparkline formula
worksheet2.Cells["A3:A13"].Formula =
"=IF(A3>=0.8, VBARSARKLINE(A4, ""#092834"", TRUE, A4), IF(A4>=0.6, VBARSARKLINE(A4, ""#B2D732"", TRUE, A4), IF(A4>=0.4, VBARSARKLINE(A4, ""#66B032"", TRUE, A4), IF(A4>=0.2, VBARSARKLINE(A4, ""#B2D732"", TRUE, A4), IF(A4>=0, VBARSARKLINE(A4, ""#8E1963"", TRUE, A4), VBARSARKLINE(A4, ""red""))))))";
    
```

Visual Basic

```

// Get sheet
Dim worksheet1 = fpSpread1.Sheets(0).AsWorksheet()
Dim worksheet2 = fpSpread1.Sheets(1).AsWorksheet()

// Set data for HBarSparkline
worksheet1.SetValue(1, 0, New Object() {
    ("Commute", 0.8, Nothing),
    ("Job Security", 0.61, Nothing),
    ("Health Plan", 0.45, Nothing),
    ("Work/ Life Balance", 0.42, Nothing),
    ("Growth Potential", 0.39, Nothing),
    ("Flexible Time Plan", 0.39, Nothing),
    ("Training Programs", 0.36, Nothing),
    ("Promotion Policy", 0.31, Nothing),
    ("Bonus Plan", 0.29, Nothing)
})

// Set number format
worksheet1.Cells["B2:B10"].NumberFormat = "0%"

// Set HBarSparkline formula
worksheet1.AsWorksheet().Cells("C2:C13").Formula =
"=IF(A2>=0.8, HBARSARKLINE(A4, ""#092834"", TRUE, B2), IF(B2>=0.6, HBARSARKLINE(B2, ""#B2D732"", TRUE, B2), IF(B2>=0.4, HBARSARKLINE(B2, ""#66B032"", TRUE, B2), IF(B2>=0.2, HBARSARKLINE(B2, ""#B2D732"", TRUE, B2), IF(B2>=0, HBARSARKLINE(B2, ""#8E1963"", TRUE, B2), HBARSARKLINE(B2, ""red""))))))";

// Set data for VBarSparkline
worksheet2.SetValue(1, 0, New Object() {
    ("Commute", "Job Security", "Health Plan", "Work/ Life Balance", "Growth Potential", "Flexible Time Plan", "Training Programs", "Promotion Policy", "Bonus Plan" )
})

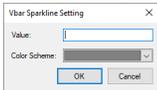
worksheet2.SetValue(3, 0, New Object() {
    (0.8, 0.61, 0.45, 0.42, 0.39, 0.39, 0.36, 0.31, 0.29)
})

// Set VBarSparkline formula
worksheet2.Cells("A3:A13").Formula =
"=IF(A3>=0.8, VBARSARKLINE(A4, ""#092834"", TRUE, A4), IF(A4>=0.6, VBARSARKLINE(A4, ""#B2D732"", TRUE, A4), IF(A4>=0.4, VBARSARKLINE(A4, ""#66B032"", TRUE, A4), IF(A4>=0.2, VBARSARKLINE(A4, ""#B2D732"", TRUE, A4), IF(A4>=0, VBARSARKLINE(A4, ""#8E1963"", TRUE, A4), VBARSARKLINE(A4, ""red""))))))";
    
```

Using the Spread Designer

1. Select a cell for the sparkline.
2. Select the Insert menu.
3. Select a sparkline type.
4. Set the Value in the Sparkline Setting dialog.

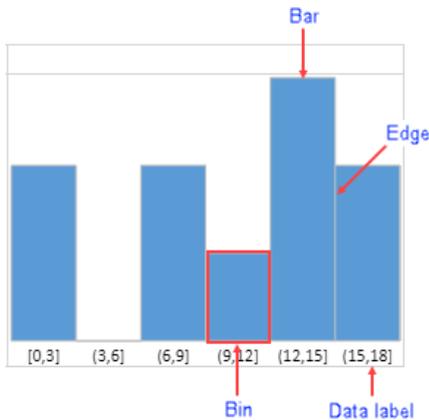




Set the additional sparkline settings as shown in the images above.
 5. Select OK.
 6. Select Apply and Exit from the File menu to save your changes and close the designer.

Histogram Sparkline

Histograms are used to represent the frequency distribution of a data set. Spread for Winforms allows you to create histogram sparklines by using the histogram sparkline function.



The above image shows a histogram sparkline and its elements:

- **Bin:** The data container which contains data of the specified range of values.
- **Bar:** The paint block responsible for UI. The bar height is determined by the following expression if the paintLabel option is true:

```
(cellRowHeight - labelFontSize - 6px)
```

Otherwise it is determined by:

```
(cellRowHeight - 6px)
```

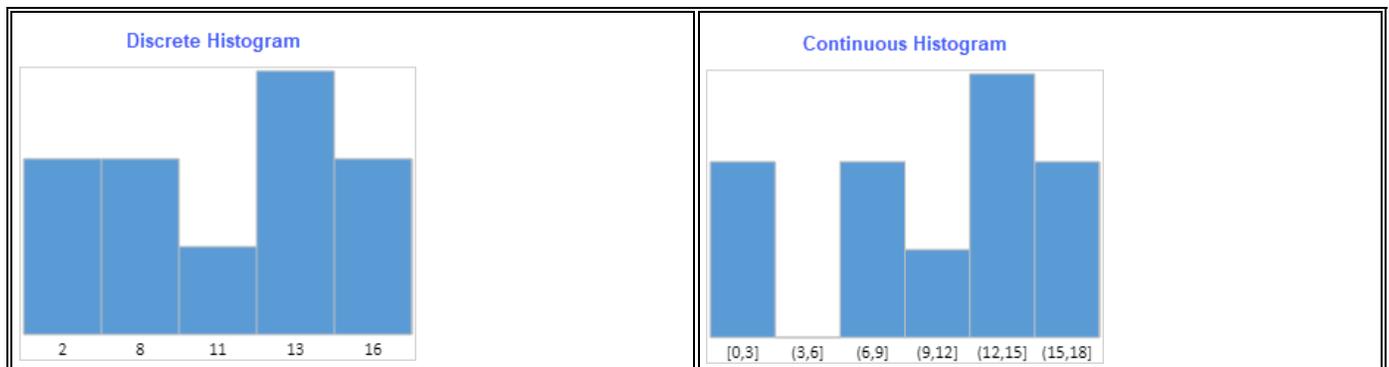
- **Edge:** The border of bar.
- **Data Label:** The data range displayed in the sparkline. The data label font size is determined by the expression:

```
Math.floor (cellRowHeight/3)
```

- If the font size is more than the expression, then the largest integer less than or equal to the expression is taken as the font size.
- If the height of the cell is less than cell row height, the font size is 12px.

Types of Histogram Sparklines

The Histogram sparklines can be displayed in a discrete or continuous manner as shown below:



Discrete Histogram represents the data in a discrete manner (without continuous intervals) by taking existing values. For example: 20, 34, 38. The values are painted in ascending order.

Continuous Histogram represents the data by taking a range of continuous values. The intervals for the first bin are left-closed and right-closed, whereas intervals for rest of the bins are left-open and right-closed.

- The lower bound is calculated using:

```
Math.floor(minValue/scale)*scale
```

Whereas the upper bound is calculated using:

```
Math.ceil(maxValue/scale)*scale
```

- The width of bin is 1, by default
- If the scale is lower than 0, the sparkline is set to 1

The type of histogram can be specified by using the **continuous** option in the HistogramSparkline formula.

The histogram sparkline formula has the following syntax:

```
=HISTOGRAMSPARKLINE(dataRange, [continuous, paintLabel, scale, barWidth, barColor, labelFontStyle, labelColor, edgeColor])
```

The formula options are described below:

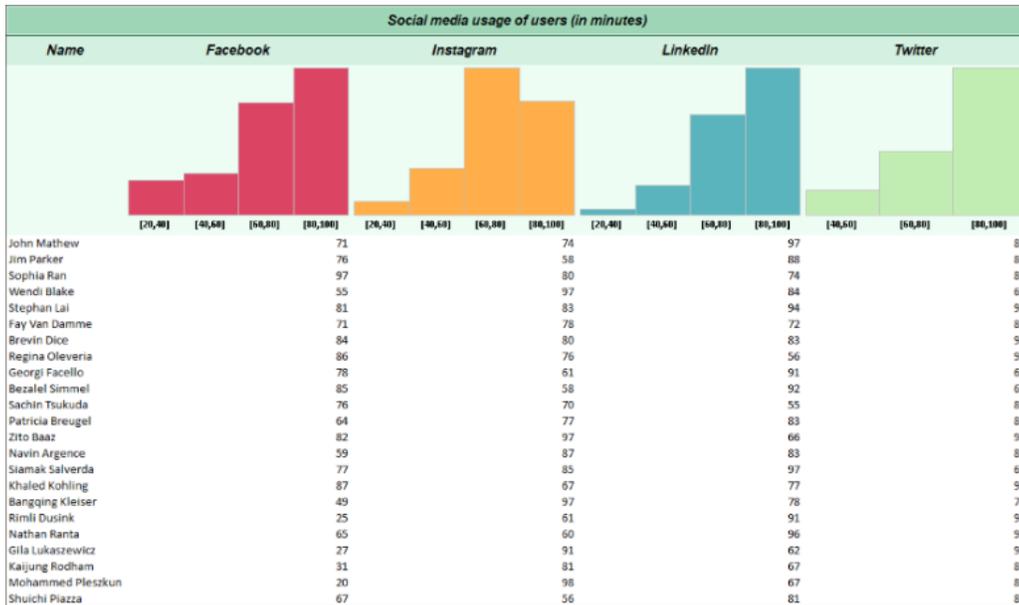
Option	Description
dataRange	Specifies the range of data sources. It supports a range. It supports calc array and calc reference.
continuous <i>Optional</i>	If set to true, the sparkline is a continuous histogram. If set to false, the sparkline is a discrete histogram. The default value is true.
paintLabel <i>Optional</i>	If set to true, the sparkline paints the data label. The default value is false.
scale <i>Optional</i>	Specifies the bin width. This value is useful when histogram type is continuous. The default value is false.
barWidth <i>Optional</i>	Specifies the bar width. Default value is 1. Bar width = auto calc width * barWidth Value range is 0 < value ≤ 1.
barColor <i>Optional</i>	Specifies the color of the bar. Default value is '5B9BD5'. Supports CSS color property.
labelFontStyle <i>Optional</i>	Specifies the font style of the data label font. The default value is 9pt Calibri. Supports CSS font property. It supports font-style, font-weight, font-size, font-family.
labelColor <i>Optional</i>	Specifies the font color of data label font. Default is black (#000000). Supports CSS color string.
edgeColor <i>Optional</i>	Specifies the color of the edge. Default value is silver (#CoCoCo). Supports CSS color property.

Notes

- The default padding of sparkline is 3px.
- The paintLabel option does not paint the data label when the custom font size is more than one-third of cell row height.
- The data label does not paint when any of the data label's width is bigger than the bar's average width. The average width is determined by the expression (cell width/bar count). For example, for 100px cell width, there are 5 bars in the sparkline. The bar average width is 20px. In this sparkline, if any data label is wider than 20px, the data label does not paint.
- In the data range, any non-number type value is ignored.

Usage Scenario

Consider a scenario where an organization has conducted a survey of social media consumption by users on a weekly basis. The histogram sparkline can depict these statistics and analyze the trends about how much time users spent on different social media platforms.



C#

```
// Get sheet
var worksheet = fpSpread1.Sheets[0].AsWorksheet();

// Set data for histogram sparkline
worksheet.SetValue(1, 0, new object[,]
{
    { "Name", "Facebook", "Instagram", "LinkedIn", "Twitter" }
});

worksheet.SetValue(3, 0, new object[,]
{
    {"John Mathew", 71, 74, 97, 85},
    {"Jim Parker", 76, 58, 88, 84},
    {"Sophia Ran", 97, 80, 74, 80},
    {"Wendi Blake", 55, 97, 84, 60},
    {"Stephan Lai", 81, 83, 94, 90},
    {"Fay Van Damme", 71, 78, 72, 82},
    {"Brevin Dice", 84, 80, 83, 93},
    {"Regina Oleveria", 86, 76, 56, 92},
    {"Georgi Facello", 78, 61, 91, 68},
    {"Bezalel Simmel", 85, 58, 92, 68},
    {"Sachin Tsukuda", 76, 70, 55, 81},
    {"Patricia Breugel", 64, 77, 83, 85},
    {"Zito Baaz", 82, 97, 66, 90},
    {"Navin Argence", 59, 87, 83, 85},
    {"Siamak Salverda", 77, 85, 97, 65},
    {"Khaled Kohling", 87, 67, 77, 97},
    {"Bangqing Kleiser", 49, 97, 78, 75},
    {"Rimli Dusink", 25, 61, 91, 91},
    {"Nathan Ranta", 65, 60, 96, 92},
    {"Gila Lukaszewicz", 27, 91, 62, 93},
    {"Kaijung Rodham", 31, 81, 67, 87},
    {"Mohammed Pleszkun", 20, 98, 67, 82},
    {"Shuichi Piazza", 67, 56, 81, 82},
    {"Katsuo Leuchs", 83, 74, 51, 42},
    {"Masanao Ducloy", 67, 64, 58, 73},
    {"Mihalis Crabtree", 80, 91, 57, 84},
    {"Danny Lenart", 92, 38, 99, 86},
    {"Yongqiao Dalton", 85, 61, 73, 81},
    {"Gaetan Veldwijk", 64, 52, 76, 72},
    {"Leszek Pulkowski", 95, 75, 64, 99},
    {"Weidon Gente", 77, 88, 77, 96},
    {"Krister Stranks", 72, 38, 89, 55},
    {"Ziyad Baaz", 89, 81, 83, 48},
    {"Ymte Perelgut", 67, 70, 97, 94},
    {"Tonia Butner", 99, 71, 87, 76},
    {"Shigeaki Narlikar", 78, 80, 80, 97},
    {"Ayakannu Beerel", 86, 61, 85, 81},

```

```

        {"Moni Bale", 60, 72, 71, 86},
        {"Manohar Heemskerk", 97, 60, 75, 78},
        {"Angus Swan", 33, 97, 99, 99},
        {"Christ Murtagh", 91, 76, 73, 80},
        {"Maren Baez", 45, 77, 86, 88},
        {"Greger Jahnichen", 98, 89, 81, 63},
        {"Ymte Duclos", 95, 83, 72, 72},
        {"Chenyi Hainaut", 81, 97, 99, 87},
        {"Kasidit Picel", 87, 80, 88, 80},
        {"Elrique Walstra", 96, 54, 38, 54},
        {"Adel Reghbaty", 56, 88, 81, 88}
    });

// Set HistogramSparkline formulas
worksheet.Cells[2, 1].Formula = "HISTOGRAMSPARKLINE(Sheet1!B4: B51, true, true, 20, 1, \"#DC4463\", \"bold normal 10pt Calibri\", \"black\")";
worksheet.Cells[2, 2].Formula = "HISTOGRAMSPARKLINE(Sheet1!C4: C51, true, true, 20, 1, \"#FFAE49\", \"bold normal 10pt Calibri\", \"black\")";
worksheet.Cells[2, 3].Formula = "HISTOGRAMSPARKLINE(Sheet1!D4: D51, true, true, 20, 1, \"#5AB4BD\", \"bold normal 10pt Calibri\", \"black\")";
worksheet.Cells[2, 4].Formula = "HISTOGRAMSPARKLINE(Sheet1!E4: E51, true, true, 20, 1, \"#C2EDB2\", \"bold normal 10pt Calibri\", \"black\")";

```

Visual Basic

```

'Get sheet
Dim worksheet = FpSpread1.Sheets(0).AsWorksheet()

'Set data for histogram sparkline
worksheet.SetValue(1, 0, New Object(,) {
    {"Name", "Facebook", "Instagram", "LinkedIn", "Twitter"}
})

worksheet.SetValue(3, 0, New Object(,) {
    {"John Mathew", 71, 74, 97, 85},
    {"Jim Parker", 76, 58, 88, 84},
    {"Sophia Ran", 97, 80, 74, 80},
    {"Wendi Blake", 55, 97, 84, 60},
    {"Stephan Lai", 81, 83, 94, 90},
    {"Fay Van Damme", 71, 78, 72, 82},
    {"Brevin Dice", 84, 80, 83, 93},
    {"Regina Oleviera", 86, 76, 56, 92},
    {"Georgi Facello", 78, 61, 91, 68},
    {"Bezalel Simmel", 85, 58, 92, 68},
    {"Sachin Tsukuda", 76, 70, 55, 81},
    {"Patricia Breugel", 64, 77, 83, 85},
    {"Zito Baaz", 82, 97, 66, 90},
    {"Navin Argence", 59, 87, 83, 85},
    {"Siamak Salverda", 77, 85, 97, 65},
    {"Khaled Kohling", 87, 67, 77, 97},
    {"Bangqing Kleiser", 49, 97, 78, 75},
    {"Rimli Dusink", 25, 61, 91, 91},
    {"Nathan Ranta", 65, 60, 96, 92},
    {"Gila Lukaszewicz", 27, 91, 62, 93},
    {"Kaijung Rodham", 31, 81, 67, 87},
    {"Mohammed Pleszkun", 20, 98, 67, 82},
    {"Shuichi Piazza", 67, 56, 81, 82},
    {"Katsuo Leuchs", 83, 74, 51, 42},
    {"Masanao Ducloy", 67, 64, 58, 73},
    {"Mihalis Crabtree", 80, 91, 57, 84},
    {"Danny Lenart", 92, 38, 99, 86},
    {"Yongqiao Dalton", 85, 61, 73, 81},
    {"Gaetan Veldwijk", 64, 52, 76, 72},
    {"Leszek Pulkowski", 95, 75, 64, 99},
    {"Weidon Gente", 77, 88, 77, 96},
    {"Krister Stranks", 72, 38, 89, 55},
    {"Ziyad Baaz", 89, 81, 83, 48},
    {"Ymte Perelgut", 67, 70, 97, 94},
    {"Tonia Butner", 99, 71, 87, 76},
    {"Shigeaki Narlikar", 78, 80, 80, 97},
    {"Ayakannu Beerel", 86, 61, 85, 81},
    {"Moni Bale", 60, 72, 71, 86},
    {"Manohar Heemskerk", 97, 60, 75, 78},
    {"Angus Swan", 33, 97, 99, 99},
    {"Christ Murtagh", 91, 76, 73, 80},
    {"Maren Baez", 45, 77, 86, 88},
    {"Greger Jahnichen", 98, 89, 81, 63},

```

```

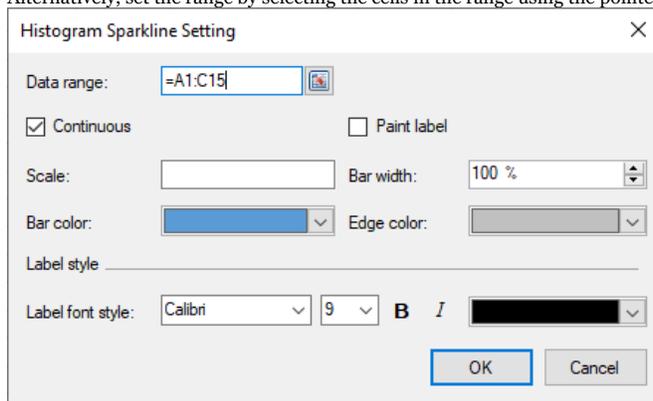
{"Ymte Duclos", 95, 83, 72, 72},
{"Chenyi Hainaut", 81, 97, 99, 87},
{"Kasidit Picel", 87, 80, 88, 80},
{"Elrique Walstra", 96, 54, 38, 54},
{"Adel Reghbati", 56, 88, 81, 88}
}))

'Set HistogramSparkline formulas
worksheet.Cells(2, 1).Formula = "HISTOGRAMSPARKLINE(Sheet1!B4: B51, true, true, 20, 1, ""#DC4463"", ""bold normal 10pt Calibri"", ""black"")"
worksheet.Cells(2, 2).Formula = "HISTOGRAMSPARKLINE(Sheet1!C4: C51, true, true, 20, 1, ""#FFAE49"", ""bold normal 10pt Calibri"", ""black"")"
worksheet.Cells(2, 3).Formula = "HISTOGRAMSPARKLINE(Sheet1!D4: D51, true, true, 20, 1, ""#5AB4BD"", ""bold normal 10pt Calibri"", ""black"")"
worksheet.Cells(2, 4).Formula = "HISTOGRAMSPARKLINE(Sheet1!E4: E51, true, true, 20, 1, ""#C2EDB2"", ""bold normal 10pt Calibri"", ""black"")"

```

Using the Spread Designer

1. Type data in a cell or a column or row of cells in the designer.
2. Select a cell for the sparkline.
3. Select the Insert menu.
4. Select a sparkline type.
5. Set the Data Range in the **Create Sparklines** dialog (such as =Sheet1!\$E\$1:\$E\$3). Alternatively, set the range by selecting the cells in the range using the pointer.



You can also set additional sparkline settings in the dialog if available.

6. Select OK.
7. Select Apply and Exit from the File menu to save your changes and close the designer.

Image Sparkline

The image sparkline can be used to place an image in a cell. The image can be displayed in different sizes by using display modes of image sparkline function:

	A	B
1	Keep aspect ratio to fit (1)	
2	Stretch to fit (2)	
3	Original Size (3)	
4	Custom Size 200x200 (4)	

The image sparkline formula has the following syntax:

```
=GC.IMAGE(url, [mode, height, width, clipX, clipY, clipHeight, clipWidth, vAlign, hAlign])
```

The formula options are described below:

Option	Description
URL	The location of the image on the web or base64 string.
mode	Specifies how to size the image.
<i>Optional</i>	<ul style="list-style-type: none"> • 1 - Keep the aspect ratio to fit the cell. • 2 - Stretch the image to cover the entire cell. • 3 - Keep original size even if cropped. • 4 - custom <p>The default value is 1.</p>
height	The height of image.
<i>Optional</i>	mode option must be 4.
width	The width of image.
<i>Optional</i>	mode option must be 4.
clipX	The x-axis coordinate of the top left corner of the source image sub-rectangle to draw into the destination context.
<i>Optional</i>	The default value is 0.
clipY	The y-axis coordinate of the top left corner of the source image sub-rectangle to draw into the destination context.
<i>Optional</i>	The default value is 0.
clipHeight	The height of the source image sub-rectangle to draw into the destination context.
<i>Optional</i>	The default value is the height of image.
clipWidth	The width of the source image sub-rectangle to draw into the destination context.
<i>Optional</i>	The default value is the width of image.
vAlign	Vertical alignment of the image.
<i>Optional</i>	0 - Top 1 - Center 2 - Bottom The default value is 1 (center).
hAlign	Horizontal alignment of the image.
<i>Optional</i>	0 - Left 1 - Center 2 - Right The default value is 1 (center).

Behavior with Different Values of Parameters

The following behavior is observed with certain parameter values in Image sparklines:

1. The height and width of the image must be specified when the mode is set to 4, otherwise, a blank cell is returned.
2. If the clipWidth value is not specified while setting clipX, the clipWidth value is set as (Image Width - clipX). The same applies for clipHeight and clipY arguments.
3. If clipX is greater than the image width, then the clipWidth is set to 0. Similarly, for clipY and image height.
4. If the mode, vAlign, or hAlign is set to an illegal value, the runtime sets the mode to 1.
5. The following parameters are replaced with 0, if they are set to a value smaller than 0:
Width, Height, ClipX, ClipY, ClipHeight, ClipWidth

Usage Scenario

Consider a scenario where a data accounting organization wants to present a list of 10 countries with the largest population in the world. The list can also display the images for country flags picked through a web URL using the image sparkline.

Top 10 Populated Countries				
Flag	Country	Rank	Population (2020)	Land Area (km sq)
	China	1	1,43,93,23,776	93,88,211
	India	2	1,38,00,04,385	29,73,190
	United States	3	33,10,02,651	91,47,420
	Indonesia	4	27,35,23,615	18,11,570
	Pakistan	5	22,08,92,340	7,70,880
	Brazil	6	21,25,59,417	83,58,140
	Nigeria	7	20,61,39,589	9,10,770
	Bangladesh	8	16,46,89,383	1,30,170
	Russia	9	14,59,34,462	1,63,76,870
	Mexico	10	12,89,32,753	19,43,950

C#

```
// Get sheet
var worksheet = fpSpread1_Sheet1.AsWorksheet();

// Set value in cells
worksheet.Cells[1, 0].Value = "Flag";
worksheet.Cells[0, 0].Value = "Top 10 Populated Countries";
fpSpread1_Sheet1.AddSpanCell(0, 0, 1, 5);

// Set data
worksheet.SetValue(1, 1, new object[,]
{
    {"Country", "Rank", "Population (2020)", "Land Area (km sq)"},
    {"China", 1, 1439323776, 9388211 },
    {"India", 2, 1380004385, 2973190},
    {"United States", 3, 331002651, 9147420},
    {"Indonesia", 4, 273523615, 1811570},
    {"Pakistan", 5, 220892340, 770880},
    {"Brazil", 6, 212559417, 8358140},
    {"Nigeria", 7, 206139589, 910770},
    {"Bangladesh", 8, 164689383, 130170},
    {"Russia", 9, 145934462, 16376870},
    {"Mexico", 10, 128932753, 1943950}
});

// Set Image function formula in cells
worksheet.Cells[2, 0].Formula = "GC.IMAGE(\"https://www.wallpaperflare.com/static/193/1001/133/five-starred-red-flag-china-flag-five-wallpaper.jpg\")";
worksheet.Cells[3, 0].Formula = "GC.IMAGE(\"https://www.pngfind.com/pngs/m/21-211631_the-indian-flag-png-indian-flag-icon-transparent.png\")";
worksheet.Cells[4, 0].Formula = "GC.IMAGE(\"https://previews.123rf.com/images/auttkhamkhauncham/auttkhamkhauncham1507/auttkhamkhauncham150700090/42304741-usa-flag.jpg\")";
worksheet.Cells[5, 0].Formula = "GC.IMAGE(\"https://i.pinimg.com/236x/e8/0a/d5/e80ad5d4ea09700a78719ca051d19cbd.jpg\")";
worksheet.Cells[6, 0].Formula = "GC.IMAGE(\"https://static.vecteezy.com/system/resources/previews/000/114/048/non_2x/free-vector-pakistan-flag.jpg\")";
worksheet.Cells[7, 0].Formula = "GC.IMAGE(\"https://upload.wikimedia.org/wikipedia/en/thumb/0/05/Flag_of_Brazil.svg/2560px-Flag_of_Brazil.svg.png\")";
worksheet.Cells[8, 0].Formula = "GC.IMAGE(\"https://i.pinimg.com/originals/73/22/94/732294310c7e9fa3da611030168923fb.jpg\")";
worksheet.Cells[9, 0].Formula = "GC.IMAGE(\"https://images-na.ssl-images-amazon.com/images/I/31V23jzzMgL._AC_.jpg\")";
worksheet.Cells[10, 0].Formula = "GC.IMAGE(\"https://upload.wikimedia.org/wikipedia/en/thumb/f/f3/Flag_of_Russia.svg/1200px-Flag_of_Russia.svg.png\")";
worksheet.Cells[11, 0].Formula = "GC.IMAGE(\"https://www.pngkey.com/png/detail/442-4423822_mexico-icon-mexican-flag-icon-png.png\")";

// Set bgcolor for cells
worksheet.Cells["A1"].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(unchecked((int)0xFF9FD5B7));
worksheet.Cells["A2:E2"].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(unchecked((int)0xFFd9eee2));
worksheet.Cells["A3:E12"].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(unchecked((int)0xFFf5fbf8));
```

Visual Basic

```
'Get sheet
Dim worksheet = FpSpread1_Sheet1.AsWorksheet()

'Set value in cells
worksheet.Cells(1, 0).Value = "Flag"
worksheet.Cells(0, 0).Value = "Top 10 Populated Countries"
FpSpread1_Sheet1.AddSpanCell(0, 0, 1, 5)

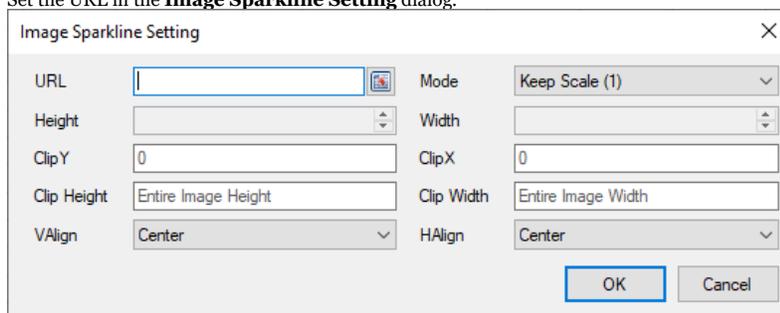
'Set data
worksheet.SetValue(1, 1, New Object(,) {
    {"Country", "Rank", "Population (2020)", "Land Area (km sq)"},
    {"China", 1, 1439323776, 9388211},
    {"India", 2, 1380004385, 2973190},
    {"United States", 3, 331002651, 9147420},
    {"Indonesia", 4, 273523615, 1811570},
    {"Pakistan", 5, 220892340, 770880},
    {"Brazil", 6, 212559417, 8358140},
    {"Nigeria", 7, 206139589, 910770},
    {"Bangladesh", 8, 164689383, 130170},
    {"Russia", 9, 145934462, 16376870},
    {"Mexico", 10, 128932753, 1943950}
})

'Set Image function formula in cells
worksheet.Cells(2, 0).Formula = "GC.IMAGE(\"\"https://www.wallpaperflare.com/static/193/1001/133/five-starred-red-flag-china-flag-five-wallpaper.jpg\"")"
worksheet.Cells(3, 0).Formula = "GC.IMAGE(\"\"https://www.pngfind.com/pngs/m/21-211631_the-indian-flag-png-indian-flag-icon-transparent.png\"")"
worksheet.Cells(4, 0).Formula =
"GC.IMAGE(\"\"https://previews.123rf.com/images/auttkhamkhauncham/auttkhamkhauncham1507/auttkhamkhauncham150700090/42304741-usa-flag.jpg\"")"
worksheet.Cells(5, 0).Formula = "GC.IMAGE(\"\"https://i.pinimg.com/236x/e8/0a/d5/e80ad5d4ea09700a78719ca051d19cbd.jpg\"")"
worksheet.Cells(6, 0).Formula = "GC.IMAGE(\"\"https://static.vecteezy.com/system/resources/previews/000/114/048/non_2x/free-vector-pakistan-flag.jpg\"")"
worksheet.Cells(7, 0).Formula =
"GC.IMAGE(\"\"https://upload.wikimedia.org/wikipedia/en/thumb/0/05/Flag_of_Brazil.svg/2560px-Flag_of_Brazil.svg.png\"")"
worksheet.Cells(8, 0).Formula =
"GC.IMAGE(\"\"https://i.pinimg.com/originals/73/22/94/732294310c7e9fa3da611030168923fb.jpg\"")"
worksheet.Cells(9, 0).Formula = "GC.IMAGE(\"\"https://images-na.ssl-images-amazon.com/images/I/31V23jzzMgL._AC_.jpg\"")"
worksheet.Cells(10, 0).Formula =
"GC.IMAGE(\"\"https://upload.wikimedia.org/wikipedia/en/thumb/f/f3/Flag_of_Russia.svg/1200px-Flag_of_Russia.svg.png\"")"
worksheet.Cells(11, 0).Formula = "GC.IMAGE(\"\"https://www.pngkey.com/png/detail/442-4423822_mexico-icon-mexican-flag-icon-png.png\"")"

'Set bgcolor for cells
worksheet.Cells("A1").Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFF9FD5B7)
worksheet.Cells("A2:E2").Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFFD9EEE2)
worksheet.Cells("A3:E12").Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFFF5FBF8)
```

Using the Spread Designer

1. Select a cell for the sparkline.
2. Select the Insert menu.
3. Select a sparkline type.
4. Set the URL in the **Image Sparkline Setting** dialog.



Set the additional sparkline settings as shown in the image above.

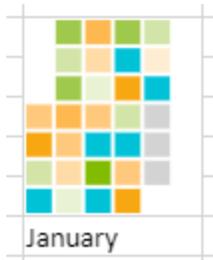
5. Select OK.
6. Select Apply and Exit from the File menu to save your changes and close the designer.

Note: Spread for WinForms also provides a function that allows you to insert an image in a cell. For more information, refer to the [IMAGE](#) function.

Month and Year Sparkline

Month and year sparklines are used for spotting monthly and annual trends in a given set of data. You can showcase the difference in data sets using a color range.

A month sparkline has 6x7 square spaces where a week of the month is displayed in horizontal direction from left to right and the days of the week (from Sunday to Saturday) are displayed in vertical direction from top to bottom. There are white separator lines among days.



Month Sparkline

A year sparkline has 54x7 square spaces where the weeks of year is displayed in horizontal direction from left to right and The days of the week (from Sunday to Saturday) are displayed in vertical direction from top to bottom.

There are white separator lines among days and black separator lines among months.



Year Sparkline

The month sparkline formula has the following formats:

```
=MONTHSPARKLINE(year, month, dataRange, emptyColor, startColor, middleColor, endColor)
or
=MONTHSPARKLINE(year, month, dataRange, colorRange)
```

The year sparkline formula has the following formats:

```
=YEARSPARKLINE(year, dataRange, emptyColor, startColor, middleColor, endColor)
or
=YEARSPARKLINE(year, dataRange, colorRange)
```

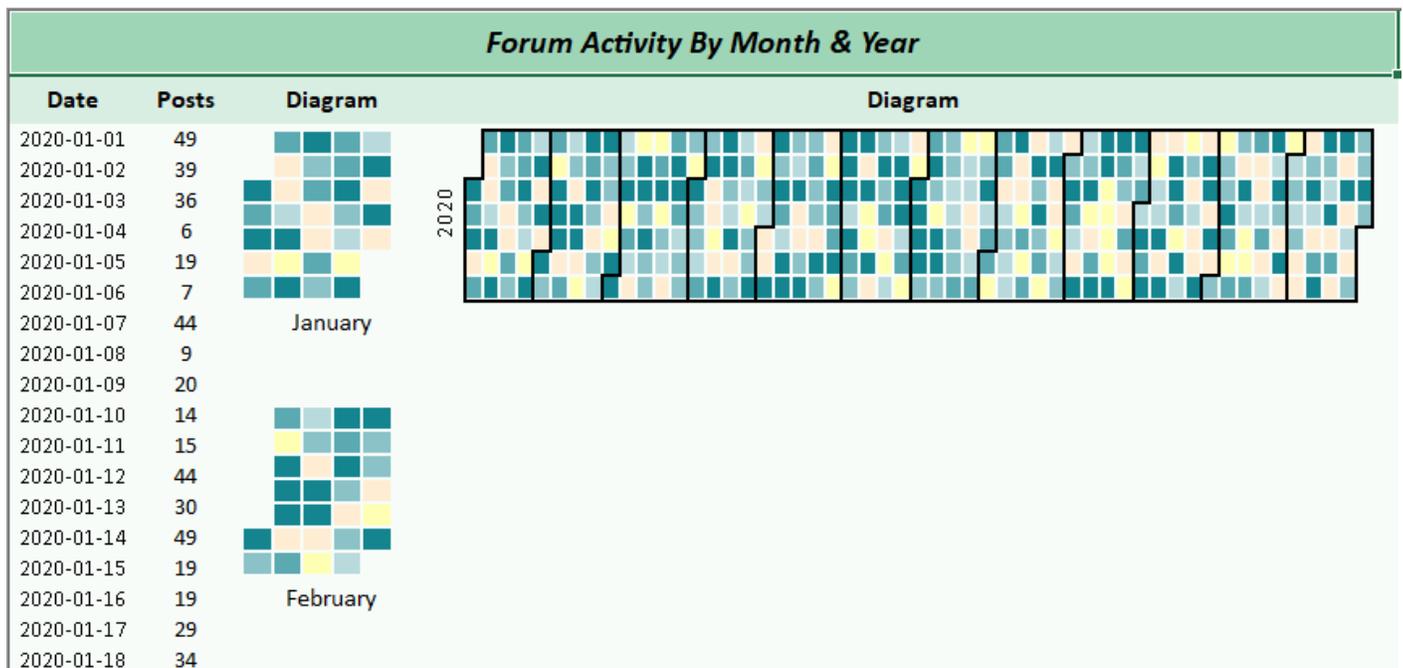
The formula options are described below:

Option	Description
year	A full year number, such as 2017.
month	A month number, such as 3. The month is 1-based (Jan = 1). This option is specific to MONTHSPARKLINE
dataRange	A reference that represents a range where the first column is a date and the second column is a number, such as "A1:B400".
emptyColor	A color string that represents days with no value or zero value, such as "lightgray".

- startColor** A color string that represents the minimum day value, such as "lightgreen".
- middleColor** A color string that represents the day with the average minimum and maximum value, such as "green".
- endColor** A color string that represents the day with the maximum value, such as "darkgreen".
- colorRange** A reference that represents a range where the data is a color string.

Usage Scenario

Consider a scenario where a company receives several query posts in its support forum all around the year. A month and year sparkline are convenient to observe the forum section activity. They can help identify the days with the most and least amount of site traffic in a specific month or a year.



C#

```
// Get sheet
var worksheet = fpSpread1_Sheet1.AsWorksheet();

// Set data
DateTime dates = new DateTime(2020, 1, 1);

// Initialize Random number
Random _rand = new Random();

for (var row = 3; row <= worksheet.RowCount; row++)
{
    worksheet.Cells["$A{row}"].Value = dates.AddDays(row - 3);
    worksheet.Cells["$B{row}"].Value = Math.Round(_rand.NextDouble() * 50);
    worksheet.Cells["$C{row}"].Value = colorList[_rand.Next(0, colorList.Length)];
}

// Set month sparkline formula
worksheet.Cells["D3"].Formula2 = "MONTHSPARKLINE(2020,1,A3:B33,C3:C33)";
worksheet.Cells["D9"].Formula2 = "TEXT(DATE(2020,1, 1),\"mmmm\")";
worksheet.Cells["D12"].Formula2 = "MONTHSPARKLINE(2020,2,A34:B62,C34:C62)";
```

```
worksheet.Cells["D18"].Formula2 = "TEXT(DATE(2020,2, 1),\"mmmm\")";

// Set year sparkline formula
worksheet.Cells["E3"].Formula2 = $"YearSparkline(2020, A3:B{worksheet.RowCount}, C3:C368)";
```

Visual Basic

```
'Get sheet
Dim worksheet = FpSpread1_Sheet1.AsWorksheet()

'Initialize random number
Dim _rand As New Random()

'Set data
Dim dates As New DateTime(2020, 1, 1)

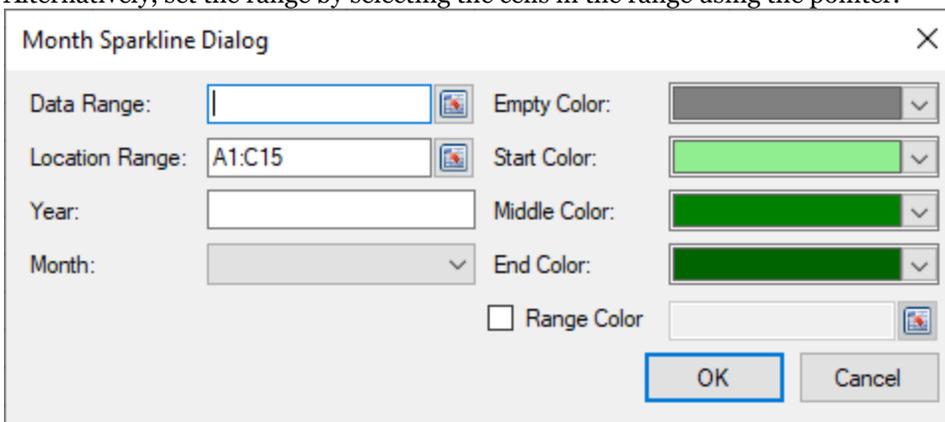
For row = 3 To worksheet.RowCount
    worksheet.Cells($"A{row}").Value = dates.AddDays(row - 3)
    worksheet.Cells($"B{row}").Value = Math.Round(_rand.NextDouble() * 50)
    worksheet.Cells($"C{row}").Value = colorList[_rand.[Next](0, colorList.Length)]
Next

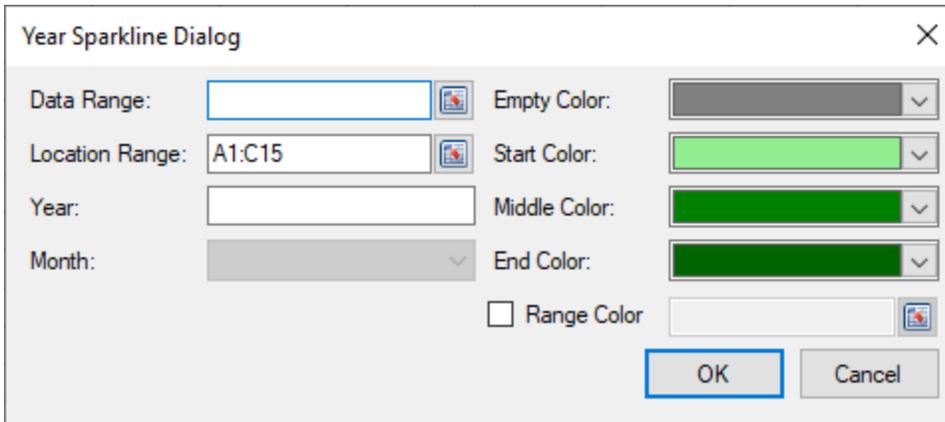
'Set month sparkline formula
worksheet.Cells("D3").Formula2 = "MONTHSPARKLINE(2020,1,A3:B33,C3:C33)"
worksheet.Cells("D9").Formula2 = "TEXT(DATE(2020,1, 1),\"mmmm\")"
worksheet.Cells("D12").Formula2 = "MONTHSPARKLINE(2020,2,A34:B62,C34:C62)"
worksheet.Cells("D18").Formula2 = "TEXT(DATE(2020,2, 1),\"mmmm\")"

'Set year sparkline formula
worksheet.Cells("E3").Formula2 = $"YearSparkline(2020, A3:B{worksheet.RowCount}, C3:C368)";
```

Using the Spread Designer

1. Type the Data Range in cells where the first column is a date and the second column is a number.
2. Select a cell for the sparkline.
3. Select the Insert menu.
4. Select a sparkline type.
5. Set the Data Range in the **Sparkline Dialog** (such as =Sheet1!\$E\$1:\$E\$3).
Alternatively, set the range by selecting the cells in the range using the pointer.



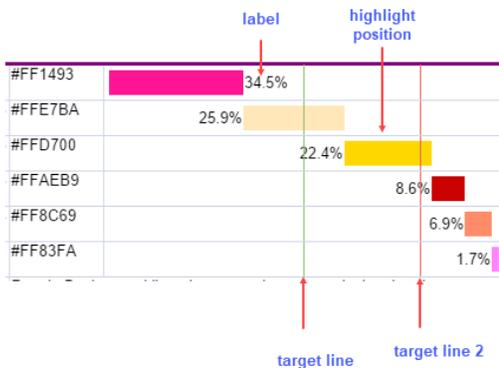


Set the additional sparkline settings as shown in the images above.

6. Select OK.
7. Select Apply and Exit from the File menu to save your changes and close the designer.

Pareto Sparkline

A pareto sparkline can be used to highlight the most important items in a set of values. This sparkline usually is taken as a quality tool since it helps analyze and prioritize issue resolution.



The pareto sparkline formula have the following format:

`=PARETOSPARKLINE(points, [pointIndex, colorRange, target, target2, highlightPosition, label, vertical, targetColor, target2Color, labelColor, barSize])`

The formula options are described:

Option	Description
points	A reference that represents the range of cells that contains all values, such as "B2:B7".
pointIndex	A number or reference that represents the segment's index of the points, such as 1 or "D2".
Optional	The pointIndex is >= 1.
colorRange	A reference that represents the range of cells that contain the color for the segment box, such as "D2:D7".
Optional	The default value is none.
target	A number or reference that represents the 'target' line position, such as 0.5.
Optional	The default value is none. The target line color is #8CBF64 if shown.
target2	A number or reference that represents the 'target2' line position, such as 0.8.
Optional	The default value is none. The target2 line color is #EE5D5D if shown.
highlightPosition	A number or reference that represents the rank of the segment to be colored in red, such as 3.
Optional	The default value is none. If you set the highlightPosition to a value such as 4, then the fourth segment box's color is set to #CBO000. If you do not set the highlightPosition, the segment box's color is set to the color you assigned to the colorRange or the default color #969696.
label	A number that represents whether the segment's label is displayed as the cumulated percentage (label = 1) or the single percentage or none (label = 2) or none, such as 2,1.
Optional	The default value is 0.
vertical	A boolean that represents whether the box's direction is vertical or horizontal.
Optional	The default value is False.
targetColor	A color string that indicates the color of the target line.

Optional

target2Color A color string that indicates the color of the target2 line.

Optional

labelColor A color string that indicates the label fore color.

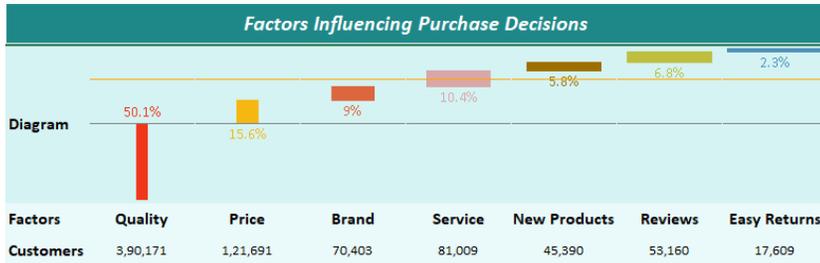
Optional

barSize A number value greater than 0 and less than or equal to 1, which indicates the percentage of bar width or height according to the cell width or height.

Optional

Usage Scenario

Consider a scenario where a survey is conducted on an e-commerce site to determine how a customer decides to purchase products from the site. A pareto sparkline helps highlight the most decisive factors and analyze how to benefit from the result.



C#

```
// Get sheet
var worksheet = fpSpread1_Sheet1.AsWorksheet();

// Set data for sparkline
worksheet.SetValue(2, 0, new object[]
{
    {"Factors","Quality","Price","Brand","Service","New Products","Reviews","Easy Returns"},
    {"Customers",390171,121691,70403,81009,45390,53160,17609},
    {"Color","#F0371A","#F4B811","#DE663E","#D9A7A7","#9E6F00","#BFBF3F","#4C90BA"},
    {"BarSize",0.1,0.2,0.4,0.6,0.7,0.8,0.9}
});
worksheet.Cells["A2"].Text = "Diagram";

// Set formula
worksheet.Cells["B2"].Formula2 = "PARETOSPARKLINE(B4:H4,,B5:H5,0.5,0.8,0,2,TRUE,\"Gray\", \"Orange\",B5:H5,B6:H6)";
```

Visual Basic

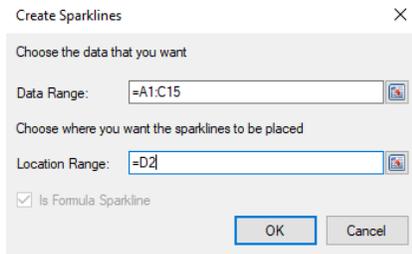
```
'Get sheet
Dim worksheet = FpSpread1_Sheet1.AsWorksheet()

'Set data for sparkline
worksheet.SetValue(2, 0, New Object() {
    {"Factors", "Quality", "Price", "Brand", "Service", "New Products", "Reviews", "Easy Returns"},
    {"Customers", 390171, 121691, 70403, 81009, 45390, 53160, 17609},
    {"Color", "#F0371A", "#F4B811", "#DE663E", "#D9A7A7", "#9E6F00", "#BFBF3F", "#4C90BA"},
    {"BarSize", 0.1, 0.2, 0.4, 0.6, 0.7, 0.8, 0.9}
})
worksheet.Cells("A2").Text = "Diagram"

'Set formula
worksheet.Cells("B2").Formula2 = "PARETOSPARKLINE(B4:H4,,B5:H5,0.5,0.8,0,2,TRUE,\"Gray\", \"Orange\",B5:H5,B6:H6)";
```

Using the Spread Designer

1. Type data in a cell or a column or row of cells in the designer.
2. Select a cell for the sparkline.
3. Select the Insert menu.
4. Select a sparkline type.
5. Set the Data Range in the **Create Sparklines** dialog (such as =Sheet1!\$E\$1:\$E\$3). Alternatively, set the range by selecting the cells in the range using the pointer.



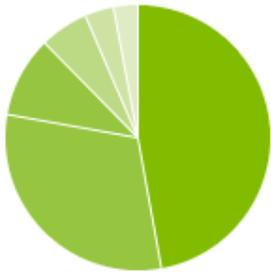
You can also set additional sparkline settings in the dialog if available.

6. Select OK.

7. Select Apply and Exit from the File menu to save your changes and close the designer.

Pie Sparkline

Pie sparkline is a circular statistical diagram, which is divided into slices to illustrate numerical proportion. This sparkline makes it easier to see the percentages of data points compared to the entire set of data.

Amount	Diagram	Note
\$120,000		47.15%
\$78,000		30.65%
\$25,000		9.82%
\$15,000		5.89%
\$9,000		3.54%
\$7,500		2.95%

The pie sparkline formula has the following syntax:

```
=PIESPARKLINE(percentage, [color1, color2, ...])
```

The formula options are described below where only 'percentage' is the required argument:

Option	Description
percentage	A value that represents the range of data sources.
color1, color2, ...	Strings that represent the sequence of colors for the pie slices.
<i>Optional</i>	

Usage Scenario

Consider a scenario where a company analyzes its energy consumption in a month. It wants to display the key areas where it can improve on energy conservation. A pie sparkline can showcase the proportions of different consumption factors in the office campus.

Energy Bill Breakdown			
Resource	Energy Consumed (KW)	Diagram	Percent
Heating and Cooling	28,910		36.45%
Water Heating	13,800		17.4%
Lighting	11,236		14.17%
Variable	8,920		11.25%
Electronics	7,656		9.65%
Washer-Dryer	8,800		11.09%

C#

```
// Get sheet
var worksheet = fpSpread1_Sheet1.AsWorksheet();

// Set data for sparkline
worksheet.SetValue(1, 0, new object[,]
{
    { "Resource", "Energy Consumed (KW)", "Diagram", "Percent" },
```

```

    { "Heating and Cooling", 28910, null, null },
    { "Water Heating", 13800, null, null },
    { "Lighting", 11236, null, null },
    { "Variable", 8920, null, null },
    { "Electronics", 7656, null, null },
    { "Washer-Dryer", 8800, null, null }
});

// Merge cells of column C
worksheet.Cells["C3:C8"].Merge();
// Apply formula for PieSparkline
worksheet.Cells["C3"].Formula =
"PIESPARKLINE(B3:B8,\"#76a3b2\",\"#8db2bf\",\"#a4c1cb\",\"#bbd1d8\",\"#d1e0e5\",\"#e8f0f2\")";

```

Visual Basic

```

'Get sheet
Dim worksheet = FpSpread1_Sheet1.AsWorksheet()

'Set data for sparkline
worksheet.SetValue(1, 0, New Object() {
{"Resource", "Energy Consumed (KW)", "Diagram", "Percent"},
{"Heating and Cooling", 28910, Nothing, Nothing},
{"Water Heating", 13800, Nothing, Nothing},
{"Lighting", 11236, Nothing, Nothing},
{"Variable", 8920, Nothing, Nothing},
{"Electronics", 7656, Nothing, Nothing},
{"Washer-Dryer", 8800, Nothing, Nothing}})

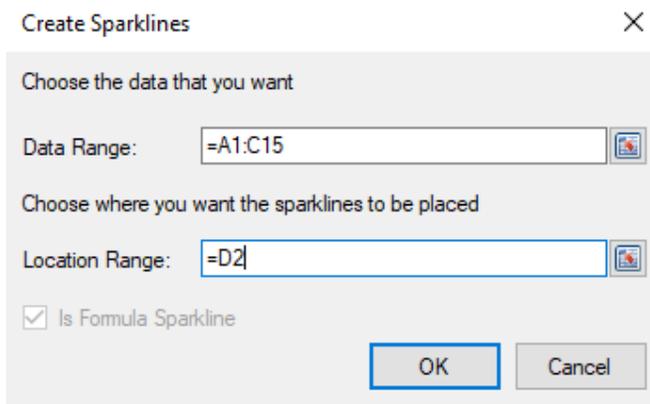
'Merge cells of column C
worksheet.Cells("C3:C8").Merge()

'Apply formula for PieSparkline
worksheet.Cells("C3").Formula =
"PIESPARKLINE(B3:B8, ""#76a3b2"", ""#8db2bf"", ""#a4c1cb"", ""#bbd1d8"", ""#d1e0e5"", ""#e8f0f2"")"

```

Using the Spread Designer

1. Type data in a cell or a column or row of cells in the designer.
2. Select a cell for the sparkline.
3. Select the Insert menu.
4. Select a sparkline type.
5. Set the Data Range in the **Create Sparklines** dialog (such as =Sheet1!\$E\$1:\$E\$3). Alternatively, set the range by selecting the cells in the range using the pointer.

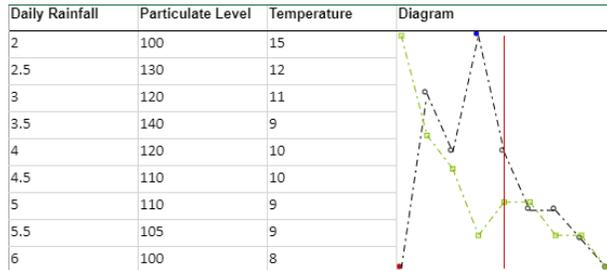


You can also set additional sparkline settings in the dialog if available.

6. Select OK.
7. Select Apply and Exit from the File menu to save your changes and close the designer.

Scatter Sparkline

Scatter sparklines plots numerical data, with one variable on each axis, to visualize correlations between them. It can also be used to compare numeric values, such as scientific, statistical, and engineering data as shown in the image below.



The scatter sparkline formula has the following syntax:

```
=SCATTERSPARKLINE([points1], [points2, minX, maxX, minY, maxY, hLine, vLine, xMinZone, xMaxZone, yMinZone, yMaxZone, tags, drawSymbol, drawLines, color1, color2, dash])
```

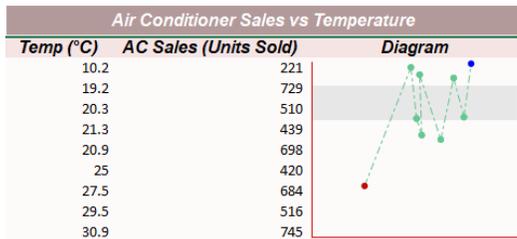
The formula options are described below:

Option	Description
points1	The first xy data series. If the row count is greater than or equal to the column count, use data from the first two columns. The first column contains x-values and the second column contains y-values. If the row count is less than the column count, use data from the first two rows. The first row contains x-values and the second row contains y-values.
points2	The second xy data series. If the row count is greater than or equal to the column count, use data from the first two columns. The first column contains x-values and the second column contains y-values. If the row count is less than the column count, use data from the first two rows. The first row contains x-values and the second row contains y-values.
Optional	
minX	The x-minimum limit of both series, each series has its own value if it is omitted.
Optional	
maxX	The x-maximum limit of both series, each series has its own value if it is omitted.
Optional	
minY	The y-minimum limit of both series, each series has its own value if it is omitted.
Optional	
maxY	The y-maximum limit of both series, each series has its own value if it is omitted.
Optional	
hLine	The horizontal axis position, there is no line if it is omitted.
Optional	
vLine	The vertical axis position, there is no line if it is omitted.
Optional	
xMinZone	The x-minimum value of the gray zone, there is no grey zone if any of these four parameters are omitted.
Optional	
xMaxZone	The x-maximum value of the gray zone, there is no grey zone if any of these four parameters are omitted.
Optional	
yMinZone	The y-minimum value of the gray zone, there is no grey zone if any of these four parameters are omitted.
Optional	
yMaxZone	The y-maximum value of the gray zone, there is no grey zone if any of these four parameters are omitted.
Optional	
tags	If this option is true, mark the point at which the y-value is the maximum of the first series as "#0000FF", and mark the point at which the y-value is the minimum of the first series as "#CB0000". This option is false if it is omitted.
Optional	
drawSymbol	If this option is true, draw each point as a symbol. The symbol of the first series is a circle, and the symbol of the second series is a square.
Optional	This option is true if it is omitted.
drawLines	If this option is true, connect each point with a line by sequence in each series.
Optional	This option is false if it is omitted.
color1	The color string of the first point series.
Optional	The value is "#969696" if it is omitted.
color2	The color string of the second point series.
Optional	The value is "#CB0000" if it is omitted.
dash	If this option is true, the line is a dashed line; otherwise, the line is a full line.
Optional	This option is false if it is omitted.

Usage Scenario

Consider a scenario where an electronic megastore wants to analyze the relationship between the sales of air conditioning units and the average temperature in a given week. A scatter sparkline can showcase the correlation

between both of these sets of values.



C#

```
// Get sheet
var worksheet = fpSpread1_Sheet1.AsWorksheet();

// Set data for sparkline
worksheet.SetValue(1, 0, new object[,]
{
    { "Temp (°C)", "AC Sales (Units Sold)", "Diagram" },
    { 10.2, 221, "null" },
    { 19.2, 729, "null" },
    { 20.3, 510, "null" },
    { 21.3, 439, "null" },
    { 20.9, 698, "null" },
    { 25, 420, "null" },
    { 27.5, 684, "null" },
    { 29.5, 516, "null" },
    { 30.9, 745, "null" }
});

// Merge cells of column C
worksheet.Cells["C3:C11"].Merge();
// Apply formula for ScatterSparkline
worksheet.Cells[2, 2].Formula = "SCATTERSPARKLINE(A3:B11,,0,40,0,750,0,0,0,40,500,650,TRUE,TRUE,TRUE,\"#63C890\",,TRUE)";
```

Visual Basic

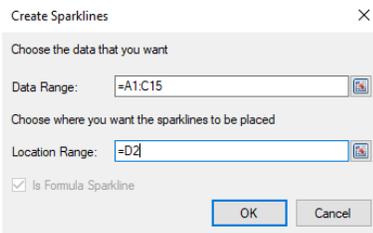
```
'Get sheet
Dim worksheet = FpSpread1_Sheet1.AsWorksheet()

'Set data for sparkline
worksheet.SetValue(1, 0, New Object() {
    {"Temp (°C)", "AC Sales (Units Sold)", "Diagram"},
    {10.2, 221, "null"},
    {19.2, 729, "null"},
    {20.3, 510, "null"},
    {21.3, 439, "null"},
    {20.9, 698, "null"},
    {25, 420, "null"},
    {27.5, 684, "null"},
    {29.5, 516, "null"},
    {30.9, 745, "null"}
})

'Merge cells of column C
worksheet.Cells("C3:C11").Merge()
'Apply formula for ScatterSparkline
worksheet.Cells(2, 2).Formula = "SCATTERSPARKLINE(A3:B11,,0,40,0,750,0,0,0,40,500,650,TRUE,TRUE,TRUE,\"#63C890\",,TRUE)";
```

Using the Spread Designer

1. Type data in a cell or a column or row of cells in the designer.
2. Select a cell for the sparkline.
3. Select the Insert menu.
4. Select a sparkline type.
5. Set the Data Range in the **Create Sparklines** dialog (such as =Sheet1!\$E\$1:\$E\$3). Alternatively, set the range by selecting the cells in the range using the pointer.

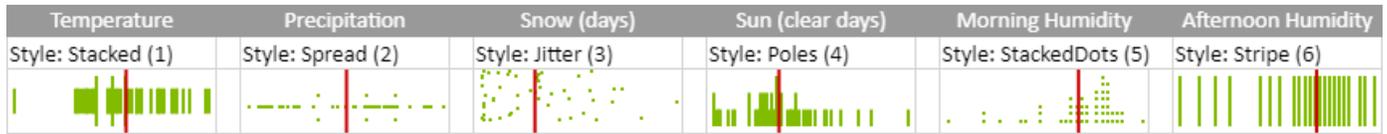


You can also set additional sparkline settings in the dialog if available.

6. Select OK.
7. Select Apply and Exit from the File menu to save your changes and close the designer.

Spread Sparkline

Spread sparklines can be used to distribute data and compare them using different styles as shown in the image below.



When the sparkline is horizontal, the horizontal axis represents each point. The length of lines or the number of dots in the vertical axis represents the frequency of occurrence.

The spread sparkline formula has the following syntax:

```
=SPREADSPARKLINE(points, [showAverage, scaleStart, scaleEnd, style, colorScheme, vertical])
```

The formula options are described below where only 'points' is the default parameter:

Option	Description
points	A reference that represents a range of cells that contains the values, such as "A1:A10".
showAverage	A boolean that represents whether to show the average.
<i>Optional</i>	The default value is false.
scaleStart	A number that represents the minimum boundary of the sparkline.
<i>Optional</i>	The default value is the minimum of all values.
scaleEnd	A number that represents the maximum boundary of the sparkline.
<i>Optional</i>	The default value is the maximum of all values.
style	A number that references the style of the Spread sparkline.
<i>Optional</i>	The following six styles are supported: <ul style="list-style-type: none"> • Stacked = 1 - line from center to two sides • Spread = 2 - dot from center to two sides • Jitter = 3 - dot with a random location • Poles = 4 - line from one side to another • StackedDots = 5 - dot from one side to another • Stripe = 6 - line with an equal length The default value is 4 (poles).
colorScheme	A string that represents the color of the sparkline.
<i>Optional</i>	The default value is "#646464".
vertical	A boolean that represents whether to display the sparkline vertically.
<i>Optional</i>	The default value is false.

Usage Scenario

Consider a scenario where the disaster management department of a country wants to share the cost distribution and the fatalities suffered in different disasters that occurred through the years. A spread sparkline can help to display the various statistics involved with calamities, as shown in the image below.

Distribution of costs and fatalities by disaster type						
Disaster type	No. of Events	Percent Frequency	Total Costs (Bil.)	Percent of Total Costs	Cost/Event (Bil.)	Deaths/Year
Drought	28	9.8%	\$258.9	15.8%	\$9.2	95
Flooding	33	11.6%	\$151.1	8.1%	\$4.6	15
Freeze	9	3.2%	\$31.1	1.6%	\$3.4	4
Severe Storm	128	44.9%	\$286.3	15.3%	\$2.2	43
Tropical Cyclone	52	18.2%	\$997.3	53.1%	\$19.2	161
Wildfire	18	6.3%	\$102.3	5.5%	\$5.7	10
Winter Storm	17	6.0%	\$50.1	2.7%	\$2.9	26

C#

```
// Get sheet
var worksheet = fpSpread1_Sheet1.AsWorksheet();

// Set data
worksheet.SetValue(2, 0, new object[,]
{
    { "Disaster type","No. of Events","Percent Frequency","Total Costs (Bil.)","Percent
of Total Costs","Cost/Event (Bil.)","Deaths/Year" },
    {"Drought",28, 0.098, 258.9,0.138,9.2, 95 },
    {"Flooding",33, 0.116,151.09,0.081,4.6, 15},
    {"Freeze",9,0.032,31.07,0.016,3.4,4},
    {"Severe Storm",128,0.449, 286.3,0.153,2.2, 43},
    {"Tropical Cyclone",52, 0.182,997.3,0.531,19.2,161},
    {"Wildfire",18,0.063,102.3,0.055,5.7,10},
    {"Winter Storm",17,0.060,50.1,0.027,2.9, 26}
});

// Set SpreadSparkline formula
worksheet.Cells["B2"].Formula = "SPREADSPARKLINE (B4:B10,TRUE,,,1,\"#00cccc\")";
worksheet.Cells["C2"].Formula = "SPREADSPARKLINE (C4:C10,TRUE,,,2,\"#00cccc\")";
worksheet.Cells["D2"].Formula = "SPREADSPARKLINE (D4:D10,TRUE,,,3,\"#00cccc\")";
worksheet.Cells["E2"].Formula = "SPREADSPARKLINE (E4:E10,TRUE,,,4,\"#00cccc\")";
worksheet.Cells["F2"].Formula = "SPREADSPARKLINE (F4:F10,TRUE,,,5,\"#00cccc\")";
worksheet.Cells["G2"].Formula = "SPREADSPARKLINE (G4:G10,TRUE,,,6,\"#00cccc\")";
```

Visual Basic

```
'Get sheet
Dim worksheet = FpSpread1.Sheets(0).AsWorksheet()

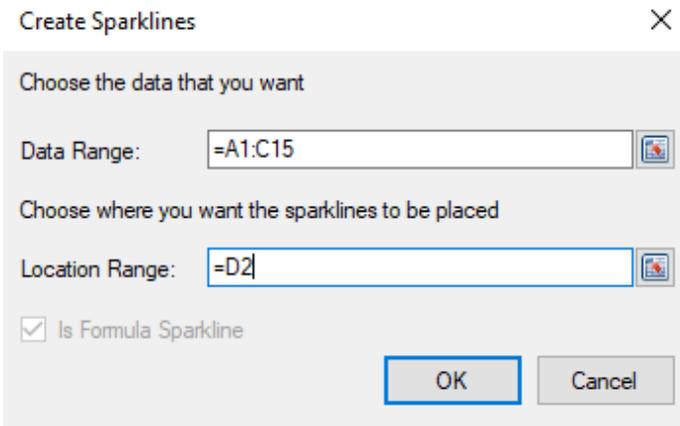
'Set data
worksheet.SetValue(2, 0, New Object(,) {
    {"Disaster type", "No. of Events", "Percent Frequency", "Total Costs (Bil.)",
"Percent of Total Costs", "Cost/Event (Bil.)", "Deaths/Year"},
    {"Drought", 28, 0.098, 258.9, 0.138, 9.2, 95},
    {"Flooding", 33, 0.116, 151.09, 0.081, 4.6, 15},
    {"Freeze", 9, 0.032, 31.07, 0.016, 3.4, 4},
    {"Severe Storm", 128, 0.449, 286.3, 0.153, 2.2, 43},
```

```
{ "Tropical Cyclone", 52, 0.182, 997.3, 0.531, 19.2, 161 },
{ "Wildfire", 18, 0.063, 102.3, 0.055, 5.7, 10 },
{ "Winter Storm", 17, 0.06, 50.1, 0.027, 2.9, 26 }
})
```

```
'Set SpreadSparkline formula
worksheet.Cells("B2").Formula = "SPREADSPARKLINE(B4:B10,TRUE,,,1,""#00cccc")"
worksheet.Cells("C2").Formula = "SPREADSPARKLINE(C4:C10,TRUE,,,2,""#00cccc")"
worksheet.Cells("D2").Formula = "SPREADSPARKLINE(D4:D10,TRUE,,,3,""#00cccc")"
worksheet.Cells("E2").Formula = "SPREADSPARKLINE(E4:E10,TRUE,,,4,""#00cccc")"
worksheet.Cells("F2").Formula = "SPREADSPARKLINE(F4:F10,TRUE,,,5,""#00cccc")"
worksheet.Cells("G2").Formula = "SPREADSPARKLINE(G4:G10,TRUE,,,6,""#00cccc")"
```

Using the Spread Designer

1. Type data in a cell or a column or row of cells in the designer.
2. Select a cell for the sparkline.
3. Select the Insert menu.
4. Select a sparkline type.
5. Set the Data Range in the **Create Sparklines** dialog (such as =Sheet1!\$E\$1:\$E\$3). Alternatively, set the range by selecting the cells in the range using the pointer.



You can also set additional sparkline settings in the dialog if available.

6. Select OK.
7. Select Apply and Exit from the File menu to save your changes and close the designer.

Stacked Sparkline

A stacked sparkline is used to show a sliced breakdown of a value in different categories. It is useful for comparing values across categories as shown in the image below.

Q1	Q2	Q3	Q4	
\$162 K	\$75 K	\$162 K	\$75 K	
\$122 K	\$30 K	\$122 K	\$30 K	
\$156 K	\$75 K	\$216 K	\$73 K	
\$71 K	\$50 K	\$271 K	\$50 K	
\$71 K	\$50 K	\$71 K	\$99 K	
\$158 K	\$123 K	\$196 K	\$81 K	

The stacked sparkline formula has the following syntax:

```
=STACKEDSPARKLINE(points, [colorRange, labelRange, maximum, targetRed, targetGreen, targetBlue, targetYellow, color, highlightPosition, vertical, textOrientation, textSize, minimum])
```

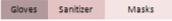
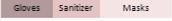
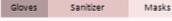
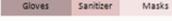
The formula options are described below:

Option	Description
points	A reference that represents a range of cells that contains the values, such as "A1:A4".
colorRange	A reference that represents a range of cells that contains all colors, such as "B1:B4".

<i>Optional</i>	The default value is generated by color.
labelRange	A reference that represents a cell range that contains all labels, such as "C1:C4".
<i>Optional</i>	The default value is empty.
maximum	A number that represents the maximum value of the sparkline.
<i>Optional</i>	The default value is the summary of all positive values.
targetRed	A number that represents the location of the red line. This setting is optional.
<i>Optional</i>	The default value is empty.
targetGreen	A number that represents the location of the green line.
<i>Optional</i>	The default value is empty.
targetBlue	A number that represents the location of the blue line.
<i>Optional</i>	The default value is empty.
targetYellow	A number that represents the location of the yellow line.
<i>Optional</i>	The default value is empty.
color	A string that represents the color if colorRange is omitted.
<i>Optional</i>	The default value is "#646464".
highlightPosition	A number that represents the index of the highlight area.
<i>Optional</i>	The default value is empty.
vertical	A boolean that represents whether to display the sparkline vertically.
<i>Optional</i>	The default value is false.
textOrientation	A number that represents the label text orientation.
<i>Optional</i>	The default value is 0 (horizontal). The vertical setting is 1.
textSize	A number that represents the size of the label text in pixels.
<i>Optional</i>	The default value is 10.
minimum	A number that represents the minimum axis value.
<i>Optional</i>	The default value is 0.

Usage Scenario

Consider a scenario where a company wants to display the sales of products in different US states. Stacked sparklines can show each US state divided into the shares of the company products.

Sales by State				
State	Gloves	Sanitizer	Masks	Diagram
Idaho	\$10,000	\$12,000	\$15,000	
Montana	\$11,000	\$10,000	\$15,000	
Oregon	\$10,000	\$17,000	\$12,000	
Washington	\$15,000	\$10,000	\$15,000	
Utah	\$10,000	\$15,000	\$12,000	

C#

```
// Get sheet
var worksheet = fpSpread1_Sheet1.AsWorksheet();

// Add data for sparkline
worksheet.SetValue(1, 0, "State");
worksheet.SetValue(1, 1, "Gloves");
worksheet.SetValue(1, 2, "Sanitizer");
worksheet.SetValue(1, 3, "Masks");
worksheet.SetValue(1, 4, "Diagram");
worksheet.SetValue(2, 0, "Idaho");
worksheet.SetValue(2, 1, 10000);
worksheet.SetValue(2, 2, 12000);
worksheet.SetValue(2, 3, 15000);
worksheet.SetValue(3, 0, "Montana");
worksheet.SetValue(3, 1, 11000);
worksheet.SetValue(3, 2, 10000);
worksheet.SetValue(3, 3, 15000);
worksheet.SetValue(4, 0, "Oregon");
worksheet.SetValue(4, 1, 10000);
worksheet.SetValue(4, 2, 17000);
worksheet.SetValue(4, 3, 12000);
worksheet.SetValue(5, 0, "Washington");
worksheet.SetValue(5, 1, 15000);
worksheet.SetValue(5, 2, 10000);
worksheet.SetValue(5, 3, 15000);
worksheet.SetValue(6, 0, "Utah");
worksheet.SetValue(6, 1, 10000);
worksheet.SetValue(6, 2, 15000);
worksheet.SetValue(6, 3, 12000);
worksheet.SetValue(7, 1, "#B39A9A");
worksheet.SetValue(7, 2, "#E3C3C3");
worksheet.SetValue(7, 3, "#F5E4E4");

// Set number format for columns
worksheet.Columns["B:D"].NumberFormat = "$#,##0";

// Set StackedSparkline formula
worksheet.Cells[2, 4].Formula = "STACKEDSPARKLINE(B3:D3,B8:D8,B2:D2,40000)";
worksheet.Cells[3, 4].Formula = "STACKEDSPARKLINE(B4:D4,B8:D8,B2:D2,40000)";
worksheet.Cells[4, 4].Formula = "STACKEDSPARKLINE(B5:D5,B8:D8,B2:D2,40000)";
worksheet.Cells[5, 4].Formula = "STACKEDSPARKLINE(B6:D6,B8:D8,B2:D2,40000)";
worksheet.Cells[6, 4].Formula = "STACKEDSPARKLINE(B7:D7,B8:D8,B2:D2,40000)";

// Set bgcolor for cells
worksheet.Cells["A1"].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(unchecked((int)0xFFDEACA7));
worksheet.Cells["A2:E2"].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(unchecked((int)0xFFFF5E4E4));
worksheet.Cells["A3:E7"].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(unchecked((int)0xFFFFdfafa));
```

Visual Basic

```
'Get sheet
Dim worksheet = FpSpread1_Sheet1.AsWorksheet()

'Add data for sparkline
worksheet.SetValue(1, 0, "State")
worksheet.SetValue(1, 1, "Gloves")
worksheet.SetValue(1, 2, "Sanitizer")
worksheet.SetValue(1, 3, "Masks")
worksheet.SetValue(1, 4, "Diagram")
worksheet.SetValue(2, 0, "Idaho")
worksheet.SetValue(2, 1, 10000)
worksheet.SetValue(2, 2, 12000)
worksheet.SetValue(2, 3, 15000)
worksheet.SetValue(3, 0, "Montana")
worksheet.SetValue(3, 1, 11000)
worksheet.SetValue(3, 2, 10000)
worksheet.SetValue(3, 3, 15000)
worksheet.SetValue(4, 0, "Oregon")
worksheet.SetValue(4, 1, 10000)
worksheet.SetValue(4, 2, 17000)
worksheet.SetValue(4, 3, 12000)
worksheet.SetValue(5, 0, "Washington")
worksheet.SetValue(5, 1, 15000)
worksheet.SetValue(5, 2, 10000)
worksheet.SetValue(5, 3, 15000)
worksheet.SetValue(6, 0, "Utah")
worksheet.SetValue(6, 1, 10000)
worksheet.SetValue(6, 2, 15000)
worksheet.SetValue(6, 3, 12000)
worksheet.SetValue(7, 1, "#B39A9A")
worksheet.SetValue(7, 2, "#E3C3C3")
worksheet.SetValue(7, 3, "#F5E4E4")

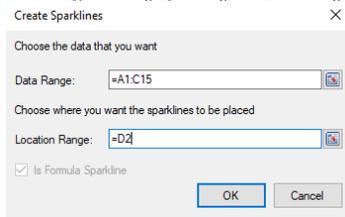
'Set number format for columns
worksheet.Columns("B:D").NumberFormat = "$#,##0"

'Set StackedSparkline formula
worksheet.Cells(2, 4).Formula = "STACKEDSPARKLINE(B3:D3,B8:D8,B2:D2,40000)"
worksheet.Cells(3, 4).Formula = "STACKEDSPARKLINE(B4:D4,B8:D8,B2:D2,40000)"
worksheet.Cells(4, 4).Formula = "STACKEDSPARKLINE(B5:D5,B8:D8,B2:D2,40000)"
worksheet.Cells(5, 4).Formula = "STACKEDSPARKLINE(B6:D6,B8:D8,B2:D2,40000)"
worksheet.Cells(6, 4).Formula = "STACKEDSPARKLINE(B7:D7,B8:D8,B2:D2,40000)"

'Set back color for cells
worksheet.Cells("A1").Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFFDEACA7)
worksheet.Cells("A2:E2").Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFFF5E4E4)
worksheet.Cells("A3:E7").Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFFFDFAF8)
```

Using the Spread Designer

1. Type data in a cell or a column or row of cells in the designer.
2. Select a cell for the sparkline.
3. Select the Insert menu.
4. Select a sparkline type.
5. Set the Data Range in the **Create Sparklines** dialog (such as =Sheet1!\$E\$1:\$E\$3). Alternatively, set the range by selecting the cells in the range using the pointer.

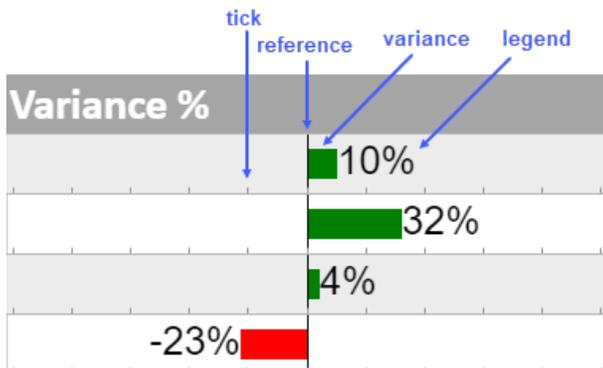


You can also set additional sparkline settings in the dialog if available.

6. Select OK.
7. Select Apply and Exit from the File menu to save your changes and close the designer.

Vari Sparkline

Vari sparklines are used to compare two sets of data and calculate the difference, or variance, between the values.



The vari sparkline formula has the following syntax:

```
=VARISPARKLINE(variance, [reference, mini, maxi, mark, tickunit, legend, colorPositive, colorNegative, vertical])
```

The formula options are described below:

Option	Description
variance	A number or reference that represents the bar length, such as 2 or "A1".
reference	A number or reference that represents the location of the reference line, such as 0 or "A2".
<i>Optional</i>	The default value is 0.
mini	A number or reference that represents the minimum value of the sparkline, such as -5 or "A3".
<i>Optional</i>	The default value is -1.
maxi	A number or reference that represents the maximum value of the sparkline, such as 5 or "A4".
<i>Optional</i>	The default value is 1.
mark	A number or reference that represents the position of the mark line, such as 3 or "A5".
<i>Optional</i>	
tickunit	A number or reference that represents a tick unit, such as 1 or "A6".
<i>Optional</i>	The default value is 0.
legend	A boolean that represents whether to display the text. The default is optional.
<i>Optional</i>	The default value is FALSE.
colorPositive	A string that represents the color scheme for variance and is larger than reference.
<i>Optional</i>	The default value is "green".
colorNegative	A string that represents the color scheme for variance and is smaller than reference.
<i>Optional</i>	The default value is "red".
vertical	A boolean that represents whether to display the sparkline vertically.
<i>Optional</i>	The default value is FALSE.

Usage Scenario

Consider a scenario where a company wants to display the revenue variation between two years, for example, in 2019 and 2020. A vari sparkline helps create a graphical representation of the data. It compares the revenue differences and presents the variance as shown in the image below.

Revenue Variance in past 2 years				
Sales	Year 2019	Year 2020	Variance	Variance %
Jan	65,431	74,930	9,499	15%
Feb	83,478	92,730	9,252	11%
Mar	90,021	12,301	-77,720	-86%
Apr	72,809	23,939	-48,870	-67%
May	1,03,832	34,719	-69,113	-67%
Jun	8,32,833	67,189	-7,65,644	-92%
Jul	6,71,801	73,289	-5,98,512	-89%
Aug	89,222	81,299	-7,923	-9%
Sep	68,919	91,200	22,281	32%
Oct	74,940	99,188	24,248	32%
Nov	81,991	1,06,181	24,190	30%
Dec	62,188	89,128	26,940	43%

C#

```
// Get sheet
var worksheet = fpSpread1_Sheet1.AsWorksheet();

// Set data
worksheet.SetValue(1, 0, new object[,]
{
    {"Sales", "Year 2019", "Year 2020", "Variance", "Variance %"},
    {"Jan", 65431, 74930, null, null},
    {"Feb", 83478, 92730, null, null},
    {"Mar", 90021, 12301, null, null},
    {"Apr", 72809, 23939, null, null},
    {"May", 103832, 34719, null, null},
    {"Jun", 832833, 67189, null, null},
    {"Jul", 671801, 73289, null, null},
    {"Aug", 89222, 81299, null, null},
    {"Sep", 68919, 91200, null, null},
    {"Oct", 74940, 99188, null, null},
    {"Nov", 81991, 106181, null, null},
    {"Dec", 62188, 89128, null, null}
});

// Set formula for difference in column D and VarianceSparkline formula in column E
for (int i = 3; i < 15; i++)
{
    worksheet.Cells["$D{i}"].Formula = "(C" + i + "-B" + i + ")";
    worksheet.Cells["$E{i}"].Formula2 = "VARISPARKLINE(ROUND((D" + i + ")/(B" + i + ")),2),0,-1,1,,0.2,TRUE)";
}
}
```

Visual Basic

```
'Get sheet
Dim worksheet = FpSpread1_Sheet1.AsWorksheet()

'Set data
worksheet.SetValue(1, 0, New Object(,) {
    {"Sales", "Year 2019", "Year 2020", "Variance", "Variance %"},
    {"Jan", 65431, 74930, Nothing, Nothing},
    {"Feb", 83478, 92730, Nothing, Nothing},
    {"Mar", 90021, 12301, Nothing, Nothing},
    {"Apr", 72809, 23939, Nothing, Nothing},
}
```

```

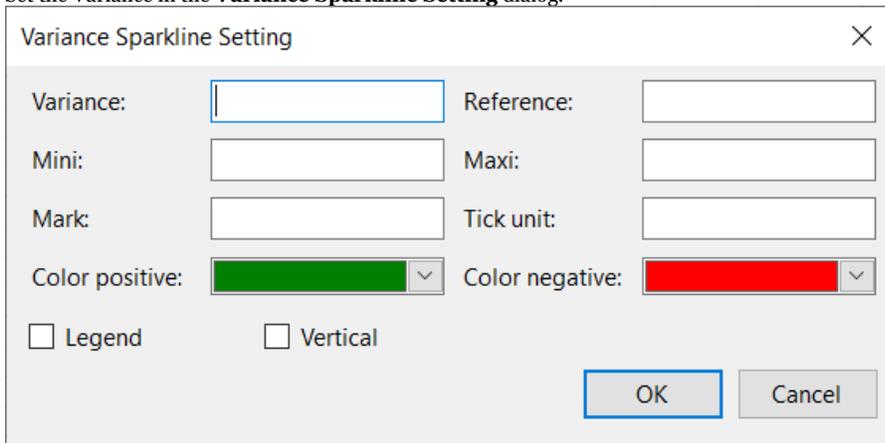
{"May", 103832, 34719, Nothing, Nothing},
{"Jun", 832833, 67189, Nothing, Nothing},
{"Jul", 671801, 73289, Nothing, Nothing},
{"Aug", 89222, 81299, Nothing, Nothing},
{"Sep", 68919, 91200, Nothing, Nothing},
{"Oct", 74940, 99188, Nothing, Nothing},
{"Nov", 81991, 106181, Nothing, Nothing},
{"Dec", 62188, 89128, Nothing, Nothing}
))

'Set formula for difference in column D and VarianceSparkline formula in column E
For i As Integer = 3 To 15 - 1
    worksheet.Cells($"D{i}").Formula = "(C" & i & "-B" & i & ")"
    worksheet.Cells($"E{i}").Formula2 = "VARISPARKLINE(ROUND((D" & i & ")/(B" & i & " ),2),0,-1,1,,0.2,TRUE)"
Next

```

Using the Spread Designer

1. Select a cell for the sparkline.
2. Select the Insert menu.
3. Select a sparkline type.
4. Set the Variance in the **Variance Sparkline Setting** dialog.



Set the additional sparkline settings as shown in the image above.

5. Select OK.
6. Select Apply and Exit from the File menu to save your changes and close the designer.

Keyboard Interaction

You can customize user interaction with your Spread component by mapping keyboard inputs from the user to particular actions on the Spread component using input maps and action maps. For example, pressing the Home key can be associated with moving the active cell to the first cell in the row. Many of the keys are mapped to actions by default. You can customize these default maps as well as add new mappings to create the interactions you want. The following topics describe the default mappings as well as procedures for setting up maps to customize keystroke processing.

- **Underlying Keystroke Processing**
- **Factors of Keyboard Map Usage**
- **Default Keyboard Navigation**
- **Default Keyboard Maps**
- **Deactivating the Default Keyboard Map**
- **Changing the Default Keyboard Map**
- **Using Input Maps with Action Maps**
- **Customizing the Input Maps**
- **Changing an Input Map for a Child View**
- **Using the Excel Compatibility Input Maps**
- **Saving and Loading Map Files**

For more information on the classes involved, refer to the **ActionMap** ('ActionMap Class' in the on-line documentation) and **InputMap** ('InputMap Class' in the on-line documentation) classes.

Underlying Keystroke Processing

The Spread component builds its keystroke processing on top of the underlying Windows keystroke processing.

The **System.Windows.Forms.Control** class provides the following methods and events for processing keystrokes that occur while the component has focus:

- **IsInputKey** method
- **IsInputChar** method
- **OnKeyDown** method
- **OnKeyPress** method
- **OnKeyUp** method
- **KeyDown** event
- **KeyPress** event
- **KeyUp** event

The **System.Windows.Forms.Control** class provides the following methods for processing keystrokes that occur while the component or one of its child controls has focus:

- **ProcessDialogKey**
- **ProcessDialogChar**

In addition to the methods and events in the .NET Framework that handle keyboard input, the Spread component provides input maps, as described in **Default Keyboard Maps**. Input maps provide a table-based description of the keyboard behavior for a Spread component. This allows the application to query the keyboard behavior for a Spread component. An input map is essentially a collection of keystrokes and related actions.

The Spread component provides a **WhenFocused** input map for processing keystrokes that occur while the component has focus. The **WhenFocused** input map is used to implement the **FpSpread** ('FpSpread Class' in the on-line documentation) class **IsInputKey**, **IsInputChar**, **OnKeyDown**, **OnKeyPress**, and **OnKeyUp** methods which in turn raise the **KeyDown**, **KeyPress**, and **KeyUp** events.

The Spread component provides a **WhenAncestorOffFocused** input map for processing keystrokes that occur while the component or one of its child controls has focus. The **WhenAncestorOffFocused** input map is used to implement the **FpSpread ('FpSpread Class' in the on-line documentation)** class's **ProcessDialogKey** and **ProcessDialogChar** methods. These methods do not raise any events. Most keystrokes processed by the Spread component must be processed whether the component or one of its child controls have focus, which means that most of the Spread keyboard behavior is described in the **WhenAncestorOffFocus** input map.

 **Note:** When keystrokes are processed by the **ProcessDialogKey** and **ProcessDialogChar** methods, no events are raised. This is true of most other grid-like components, including the Microsoft DataGrid.

The Spread component has multiple operation modes (Normal, ReadOnly, RowMode, SingleSelect, MultiSelect, ExtendedSelect). Each operation mode requires different keyboard behavior. Therefore, the Spread component has separate **WhenFocused** and **WhenAncestorOffFocused** input maps for each operation mode, as listed in the topic **Default Keyboard Maps**. For more discussion of these, refer to **Factors of Keyboard Map Usage**.

The predefined actions available for the Spread component are in the **SpreadActions ('SpreadActions Class' in the on-line documentation)** class. For a complete list of actions (including actions that do not have default keys), refer to the **SpreadActions ('SpreadActions Class' in the on-line documentation)** class.

If you want to turn off one of the default map keystroke settings, you can set the keystroke to the **None** member of the **SpreadActions ('SpreadActions Class' in the on-line documentation)** class.

For more information about the programming interface for inputs and actions and maps, refer to these classes:

- **Action ('Action Class' in the on-line documentation)** class
- **ActionMap ('ActionMap Class' in the on-line documentation)** class
- **InputMap ('InputMap Class' in the on-line documentation)** class
- **KeyStroke ('Keystroke Structure' in the on-line documentation)** class
- **SpreadActions ('SpreadActions Class' in the on-line documentation)** class

Factors of Keyboard Map Usage

There are several factors involved with using mappings that should be understood before either changing the default mappings or creating your own mappings. These include:

- Focus Location and Operation Mode
- Global versus Local
- Parent versus Child Workbook
- Enter Keystroke Binding Exception
- Action Called None

Focus Location and Operation Mode

Keyboard mapping is dependent on two factors: the focus location and the operation mode. For example, the Enter key has different actions depending if it is pressed when a cell editor has the focus or when the control has the focus. If the cell editor has the focus (the cell is in edit mode), pressing Enter takes the cell out of edit mode. If the control has the focus, and not a child control (such as a cell editor), pressing the Enter key puts the active cell in edit mode. For operation modes, for example, if the sheet is in ReadOnly mode, its map does not have entries for moving the active cell because there is no active cell. Customize the default maps keeping focus locations and operation modes in mind. Each mapping is like a dictionary, defining a meaning (an action) for each keystroke. Just as different dictionaries are needed for different languages, different keyboard maps are needed for different focus locations and operation modes.

The Spread component provides two input maps, **WhenFocused** and **WhenAncestorOffFocused**, one for each type of focus location. The **WhenFocused** map contains keyboard bindings that apply when the Spread component has focus but not when a child control has focus. Placing an entry in the **WhenFocused** input map is equivalent to overriding the **IsInputKey** and **IsInputChar** methods (to return true for the keystroke) and the **OnKeyDown**,

OnKeyPress, and **OnKeyUp** methods (to process the keystroke). The **WhenAncestorOfFocused** map contains keyboard bindings that apply when either the Spread component or a child control has focus. Placing an entry in the **WhenAncestorOfFocused** map is equivalent to overriding the **ProcessDialogKey** and **ProcessDialogChar** methods (to process the keystroke).

The first factor is what receives the keystroke or keystroke combination. Keystroke processing in .NET is handled in two phases: a pre-processing phase and a normal-processing phase. Most keystroke processing in Spread is handled during the pre-processing phase, which corresponds to the **WhenAncestorOfFocused** input map mode (refer to **InputMapMode ('InputMapMode Enumeration' in the on-line documentation)** enumeration settings). In the pre-processing phase, keystrokes are handled by the **IsInputChar**, **IsInputKey**, **ProcessDialogChar**, and **ProcessDialogKey** methods. The normal-processing phase corresponds to the **WhenFocused** input map mode. In the normal-processing phase, keystrokes are handled by the **OnKeyDown**, **OnKeyPress**, and **OnKeyUp** methods (which raise the **KeyDown**, **KeyPress**, and **KeyUp** events respectively).

For a typical Spread component, most interactions that occur while you are working with the component occur as part of the **WhenAncestorOfFocused** input map. Actions such as moving the active cell, selecting a range of cells, and others would be part of this map. For example, pressing the Tab key moves the active cell, even if a cell is in edit mode. In contrast, some keys are only mapped when there is not a cell in edit mode (the component has the focus, but not a cell editor). For example, the equals (=) key does not perform an action if pressed when a cell is in edit mode. If no cell is in edit mode, pressing the equals key starts formula editing for the active cell.

Global versus Local

The Spread component provides a parent global input map, which is shared by all Spread components in a project. When you customize the input map for a Spread component, you are customizing a local map for only that component.

The global map allows the Spread component to save memory by sharing a map. The local map lets you customize settings, then clear them later and retain the default settings as listed in **Default Keyboard Maps**.

Therefore, when working with input maps, you need to be aware of which methods work with the global map and which work with the local map.

The **Get** method to return input maps searches all input maps, including the global and the local. The **Put**, **Remove**, and **Clear** methods only work with the local map. For example, if you call the **Remove** method, it removes the custom settings for a local map, but does not change the default settings provided by the global map.

You cannot modify the global map itself, but if you want to do so, you can create a new input map and assign it to be the global map instead of the default one provided.

Parent versus Child Workbook

The keyboard bindings in the parent workbook's input map do not affect the child workbook. The Spread component contains one or more instances of the **SpreadView** class. In a hierarchy setup, the parent is one instance of the **SpreadView** class and each child is an instance of the **SpreadView** class. Each instance of the **SpreadView** class has its own input map. This gives you the option of having different keyboard behaviors at each level of the hierarchy. If you want a particular keyboard behavior at all levels in the hierarchy then you would need to modify the input map for the parent and each child.

Enter Keystroke Binding Exception

Most of the built-in keystrokes only have a binding in the **WhenAncestorOfFocused** map and thus only let you override one binding. The Enter keystroke is the exception. In the **WhenAncestorOfFocused** map, the Enter keystroke is bound to the **StopEditing** action. In the **WhenFocused** map, the Enter keystroke is bound to the **StartEditing** action. Thus, you need to override both bindings. Here is some sample code:

C#

```
InputMap whenAncestorOfFocusedMap =  
spread.GetInputMap(InputMapMode.WhenAncestorOfFocused);
```

```

InputMap whenFocusedMap = spread.GetInputMap(InputMapMode.WhenFocused);
whenAncestorOfFocusedMap.Put(new Keystroke(Keys.Enter, Keys.None),
SpreadActions.MoveToNextRow);
whenFocusedMap.Put(new Keystroke(Keys.Enter, Keys.None), SpreadActions.MoveToNextRow);

```

Action Called None

The **None** action is a special action indicating that the keystroke is not processed by the input map.

Placing a **None** action in an input map is similar to calling the **Remove** method on the input map. The difference between them is that a **None** action can override an entry in a parent map whereas the **Remove** method only affects entries in the specified map. By default, the **WhenFocused** and **WhenAncestorOfFocused** maps have no direct entries but do have indirect entries via parent maps. The **WhenFocused**'s parent map contains no binding for the Esc key.

Default Keyboard Navigation

The default behavior for end-user keyboard action is summarized in these tables.

Default Behavior on a Sheet

The default behavior for end-user keyboard action on the sheet is summarized in this table.

Key Code	Action	Action Name
Escape	If edit mode is on, previous cell value replaces new value and edit mode is turned off	CancelEditing ('CancelEditing Field' in the on-line documentation)
F2	If edit mode is on, clears the active cell value	ClearCell ('ClearCell Field' in the on-line documentation)
Ctrl+C or Ctrl+Insert	Copies the selection to the Clipboard	ClipboardCopy ('ClipboardCopy Field' in the on-line documentation)
Ctrl+X or Shift+Delete	Cuts the selection to the Clipboard	ClipboardCut ('ClipboardCut Field' in the on-line documentation)
Ctrl+V or Shift+Insert	Pastes the data and formatting from the Clipboard	ClipboardPasteAll ('ClipboardPasteAll Field' in the on-line documentation)
F3	If edit mode is on, places the current date and time in a date-time cell	DateTimeNow ('DateTimeNow Field' in the on-line documentation)
Ctrl + Shift + Home	Extends the selection to include the first cell	ExtendToFirstCell ('ExtendToFirstCell Field' in the on-line documentation)
Shift + Home	Extends the selection to include the first column	ExtendToFirstColumn ('ExtendToFirstColumn Field' in the on-line documentation)
Shift + Home	Extends the selection to include the first item in a list	ExtendToFirstItem ('ExtendToFirstItem Field' in the on-line documentation)

Ctrl + Shift + End	Extends the selection to include the last cell	ExtendToLastCell ('ExtendToLastCell Field' in the on-line documentation)
Shift + End	Extends the selection to include the last column	ExtendToLastColumn ('ExtendToLastColumn Field' in the on-line documentation)
Shift + End	Extends the selection to include the last item in a list	ExtendToLastItem ('ExtendToLastItem Field' in the on-line documentation)
Shift + Right Arrow or Ctrl + Shift + Right Arrow	Extends selection right one column by index	ExtendToNextColumn ('ExtendToNextColumn Field' in the on-line documentation)
Shift + Right Arrow or Ctrl + Shift + Right Arrow	Extends selection right one column by visual	ExtendToNextColumnVisual ('ExtendToNextColumnVisual Field' in the on-line documentation)
Shift + Down Arrow	Extends selection down one row	ExtendToNextItem ('ExtendToNextItem Field' in the on-line documentation)
Ctrl + Shift + Page Down	Extends selection right one page of columns	ExtendToNextPageOfColumns ('ExtendToNextPageOfColumns Field' in the on-line documentation)
Shift + Page Down	Extends selection right one page of items	ExtendToNextPageOfItems ('ExtendToNextPageOfItems Field' in the on-line documentation)
Shift + Page Down	Extends selection down one page of rows	ExtendToNextPageOfRows ('ExtendToNextPageOfRows Field' in the on-line documentation)
Shift + Down Arrow or Ctrl + Shift + Down Arrow	Extends selection down one row	ExtendToNextRow ('ExtendToNextRow Field' in the on-line documentation)
Shift + Left Arrow or Ctrl + Shift + Left Arrow	Extends selection left one column by index	ExtendToPreviousColumn ('ExtendToPreviousColumn Field' in the on-line documentation)
Shift + Left Arrow or Ctrl + Shift + Left Arrow	Extends selection left one column by visual	ExtendToPreviousColumnVisual ('ExtendToPreviousColumnVisual Field' in the on-line documentation)
Shift + Up Arrow	Extends selection left one item	ExtendToNextItem ('ExtendToNextItem Field' in the on-line documentation)
Ctrl + Shift	Extends selection left one page of columns	ExtendToPreviousPageOfColumns

+ Page Up		(ExtendToPreviousPageOfColumns Field' in the on-line documentation)
Shift + Page Up	Extends selection up one page of items	ExtendToPreviousPageOfItems ('ExtendToPreviousPageOfItems Field' in the on-line documentation)
Shift + Page Up	Extends selection up one page of rows	ExtendToPreviousPageOfRows ('ExtendToPreviousPageOfRows Field' in the on-line documentation)
Shift + Up Arrow or Ctrl + Shift + Up Arrow	Extends selection up one row	ExtendToPreviousRow ('ExtendToPreviousRow Field' in the on-line documentation)
Ctrl + Home	Moves active cell to first row, first column	MoveToFirstCell ('MoveToFirstCell Field' in the on-line documentation)
Home	Moves active cell to the first cell in the row	MoveToFirstColumn ('MoveToFirstColumn Field' in the on-line documentation)
Home	Moves active cell to the first item in the list	MoveToFirstItem ('MoveToFirstItem Field' in the on-line documentation)
Ctrl + End	Moves active cell to last row, last column	MoveToLastCell ('MoveToLastCell Field' in the on-line documentation)
End	Moves active cell to the last cell in the row	MoveToLastColumn ('MoveToLastColumn Field' in the on-line documentation)
End	Moves active cell to the last item in the list	MoveToLastItem ('MoveToLastItem Field' in the on-line documentation)
Right Arrow or Ctrl + Right Arrow	Moves active cell right one column by index	MoveToNextColumn ('MoveToNextColumn Field' in the on-line documentation)
Right Arrow or Ctrl + Right Arrow	Moves active cell right one column	MoveToNextColumnVisual ('MoveToNextColumnVisual Field' in the on-line documentation)
Tab	Moves the active cell to the next column and wraps at the end of the row. (The Tab key skips hidden columns automatically.)	MoveToNextColumnWrap ('MoveToNextColumnWrap Field' in the on-line documentation)
Down Arrow	Moves to the next item in the list.	MoveToNextItem ('MoveToNextItem Field' in the on-line documentation)
Ctrl + Page Down	Moves active cell right one page of columns	MoveToNextPageOfColumns ('MoveToNextPageOfColumns Field' in the on-line documentation)

Page Down	Moves down one page of items	MoveToNextPageOfItems ('MoveToNextPageOfItems Field' in the on-line documentation)
Page Down	Moves active cell down one page of rows	MoveToNextPageOfRows ('MoveToNextPageOfRows Field' in the on-line documentation)
Down Arrow	Moves active cell down one row	MoveToNextRow ('MoveToNextRow Field' in the on-line documentation)
Down Arrow or Ctrl + Down Arrow	Moves active cell down one row	MoveToNextRow ('MoveToNextRow Field' in the on-line documentation)
Left row or Ctrl + Left Arrow	Moves active cell left one column by index	MoveToPreviousColumn ('MoveToPreviousColumn Field' in the on-line documentation)
Left Arrow or Ctrl + Left Arrow	Moves active cell left one column by visual	MoveToPreviousColumnVisual Field' in the on-line documentation)
Shift + Tab	Moves the active cell to the previous column and wraps at the end of the row. (The Tab key skips hidden columns automatically).	MoveToPreviousColumnWrap ('MoveToPreviousColumnWrap Field' in the on-line documentation)
Ctrl + Left Arrow	Moves to previous item in the list	MoveToPreviousItem ('MoveToPreviousItem Field' in the on-line documentation)
Ctrl + Page Up	Moves active cell left one page of columns	MoveToPreviousPageOfColumns ('MoveToPreviousPageOfColumns Field' in the on-line documentation)
Page Up	Moves up one page of items	MoveToPreviousPageOfItems ('MoveToPreviousPageOfItems Field' in the on-line documentation)
Page Up	Moves active cell up one page of rows	MoveToPreviousPageOfRows ('MoveToPreviousPageOfRows Field' in the on-line documentation)
Up Arrow or Ctrl + Up Arrow	Moves active cell up one row	MoveToPreviousRow ('MoveToPreviousRow Field' in the on-line documentation)
Ctrl + Y	Moves active cell up one row	Redo ('Redo Field' in the on-line documentation)
Ctrl + Home	Scrolls to display the first cell	ScrollToFirstCell ('ScrollToFirstCell Field' in the on-line documentation)
Home	Scrolls to display the first column	ScrollToFirstColumn ('ScrollToFirstColumn Field' in the on-line documentation)

Ctrl + End	Scrolls to display the last cell	ScrollToLastCell ('ScrollToLastCell Field' in the on-line documentation)
End	Scrolls to display the last column	ScrollToLastColumn ('ScrollToLastColumn Field' in the on-line documentation)
Right	Scrolls to display the next column by index	ScrollToNextColumn ('ScrollToNextColumn Field' in the on-line documentation)
Right	Scrolls to display the next column by visual	ScrollToNextColumnVisual ('ScrollToNextColumnVisual Field' in the on-line documentation)
Ctrl + Page Down	Scrolls to display the next page of columns	ScrollToNextPageOfColumns ('ScrollToNextPageOfColumns Field' in the on-line documentation)
Page Down	Scrolls to display the next page of rows	ScrollToNextPageOfRows ('ScrollToNextPageOfRows Field' in the on-line documentation)
Down	Scrolls to display the next row	ScrollToNextRow ('ScrollToNextRow Field' in the on-line documentation)
Left	Scrolls to display the previous column by index	ScrollToPreviousColumn ('ScrollToPreviousColumn Field' in the on-line documentation)
Left	Scrolls to display the previous column by visual	ScrollToPreviousColumnVisual ('ScrollToPreviousColumnVisual Field' in the on-line documentation)
Ctrl + Page Up	Scrolls to display the previous page of columns	ScrollToPreviousPageOfColumns ('ScrollToPreviousPageOfColumns Field' in the on-line documentation)
Page Up	Scrolls to display the previous page of rows	ScrollToPreviousPageOfRows ('ScrollToPreviousPageOfRows Field' in the on-line documentation)
Up Arrow	Scrolls to display the previous row	ScrollToPreviousRow ('ScrollToPreviousRow Field' in the on-line documentation)
Ctrl + spacebar	Selects the column containing the active cell	SelectColumn ('SelectColumn Field' in the on-line documentation)
Home	Selects the first item in the list	SelectFirstItem ('SelectFirstItem Field' in the on-line documentation)
End	Selects the last item in the list	SelectLastItem ('SelectLastItem Field' in the on-line documentation)
Down	Selects the next item in the list	SelectNextItem ('SelectNextItem

Page Down	Selects the next page of items in the list	Field' in the on-line documentation) SelectNextPageOfItems ('SelectNextPageOfItems Field' in the on-line documentation)
Up Arrow	Selects the previous item in the list	SelectPreviousItem ('SelectPreviousItem Field' in the on-line documentation)
Page Up	Selects the previous page of items in the list	SelectPreviousPageOfItems ('SelectPreviousPageOfItems Field' in the on-line documentation)
Shift + spacebar	Selects the row containing the active cell	SelectRow ('SelectRow Field' in the on-line documentation)
Ctrl + Shift + spacebar	Selects the current sheet	SelectSheet ('SelectSheet Field' in the on-line documentation)
F4	If edit mode is enabled in a date cell, spreadsheet displays a pop-up calendar to let you choose a date. If the edit mode is enabled in a formula text box or in a cell containing formula and a cell reference or range is selected, pressing the F4 key will show up various combinations of absolute and relative references. For example, let's say you have entered a formula "=A1" in a cell. Now, everytime the F4 key is pressed, the formula will change as per this cycle : A1 -> \$A\$1 -> A\$1 -> \$A1 -> A1	ShowSubEditor ('ShowSubEditor Field' in the on-line documentation) SwitchReferenceType ('SwitchReferenceType Method' in the on-line documentation)
Enter or Backspace	Begins editing; stops editing if edit mode is on.	StartEditing ('StartEditing Field' in the on-line documentation) or StopEditing ('StopEditing Field' in the on-line documentation)
=	Begins editing formula	StartEditingFormula ('StartEditingFormula Field' in the on-line documentation)
Alt + =	Calculates sum of rows and/or columns.	AutoSum ('AutoSum Field' in the on-line documentation)
Ctrl + Z	Moves active cell up one row	Undo ('Undo Field' in the on-line documentation)

Keyboard navigation is defined by default maps, that map user keyboard actions with Spread component actions. For example, by default, pressing Tab moves the active cell to the next column. You can customize any or all of the keyboard actions by mapping them to Spread component actions.

The built-in keyboard actions (for example, **MoveToNextRowWrap**) treat a cell span as existing in both columns or rows. You can enter the span by navigating down either column or row. When leaving the span in a backwards direction (for example, **MoveToPreviousRowWrap**), the built-in action uses the upper left corner of the span for computing the previous column or row. When leaving the span in a forwards direction (for example, **MoveToNextRowWrap**), the built-in action uses the lower right corner of the span for computing the new column or row.

For the Ctrl+PageUp and Ctrl+PageDown keys, if you want your application to mimic the behavior found in Excel (that is, move left or right one sheet regardless of number of sheets) then rebind the keystrokes to the **MoveToPreviousSheet** and **MoveToNextSheet** actions.

Actions that extend, move, or scroll to the next or previous column use the visual layout of the screen by default. The

previous column is a column that is visually left of the active column and the next column is column that is visually right of the active column. In Spread Windows Forms 2.5, cell coordinates were used. In cell coordinates, the previous column is the active column - 1 and the next column is the active column + 1. The cell coordinate actions are still available and are listed in the **SpreadActions ('SpreadActions Class' in the on-line documentation)** class.

Default Behavior for Shapes on a Sheet

The default navigation keys for shapes on a sheet are used with all operation modes and can be changed with the **SetInputMapWhenShapeHasFocus ('SetInputMapWhenShapeHasFocus Method' in the on-line documentation)** method. The default navigation keys for shapes are listed in the following table.

Key Code	Action	Action Name
Tab	Moves to next shape	ActivateNextShape ('ActivateNextShape Field' in the on-line documentation)
Shift + Tab	Moves to previous shape	ActivatePreviousShape ('ActivatePreviousShape Field' in the on-line documentation)
Ctrl + C or Ctrl + Insert	Copies shape	ClipboardCopyShape ('ClipboardCopyShape Field' in the on-line documentation)
Ctrl + X or Shift + Delete	Cuts shape	ClipboardCutShape ('ClipboardCutShape Field' in the on-line documentation)
Ctrl + V or Shift + Insert	Pastes shape	ClipboardPasteShape ('ClipboardPasteShape Field' in the on-line documentation)
N/A	Cuts data only	ClipboardCutDataOnly ('ClipboardCutDataOnly Field' in the on-line documentation)
Escape	Deactivates shape	DeactivateShape ('DeactivateShape Field' in the on-line documentation)
Ctrl + Up Arrow	Decreases shape height	DecreaseShapeHeight ('DecreaseShapeHeight Field' in the on-line documentation)
Ctrl + Left Arrow	Decreases shape width	DecreaseShapeWidth ('DecreaseShapeWidth Field' in the on-line documentation)
Delete	Deletes shape	DeleteShape ('DeleteShape Field' in the on-line documentation)
Ctrl + Down Arrow	Increases shape height	IncreaseShapeHeight ('IncreaseShapeHeight Field' in the on-line documentation)
Ctrl + Right Arrow	Increases shape width	IncreaseShapeWidth ('IncreaseShapeWidth Field' in the on-line documentation)
Down Arrow	Moves shape down	MoveShapeDown ('MoveShapeDown Field' in the on-line documentation)
Left Arrow	Moves shape left	MoveShapeLeft ('MoveShapeLeft Field' in the on-line documentation)
Right Arrow	Moves shape right	MoveShapeRight ('MoveShapeRight Field' in the on-line documentation)
Up Arrow	Moves shape up	MoveShapeUp ('MoveShapeUp Field' in the on-line documentation)
Alt + Right Arrow	Rotates shape clockwise	RotateShapeClockwise ('RotateShapeClockwise Field' in the on-line documentation)
Alt + Left Arrow	Rotates shape counter-clockwise	RotateShapeCounterClockwise ('RotateShapeCounterClockwise Field' in the on-line documentation)

For more information about shapes, refer to **Customizing Drawing**.

Default Behavior for Child Controls on a Sheet

The default navigation keys for child controls on a sheet are used with all operation modes and can be changed with the **SetInputMapWhenChildHasFocus** ('**SetInputMapWhenChildHasFocus Method**' in the **on-line documentation**) method. The default navigation keys for child controls on a sheet are listed in the following table.

Key Code	Action	Action Name
Tab	Moves to next control	ActivateNextChild (' ActivateNextChild Field ' in the on-line documentation)
Shift + Tab	Moves to previous control	ActivateNextShape (' ActivateNextShape Field ' in the on-line documentation)
Escape	Deactivates control	DeactivateChild (' DeactivateChild Field ' in the on-line documentation)
Ctrl + Up Arrow	Decreases control height	DecreaseChildHeight (' DecreaseChildHeight Field ' in the on-line documentation)
Ctrl + Left Arrow	Decreases control width	DecreaseShapeWidth (' DecreaseShapeWidth Field ' in the on-line documentation)
Delete	Deletes control	DeleteChild (' DeleteChild Field ' in the on-line documentation)
Ctrl + Down Arrow	Increases control height	IncreaseChildHeight (' IncreaseChildHeight Field ' in the on-line documentation)
Ctrl + Right Arrow	Increases control width	IncreaseChildWidth (' IncreaseChildWidth Field ' in the on-line documentation)
Down Arrow	Moves control down	MoveChildDown (' MoveChildDown Field ' in the on-line documentation)
Left Arrow	Moves control left	MoveChildLeft (' MoveChildLeft Field ' in the on-line documentation)
Right Arrow	Moves control right	MoveChildRight (' MoveChildRight Field ' in the on-line documentation)
Up Arrow	Moves control up	MoveChildUp (' MoveChildUp Field ' in the on-line documentation)

For more information about controls, refer to **Placing Child Controls on a Sheet**.

Default Keyboard Maps

Spread provides twelve default maps that map keystrokes to actions for each focus location (also referred to as input map mode) and operation mode. You can customize any or all of these maps to change the action associated with a keystroke or to add additional actions for other keystrokes.

The following default maps are provided with Spread Windows Forms for each operation mode and input map mode.

- **Default Map for Excel Compatibility**
- **Default Map for Normal and WhenFocused**
- **Default Map for Normal and WhenAncestorOfFocused**
- **Default Map for ReadOnly and WhenFocused**
- **Default Map for ReadOnly and WhenAncestorOfFocused**
- **Default Map for RowMode and WhenFocused**

- **Default Map for RowMode and WhenAncestorOfFocused**
- **Default Map for SingleSelect and WhenFocused**
- **Default Map for SingleSelect and WhenAncestorOfFocused**
- **Default Map for MultiSelect and WhenFocused**
- **Default Map for MultiSelect and WhenAncestorOfFocused**
- **Default Map for ExtendedSelect and WhenFocused**
- **Default Map for ExtendedSelect and WhenAncestorOfFocused**

Default Map for Normal and WhenFocused

The default map for this operation mode and input map mode is summarized in this table.

Key Code	Action Name
Ctrl+C	ClipboardCopy ('ClipboardCopy Field' in the on-line documentation)
Ctrl+Insert	ClipboardCopy ('ClipboardCopy Field' in the on-line documentation)
Ctrl+X	ClipboardCut ('ClipboardCut Field' in the on-line documentation)
Shift+Delete	ClipboardCut ('ClipboardCut Field' in the on-line documentation)
Ctrl+V	ClipboardPasteAll ('ClipboardPasteAll Field' in the on-line documentation)
Shift+Insert	ClipboardPasteAll ('ClipboardPasteAll Field' in the on-line documentation)
Enter	StartEditing ('StartEditing Field' in the on-line documentation)
=	StartEditingFormula ('StartEditingFormula Field' in the on-line documentation)

Default Map for Normal and WhenAncestorOfFocused

The default map for this operation mode and input map mode is summarized in this table.

Key Code	Action Name
Esc	CancelEditing ('CancelEditing Field' in the on-line documentation)
Alt+Down Arrow	ComboShowList ('ComboShowList Field' in the on-line documentation)
Alt+Up Arrow	ComboShowList ('ComboShowList Field' in the on-line documentation)
F2	ClearCell ('ClearCell Field' in the on-line documentation)
F3	DateTimeNow ('DateTimeNow Field' in the on-line documentation)
Ctrl+Shift+Home	ExtendToFirstCell ('ExtendToFirstCell Field' in the on-line documentation)
Shift+Home	ExtendToFirstColumn ('ExtendToFirstColumn Field' in the on-line documentation)
Ctrl+Shift+End	ExtendToLastCell ('ExtendToLastCell Field' in the on-line documentation)
Shift+End	ExtendToLastColumn ('ExtendToLastColumn Field' in the on-line documentation)
Shift+Right Arrow	ExtendToNextColumnVisual ('ExtendToNextColumnVisual Field' in the on-line documentation)
Ctrl+Shift+Right Arrow	ExtendToNextColumnVisual ('ExtendToNextColumnVisual Field' in the on-line documentation)

Ctrl+Shift+Page Down	ExtendToNextPageOfColumns ('ExtendToNextPageOfColumns Field' in the on-line documentation)
Page Down	ExtendToNextPageOfRows ('ExtendToNextPageOfRows Field' in the on-line documentation)
Shift+Page Down	ExtendToNextPageOfRows ('ExtendToNextPageOfRows Field' in the on-line documentation)
Shift+Down Arrow	ExtendToNextRow ('ExtendToNextRow Field' in the on-line documentation)
Ctrl+Shift+Down Arrow	ExtendToNextRow ('ExtendToNextRow Field' in the on-line documentation)
Shift+Left Arrow	ExtendToPreviousColumnVisual ('ExtendToPreviousColumnVisual Field' in the on-line documentation)
Ctrl+Shift+Left Arrow	ExtendToPreviousColumnVisual ('ExtendToPreviousColumnVisual Field' in the on-line documentation)
Ctrl+Shift+Page Up	ExtendToPreviousPageOfColumns ('ExtendToPreviousPageOfColumns Field' in the on-line documentation)
Shift+Up Arrow	ExtendToPreviousRow ('ExtendToPreviousRow Field' in the on-line documentation)
Ctrl+Shift+Up Arrow	ExtendToPreviousRow ('ExtendToPreviousRow Field' in the on-line documentation)
Page Up	ExtendToPreviousPageOfRows ('ExtendToPreviousPageOfRows Field' in the on-line documentation)
Shift+Page Up	ExtendToPreviousPageOfRows ('ExtendToPreviousPageOfRows Field' in the on-line documentation)
Ctrl+Home	MoveToFirstCell ('MoveToFirstCell Field' in the on-line documentation)
Home	MoveToFirstColumn ('MoveToFirstColumn Field' in the on-line documentation)
Ctrl+End	MoveToLastCell ('MoveToLastCell Field' in the on-line documentation)
End	MoveToLastColumn ('MoveToLastColumn Field' in the on-line documentation)
Right Arrow	MoveToNextColumnVisual ('MoveToNextColumnVisual Field' in the on-line documentation)
Ctrl+Right Arrow	MoveToNextColumnVisual ('MoveToNextColumnVisual Field' in the on-line documentation)
Tab	MoveToNextColumnWrap ('MoveToNextColumnWrap Field' in the on-line documentation)
Down Arrow	MoveToNextRow ('MoveToNextRow Field' in the on-line documentation)
Ctrl+Down Arrow	MoveToNextRow ('MoveToNextRow Field' in the on-line documentation)
Left Arrow	MoveToPreviousColumnVisual ('MoveToPreviousColumnVisual Field' in the on-line documentation)
Ctrl+Left Arrow	MoveToPreviousColumnVisual ('MoveToPreviousColumnVisual Field' in the on-line documentation)
Shift+Tab	MoveToPreviousColumnWrap ('MoveToPreviousColumnWrap Field' in the on-line documentation)
Ctrl+Page Up	MoveToPreviousPageOfRows ('MoveToPreviousPageOfRows Field' in the on-line documentation)

Ctrl+Page Down	MoveToPreviousPageOfColumns ('MoveToPreviousPageOfColumns Field' in the on-line documentation)
Ctrl+Up Arrow	MoveToPreviousRow ('MoveToPreviousRow Field' in the on-line documentation)
Up Arrow	MoveToPreviousRow ('MoveToPreviousRow Field' in the on-line documentation)
Ctrl+spacebar	SelectColumn ('SelectColumn Field' in the on-line documentation)
Shift+spacebar	SelectRow ('SelectRow Field' in the on-line documentation)
Enter	SelectRow ('SelectRow Field' in the on-line documentation)
Ctrl+Shift+spacebar	SelectSheet ('SelectSheet Field' in the on-line documentation)
F4	ShowSubEditor ('ShowSubEditor Field' in the on-line documentation)

Default Map for ReadOnly and WhenFocused

No default actions are defined for this operation mode and input map mode.

Default Map for ReadOnly and WhenAncestorOfFocused

The default map for this operation mode and input map mode is summarized in this table.

Key Code Action Name

Ctrl+Home	ScrollToFirstCell ('ScrollToFirstCell Field' in the on-line documentation)
Home	ScrollToFirstColumn ('ScrollToFirstColumn Field' in the on-line documentation)
Ctrl+End	ScrollToLastCell ('ScrollToLastCell Field' in the on-line documentation)
End	ScrollToLastColumn ('ScrollToLastColumn Field' in the on-line documentation)
Right Arrow	ScrollToNextColumnVisual ('ScrollToNextColumnVisual Field' in the on-line documentation)
Ctrl+Page Down	ScrollToNextPageOfColumns ('ScrollToNextPageOfColumns Field' in the on-line documentation)
Page Down	ScrollToNextPageOfRows ('ScrollToNextPageOfRows Field' in the on-line documentation)
Down Arrow	ScrollToNextRow ('ScrollToNextRow Field' in the on-line documentation)
Left Arrow	ScrollToPreviousColumnVisual ('ScrollToPreviousColumnVisual Field' in the on-line documentation)
Ctrl+Page Up	ScrollToPreviousPageOfColumns ('ScrollToPreviousPageOfColumns Field' in the on-line documentation)
Page Up	ScrollToPreviousPageOfRows ('ScrollToPreviousPageOfRows Field' in the on-line documentation)
Up Arrow	ScrollToPreviousRow ('ScrollToPreviousRow Field' in the on-line documentation)

Default Map for ReadOnly and WhenAncestorOfFocused

The default map for this operation mode and input map mode is summarized in this table.

Key Code Action Name

Ctrl+Home	ScrollToFirstCell ('ScrollToFirstCell Field' in the on-line documentation)
Home	ScrollToFirstColumn ('ScrollToFirstColumn Field' in the on-line documentation)
Ctrl+End	ScrollToLastCell ('ScrollToLastCell Field' in the on-line documentation)
End	ScrollToLastColumn ('ScrollToLastColumn Field' in the on-line documentation)
Right Arrow	ScrollToNextColumnVisual ('ScrollToNextColumnVisual Field' in the on-line documentation)
Ctrl+Page Down	ScrollToNextPageOfColumns ('ScrollToNextPageOfColumns Field' in the on-line documentation)
Page Down	ScrollToNextPageOfRows ('ScrollToNextPageOfRows Field' in the on-line documentation)
Down Arrow	ScrollToNextRow ('ScrollToNextRow Field' in the on-line documentation)
Left Arrow	ScrollToPreviousColumnVisual ('ScrollToPreviousColumnVisual Field' in the on-line documentation)
Ctrl+Page Up	ScrollToPreviousPageOfColumns ('ScrollToPreviousPageOfColumns Field' in the on-line documentation)
Page Up	ScrollToPreviousPageOfRows ('ScrollToPreviousPageOfRows Field' in the on-line documentation)
Up Arrow	ScrollToPreviousRow ('ScrollToPreviousRow Field' in the on-line documentation)

Default Map for RowMode and WhenAncestorOfFocused

The default map for this operation mode and input map mode is summarized in this table.

Key Code Action Name

Esc	CancelEditing ('CancelEditing Field' in the on-line documentation)
Ctrl+Home	MoveToFirstCell ('MoveToFirstCell Field' in the on-line documentation)
Home	MoveToFirstColumn ('MoveToFirstColumn Field' in the on-line documentation)
Ctrl+End	MoveToLastCell ('MoveToLastCell Field' in the on-line documentation)
End	MoveToLastColumn ('MoveToLastColumn Field' in the on-line documentation)
Right Arrow	MoveToNextColumnVisual ('MoveToNextColumnVisual Field' in the on-line documentation)
Ctrl+Right Arrow	MoveToNextColumnVisual ('MoveToNextColumnVisual Field' in the on-line documentation)
Tab	MoveToNextColumnWrap ('MoveToNextColumnWrap Field' in the on-line documentation)
Ctrl+Page Down	MoveToNextPageOfColumns ('MoveToNextPageOfColumns Field' in the on-line documentation)
Page Down	MoveToNextPageOfRows ('MoveToNextPageOfRows Field' in the on-line documentation)
Down Arrow	MoveToNextRow ('MoveToNextRow Field' in the on-line documentation)
Ctrl+Down	MoveToNextRow ('MoveToNextRow Field' in the on-line documentation)

Arrow	
Left Arrow	MoveToPreviousColumnVisual ('MoveToPreviousColumnVisual Field' in the on-line documentation)
Ctrl+Left Arrow	MoveToPreviousColumnVisual ('MoveToPreviousColumnVisual Field' in the on-line documentation)
Shift+Tab	MoveToPreviousColumnWrap ('MoveToPreviousColumnWrap Field' in the on-line documentation)
Ctrl+Page Up	MoveToPreviousPageOfColumns ('MoveToPreviousPageOfColumns Field' in the on-line documentation)
Page Up	MoveToPreviousPageOfRows ('MoveToPreviousPageOfRows Field' in the on-line documentation)
Up Arrow	MoveToPreviousRow ('MoveToPreviousRow Field' in the on-line documentation)
Ctrl+Up Arrow	MoveToPreviousRow ('MoveToPreviousRow Field' in the on-line documentation)
Enter	StopEditing ('StopEditing Field' in the on-line documentation)

Default Map for SingleSelect and WhenFocused

The default map for this operation mode and input map mode is summarized in this table.

Key Code	Action Name
Ctrl+C	ClipboardCopy ('ClipboardCopy Field' in the on-line documentation)
Ctrl+Insert	ClipboardCopy ('ClipboardCopy Field' in the on-line documentation)

Default Map for SingleSelect and WhenAncestorOfFocused

The default map for this operation mode and input map mode is summarized in this table.

Key Code	Action Name
Right Arrow	ScrollToNextColumnVisual ('ScrollToNextColumnVisual Field' in the on-line documentation)
Left Arrow	ScrollToPreviousColumnVisual ('ScrollToPreviousColumnVisual Field' in the on-line documentation)
Home	SelectFirstItem ('SelectFirstItem Field' in the on-line documentation)
End	SelectLastItem ('SelectLastItem Field' in the on-line documentation)
Down Arrow	SelectNextItem ('SelectNextItem Field' in the on-line documentation)
Page Down	SelectNextPageOfItems ('SelectNextPageOfItems Field' in the on-line documentation)
Up Arrow	SelectPreviousItem ('SelectPreviousItem Field' in the on-line documentation)
Page Up	SelectPreviousPageOfItems ('SelectPreviousPageOfItems Field' in the on-line documentation)

Default Map for MultiSelect and WhenFocused

The default map for this operation mode and input map mode is summarized in this table.

Key Code	Action Name
Ctrl+C	ClipboardCopy ('ClipboardCopy Field' in the on-line documentation)
Ctrl+Insert	ClipboardCopy ('ClipboardCopy Field' in the on-line documentation)

Default Map for MultiSelect and WhenAncestorOfFocused

The default map for this operation mode and input map mode is summarized in this table.

Key Code	Action Name
Home	MoveToFirstItem ('MoveToFirstItem Field' in the on-line documentation)
End	MoveToLastItem ('MoveToLastItem Field' in the on-line documentation)
Down Arrow	MoveToNextItem ('MoveToNextItem Field' in the on-line documentation)
Page Down	MoveToNextPageOfItems ('MoveToNextPageOfItems Field' in the on-line documentation)
Up Arrow	MoveToPreviousItem ('MoveToPreviousItem Field' in the on-line documentation)
Page Up	MoveToPreviousPageOfItems ('MoveToPreviousPageOfItems Field' in the on-line documentation)
Right Arrow	ScrollToNextColumnVisual ('ScrollToNextColumnVisual Field' in the on-line documentation)
Left Arrow	ScrollToPreviousColumnVisual ('ScrollToPreviousColumnVisual Field' in the on-line documentation)
spacebar	ToggleItem ('ToggleItem Field' in the on-line documentation)

Default Map for ExtendedSelect and WhenFocused

The default map for this operation mode and input map mode is summarized in this table.

Key Code	Action Name
Ctrl+C	ClipboardCopy ('ClipboardCopy Field' in the on-line documentation)
Ctrl+Insert	ClipboardCopy ('ClipboardCopy Field' in the on-line documentation)

Default Map for ExtendedSelect and WhenAncestorOfFocused

The default map for this operation mode and input map mode is summarized in this table.

Key Code	Action Name
Right	ScrollToNextColumnVisual ('ScrollToNextColumnVisual Field' in the on-line documentation)

	documentation)
Left Arrow	ScrollToPreviousColumnVisual ('ScrollToPreviousColumnVisual Field' in the on-line documentation)
Home	SelectFirstItem ('SelectFirstItem Field' in the on-line documentation)
End	SelectLastItem ('SelectLastItem Field' in the on-line documentation)
Down Arrow	SelectNextItem ('SelectNextItem Field' in the on-line documentation)
Page Down	SelectNextPageOfItems ('SelectNextPageOfItems Field' in the on-line documentation)
Page Up	SelectPreviousPageOfItems ('SelectPreviousPageOfItems Field' in the on-line documentation)
UpArrow	SelectPreviousItem ('SelectPreviousItem Field' in the on-line documentation)

Deactivating the Default Keyboard Map

You can deactivate or turn off the default input map.

Using Code

1. Create an **InputMap ('InputMap Class' in the on-line documentation)** object.
2. Use the **GetInputMap ('GetInputMap Method' in the on-line documentation)** method.

Example

This example deactivates the F2 key.

C#

```
private void Form1_Load(object sender, System.EventArgs e)
{
    FarPoint.Win.Spread.InputMap im = new FarPoint.Win.Spread.InputMap();
    // Deactivate F2 key in cells not being edited.
    im = fpSpread1.GetInputMap(FarPoint.Win.Spread.InputMapMode.WhenFocused);
    im.Put(new FarPoint.Win.Spread.Keystroke(Keys.F2, Keys.None),
FarPoint.Win.Spread.SpreadActions.None);
    // Deactivate F2 key in cells being edited.
    im = fpSpread1.GetInputMap(FarPoint.Win.Spread.InputMapMode.WhenAncestorOfFocused);
    im.Put(new FarPoint.Win.Spread.Keystroke(Keys.F2, Keys.None),
FarPoint.Win.Spread.SpreadActions.None);
}
```

VB

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
MyBase.Load
    Dim im As New FarPoint.Win.Spread.InputMap
    ' Deactivate F2 key in cells not being edited.
    im = fpSpread1.GetInputMap(FarPoint.Win.Spread.InputMapMode.WhenFocused)
    im.Put(New FarPoint.Win.Spread.Keystroke(Keys.F2, Keys.None),
FarPoint.Win.Spread.SpreadActions.None)
    ' Deactivate F2 key in cells being edited.
    im = fpSpread1.GetInputMap(FarPoint.Win.Spread.InputMapMode.WhenAncestorOfFocused)
    im.Put(New FarPoint.Win.Spread.Keystroke(Keys.F2, Keys.None),
```

```
FarPoint.Win.Spread.SpreadActions.None)  
End Sub
```

Changing the Default Keyboard Map

The default input map defines the behavior of the component for end user interaction with the keyboard. For example, by default, when the end user presses the Enter key in an active cell, the edit mode turns on for that cell. You can change this default behavior, by changing the default input map.

Using Code

1. Create an **InputMap ('InputMap Class' in the on-line documentation)** object.
2. Use the **GetInputMap ('GetInputMap Method' in the on-line documentation)** method.

Example

This example changes the behavior so that pressing the Enter key moves the active cell to the next row.

C#

```
private void Form1_Load(object sender, System.EventArgs e)  
{  
    FarPoint.Win.Spread.InputMap im = new FarPoint.Win.Spread.InputMap();  
  
    // Define the operation of pressing Enter key in cells not being edited as "Move to  
the next row".  
    im = fpSpread1.GetInputMap(FarPoint.Win.Spread.InputMapMode.WhenFocused);  
    im.Put(new FarPoint.Win.Spread.Keystroke(Keys.Enter, Keys.None),  
FarPoint.Win.Spread.SpreadActions.MoveToNextRow);  
  
    // Define the operation of pressing Enter key in cells being edited as "Move to the  
next row".  
    im = fpSpread1.GetInputMap(FarPoint.Win.Spread.InputMapMode.WhenAncestorOfFocused);  
    im.Put(new FarPoint.Win.Spread.Keystroke(Keys.Enter, Keys.None),  
FarPoint.Win.Spread.SpreadActions.MoveToNextRow);  
}
```

VB

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles  
MyBase.Load  
    Dim im As New FarPoint.Win.Spread.InputMap  
  
    ' Define the operation of pressing Enter key in cells not being edited as "Move to  
the next row".  
    im = fpSpread1.GetInputMap(FarPoint.Win.Spread.InputMapMode.WhenFocused)  
    im.Put(New FarPoint.Win.Spread.Keystroke(Keys.Enter, Keys.None),  
FarPoint.Win.Spread.SpreadActions.MoveToNextRow)  
  
    ' Define the operation of pressing Enter key in cells being edited as "Move to the  
next row".  
    im = fpSpread1.GetInputMap(FarPoint.Win.Spread.InputMapMode.WhenAncestorOfFocused)  
    im.Put(New FarPoint.Win.Spread.Keystroke(Keys.Enter, Keys.None),  
FarPoint.Win.Spread.SpreadActions.MoveToNextRow)  
End Sub
```

Example

This example deactivates the F2 key.

C#

```
private void Form1_Load(object sender, System.EventArgs e)
{
    FarPoint.Win.Spread.InputMap im = new FarPoint.Win.Spread.InputMap();

    // Deactivate F2 key in cells not being edited.
    im = fpSpread1.GetInputMap(FarPoint.Win.Spread.InputMapMode.WhenFocused);
    im.Put(new FarPoint.Win.Spread.Keystroke(Keys.F2, Keys.None),
    FarPoint.Win.Spread.SpreadActions.None);

    // Deactivate F2 key in cells being edited.
    im = fpSpread1.GetInputMap(FarPoint.Win.Spread.InputMapMode.WhenAncestorOfFocused);
    im.Put(new FarPoint.Win.Spread.Keystroke(Keys.F2, Keys.None),
    FarPoint.Win.Spread.SpreadActions.None);
}
```

VB

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
MyBase.Load
    Dim im As New FarPoint.Win.Spread.InputMap
    ' Deactivate F2 key in cells not being edited.
    im = fpSpread1.GetInputMap(FarPoint.Win.Spread.InputMapMode.WhenFocused)
    im.Put(New FarPoint.Win.Spread.Keystroke(Keys.F2, Keys.None),
    FarPoint.Win.Spread.SpreadActions.None)
    ' Deactivate F2 key in cells being edited.
    im = fpSpread1.GetInputMap(FarPoint.Win.Spread.InputMapMode.WhenAncestorOfFocused)
    im.Put(New FarPoint.Win.Spread.Keystroke(Keys.F2, Keys.None),
    FarPoint.Win.Spread.SpreadActions.None)
End Sub
```

Using Input Maps with Action Maps

Internally, the **SpreadView** object uses an input map paired with an action map to process a keystroke. An input map (**InputMap** ('**InputMap Class**' in the on-line documentation) object) is used to convert a key stroke (**KeyStroke** ('**Keystroke Structure**' in the on-line documentation) object) to an object that identifies the action. An action map (**ActionMap** ('**ActionMap Class**' in the on-line documentation) object) is used to convert the object to an action.

Using Code

1. Create a new **InputMap** ('**InputMap Class**' in the on-line documentation) object for which to map keys and actions.
2. Set an existing input map equal to the **InputMap** ('**InputMap Class**' in the on-line documentation) object you created.
3. Use the **InputMap** ('**InputMap Class**' in the on-line documentation) class **Put** method to map specific keys to specific actions.

Example

For example, the internal code that handles KeyDown events looks something like this:

C#

```
object actionMapKey = GetInputMap(InputMapMode.WhenFocused).Get(new Keystroke(e.KeyCode,
e.Modifiers));
if (actionMapKey != null)
{
    Action action = GetActionMap().Get(actionMapKey);
    if (action != null)
    {
        action.PerformAction(this);
        e.Handled = true;
    }
}
}
```

VB

```
Dim actionMapKey As Object = GetInputMap(InputMapMode.WhenFocused).Get(New
Keystroke(e.KeyCode, e.Modifiers))
If Not (actionMapKey Is Nothing) Then
    Dim action As Action = GetActionMap().Get(actionMapKey)
    If Not (action Is Nothing) Then
        action.PerformAction(Me)
        e.Handled = True
    End If
End If
```

Excel uses Ctl+9 and Ctl+Shift+9 to hide and unhide rows. Suppose you want to implement this feature in Spread. You could create the following classes to define the hide and unhide actions.

C#

```
private class HideRowAction : Action
{
    public override void PerformAction(object source)
    {
        if (source is SpreadView)
        {
            SpreadView spread = (SpreadView)source;
            SheetView sheet = spread.Sheets[spread.ActiveSheetIndex];
            if (sheet.SelectionCount > 0)
            {
                for (int i = 0; i < sheet.SelectionCount; i++)
                {
                    CellRange range = sheet.GetSelection(i);
                    if (range.Row == -1)
                        sheet.Rows[0, sheet.RowCount - 1].Visible = false;
                    else
                        sheet.Rows[range.Row, range.Row + range.RowCount - 1].Visible = false;
                }
            }
        }
        else
        {
            sheet.Rows[sheet.ActiveRowIndex].Visible = false;
        }
    }
}
```

```

}

private class UnhideRowAction : Action
{
    public override void PerformAction(object source)
    {
        if (source is SpreadView)
        {
            SpreadView spread = (SpreadView)source;
            SheetView sheet = spread.Sheets[spread.ActiveSheetIndex];
            if (sheet.SelectionCount > 0)
            {
                for (int i = 0; i < sheet.SelectionCount; i++)
                {
                    CellRange range = sheet.GetSelection(i);
                    if (range.Row == -1)
                        sheet.Rows[0, sheet.RowCount - 1].Visible = true;
                    else
                        sheet.Rows[range.Row, range.Row + range.RowCount - 1].Visible = true;
                }
            }
            else
            {
                sheet.Rows[sheet.ActiveRowIndex].Visible = true;
            }
        }
    }
}

```

VB

```

Private Class HideRowAction
    Inherits Action

    Public Overrides Sub PerformAction([source] As Object)
        If TypeOf [source] Is SpreadView Then
            Dim spread As SpreadView = CType([source], SpreadView)
            Dim sheet As SheetView = spread.Sheets(spread.ActiveSheetIndex)
            If sheet.SelectionCount > 0 Then
                Dim i As Integer
                For i = 0 To sheet.SelectionCount - 1
                    Dim range As CellRange = sheet.GetSelection(i)
                    If range.Row = - 1 Then
                        sheet.Rows(0, sheet.RowCount - 1).Visible = False
                    Else
                        sheet.Rows(range.Row, range.Row + range.RowCount - 1).Visible = False
                    End If
                Next i
            Else
                sheet.Rows(sheet.ActiveRowIndex).Visible = False
            End If
        End If
    End Sub 'PerformAction
End Class 'HideRowAction

Private Class UnhideRowAction
    Inherits Action

```

```

Public Overrides Sub PerformAction([source] As Object)
  If TypeOf [source] Is SpreadView Then
    Dim spread As SpreadView = CType([source], SpreadView)
    Dim sheet As SheetView = spread.Sheets(spread.ActiveSheetIndex)
    If sheet.SelectionCount > 0 Then
      Dim i As Integer
      For i = 0 To sheet.SelectionCount - 1
        Dim range As CellRange = sheet.GetSelection(i)
        If range.Row = - 1 Then
          sheet.Rows(0, sheet.RowCount - 1).Visible = True
        Else
          sheet.Rows(range.Row, range.Row + range.RowCount - 1).Visible = True
        End If
      Next i
    Else
      sheet.Rows(sheet.ActiveRowIndex).Visible = True
    End If
  End If
End Sub 'PerformAction
End Class 'UnhideRowAction

```

You could then add the keystrokes and actions to the maps as follows.

C#

```

InputMap im = spread.GetInputMap(InputMapMode.WhenFocused);
ActionMap am = spread.GetActionMap();
im.Put(new Keystroke(Keys.D9, Keys.Control), "HideRow");
im.Put(new Keystroke(Keys.D9, Keys.Control | Keys.Shift), "UnhideRow");
am.Put("HideRow", new HideRowAction());
am.Put("UnhideRow", new UnhideRowAction());

```

VB

```

Dim im As InputMap = spread.GetInputMap(InputMapMode.WhenFocused)
Dim am As ActionMap = spread.GetActionMap()
im.Put(New Keystroke(Keys.D9, Keys.Control), "HideRow")
im.Put(New Keystroke(Keys.D9, Keys.Control Or Keys.Shift), "UnhideRow")
am.Put("HideRow", New HideRowAction())
am.Put("UnhideRow", New UnhideRowAction())

```

For more information, refer to the methods in **ActionMap ('ActionMap Class' in the on-line documentation)** and **InputMap ('InputMap Class' in the on-line documentation)** classes.

C#

```

public class FpSpread : ...{
    ...
    public ActionMap GetActionMap();
    public void SetActionMap(ActionMap value);
}

public class SpreadView : ...
{
    ...
    public ActionMap GetActionMap();
    public void SetActionMap(ActionMap value);
}

```

```

public class ActionMap{
public ActionMap();
public object[] AllKeys();
public object[] Keys();
public ActionMap Parent { get; set; }
public int Size { get; }
public void Clear();
public Action Get(object key);
public void Put(object key, Action action);
public void Remove(object key);}
public abstract class Action{ public abstract void PerformAction(object source);}

```

VB

```

Public Class FpSpread ...
...
Public Function GetActionMap() As ActionMap

Public Sub SetActionMap(value As ActionMap)

End Class 'FpSpread

Public Class SpreadView ...
...
Public Function GetActionMap() As ActionMap

Public Sub SetActionMap(value As ActionMap)

End Class 'SpreadView

Public Class ActionMap

Public Sub ActionMap()
Public object AllKeys()
Public object Keys()
Public ActionMap Parent
Public Size As Integer
Public Sub Clear()
Public Action Get(By key As object)
Public Sub Put(By key As object, By action As Action)
Public Sub Remove(By key As object)
}

Public MustOverride Class Action
{
Public Sub PerformAction(By source As object)
}

```

Customizing the Input Maps

You can use the default navigation keys that are mapped to Spread actions. You can also customize these input maps so that any key or key combination can map to any Spread component action.

Be sure to read **Factors of Keyboard Map Usage**.

The available actions to which you can map keys or key combinations are provided in the **SpreadActions** (**'SpreadActions Class' in the on-line documentation**) class.

The default maps create the default navigation and action keys listed in **Default Keyboard Maps**. These maps are for the default operation mode, Normal. Other maps are provided for other operation modes. Be sure to set the map for the operation mode in which your component is working when customizing input maps. For more information about default values of the various modes, refer to **Default Keyboard Maps**. Input maps work differently depending on the current focus.

For more detailed descriptions of input maps, refer to the **InputMap ('InputMap Class' in the on-line documentation)** class.

The **Properties** window does not have options to customize input maps.

Using Code

1. Create a new **InputMap ('InputMap Class' in the on-line documentation)** object for which to map keys and actions.
2. Set an existing input map equal to the **InputMap ('InputMap Class' in the on-line documentation)** object you created.
3. Use the **InputMap ('InputMap Class' in the on-line documentation)** class **Put** method to map specific keys to specific actions.

Example

This example code sets the input map for operation mode Normal for both the object with focus and its ancestor. This example sets the Enter key to always move to the next row.

C#

```
// Create an InputMap object.
FarPoint.Win.Spread.InputMap inputmap1;
// Assign the InputMap object to the existing map.
inputmap1 =
fpSpread1.GetInputMap(FarPoint.Win.Spread.InputMapMode.WhenAncestorOfFocused);
// Map the Enter key.
inputmap1.Put(new FarPoint.Win.Spread.Keystroke(Keys.Enter, Keys.None),
FarPoint.Win.Spread.SpreadActions.MoveToNextRow);
// Create another InputMap object.
FarPoint.Win.Spread.InputMap inputmap2;
// Assign this InputMap object to the existing map.
inputmap2 = fpSpread1.GetInputMap(FarPoint.Win.Spread.InputMapMode.WhenFocused);
// Map the Enter key.
inputmap2.Put(new FarPoint.Win.Spread.Keystroke(Keys.Enter, Keys.None),
FarPoint.Win.Spread.SpreadActions.MoveToNextRow);
```

VB

```
' Create an InputMap object.
Dim inputmap1 As New FarPoint.Win.Spread.InputMap()
' Assign the InputMap object to the existing map.
inputmap1 =
fpSpread1.GetInputMap(FarPoint.Win.Spread.InputMapMode.WhenAncestorOfFocused)
' Map the Enter key.
inputmap1.Put(New FarPoint.Win.Spread.Keystroke(Keys.Enter, Keys.None),
FarPoint.Win.Spread.SpreadActions.MoveToNextRow)
' Create another InputMap object.
Dim inputmap2 As New FarPoint.Win.Spread.InputMap()
' Assign this InputMap object to the existing map.
inputmap2 = fpSpread1.GetInputMap(FarPoint.Win.Spread.InputMapMode.WhenFocused)
```

```
' Map the Enter key.
inputmap2.Put(New FarPoint.Win.Spread.Keystroke(Keys.Enter, Keys.None),
FarPoint.Win.Spread.SpreadActions.MoveToNextRow)
```

Using the Spread Designer

1. In the **Settings** menu, click the **Input Map** icon under the **Other Settings** section.
2. Select the input map, key, and action.
3. Add the new key and action.
4. Select **OK**.
5. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Changing an Input Map for a Child View

Normally, when you change an input map definition, it applies only to inputs in the active sheet. For a hierarchical display, this only applies to the parent sheet and not to any expanded child sheets of the hierarchy. Spread treats the parent and each child as different workbooks. If you want to change input maps for child parts of a hierarchy, you need to implement the **ChildWorkbookCreated ('ChildWorkbookCreated Event' in the on-line documentation)** event which occurs when the child workbooks are created. Then you can change the input maps for those child workbooks.

	Column1	Column2
1	Parent1	0
	Column1	Column2
1	Child1-1	0
2	Child1-2	0
3	Child1-3	0
2	Parent2	1
	Column1	Column2
1	Child2-1	1
2	Child2-2	1
3	Child2-3	1

Using Code

1. Create an **InputMap ('InputMap Class' in the on-line documentation)** object.
2. Use the **GetInputMap ('GetInputMap Method' in the on-line documentation)** method.
3. Use the **Put ('Put Method' in the on-line documentation)** method.
4. Create a data set.

Example

This example shows how to change an input map for a parent, then expand a hierarchical display and change the input map for a child of that hierarchy.

C#

```
private void Form1_Load(object sender, System.EventArgs e) {
```

```

// Change input maps for Enter key in parent hierarchies.
FarPoint.Win.Spread.InputMap im = new FarPoint.Win.Spread.InputMap();
im = fpSpread1.GetInputMap(FarPoint.Win.Spread.InputMapMode.WhenFocused);
im.Put(new FarPoint.Win.Spread.KeyStroke(Keys.Enter, Keys.None),
FarPoint.Win.Spread.SpreadActions.MoveToNextRow);
im = fpSpread1.GetInputMap(FarPoint.Win.Spread.InputMapMode.WhenAncestorOfFocused);
im.Put(new FarPoint.Win.Spread.KeyStroke(Keys.Enter, Keys.None),
FarPoint.Win.Spread.SpreadActions.MoveToNextRow);

DataSet ds = new DataSet();
DataTable fpParent = new DataTable();
DataTable fpChild1 = new DataTable();

fpParent = ds.Tables.Add("SAMPLE");
fpParent.Columns.AddRange(new DataColumn[] {new DataColumn("Column1",
Type.GetType("System.String")), new DataColumn("Column2",
Type.GetType("System.Int32"))});
fpParent.Rows.Add(new object[] {"Parent1", 0});
fpParent.Rows.Add(new object[] {"Parent2", 1});

fpChild1 = ds.Tables.Add("Child1");
fpChild1.Columns.AddRange(new DataColumn[] {new
DataColumn("Column1", Type.GetType("System.String")), new DataColumn("Column2",
Type.GetType("System.Int32"))});
fpChild1.Rows.Add(new object[] {"Child1-1", 0});
fpChild1.Rows.Add(new object[] {"Child1-2", 0});
fpChild1.Rows.Add(new object[] {"Child1-3", 0});
fpChild1.Rows.Add(new object[] {"Child2-1", 1});
fpChild1.Rows.Add(new object[] {"Child2-2", 1});
fpChild1.Rows.Add(new object[] {"Child2-3", 1});

ds.Relations.Add("Relation1", fpParent.Columns["Column2"],
fpChild1.Columns["Column2"]);
fpSpread1.ActiveSheet.DataSource = ds;

// Expand child hierarchies.
fpSpread1.ActiveSheet.ExpandRow(0, true);
fpSpread1.ActiveSheet.ExpandRow(1, true);
}
private void fpSpread1_ChildWorkbookCreated(object sender,
FarPoint.Win.Spread.ChildWorkbookCreatedEventArgs e)
{
// Change its input map for Enter key in child hierarchies (e.Workbook: the target
is child SpreadView).
FarPoint.Win.Spread.InputMap im = new FarPoint.Win.Spread.InputMap();

im = e.Workbook.GetInputMap(FarPoint.Win.Spread.InputMapMode.WhenFocused);
im.Put(new FarPoint.Win.Spread.KeyStroke(Keys.Enter, Keys.None),
FarPoint.Win.Spread.SpreadActions.MoveToNextRow);
im = e.Workbook.GetInputMap(FarPoint.Win.Spread.InputMapMode.WhenAncestorOfFocused);
im.Put(new FarPoint.Win.Spread.KeyStroke(Keys.Enter, Keys.None),
FarPoint.Win.Spread.SpreadActions.MoveToNextRow);
}

```

VB

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
```

```
MyBase.Load
```

```

    ' Change input maps for Enter key in parent hierarchies.
    Dim im As New FarPoint.Win.Spread.InputMap
    im = fpSpread1.GetInputMap(FarPoint.Win.Spread.InputMapMode.WhenFocused)
    im.Put(New FarPoint.Win.Spread.Keystroke(Keys.Enter, Keys.None),
    FarPoint.Win.Spread.SpreadActions.MoveToNextRow)
    im = fpSpread1.GetInputMap(FarPoint.Win.Spread.InputMapMode.WhenAncestorOfFocused)
    im.Put(New FarPoint.Win.Spread.Keystroke(Keys.Enter, Keys.None),
    FarPoint.Win.Spread.SpreadActions.MoveToNextRow)

    Dim ds As New DataSet
    Dim fpParent As DataTable
    Dim fpChild1 As DataTable
    fpParent = ds.Tables.Add("SAMPLE")
    fpParent.Columns.AddRange(New DataColumn() {New DataColumn("Column1",
    Type.GetType("System.String")), New DataColumn("Column2",
    Type.GetType("System.Int32"))})
    fpParent.Rows.Add(New Object() {"Parent1", 0})
    fpParent.Rows.Add(New Object() {"Parent2", 1})

    fpChild1 = ds.Tables.Add("Child1")
    fpChild1.Columns.AddRange(New DataColumn() {New DataColumn("Column1",
    Type.GetType("System.String")), New DataColumn("Column2",
    Type.GetType("System.Int32"))})
    fpChild1.Rows.Add(New Object() {"Child1-1", 0})
    fpChild1.Rows.Add(New Object() {"Child1-2", 0})
    fpChild1.Rows.Add(New Object() {"Child1-3", 0})
    fpChild1.Rows.Add(New Object() {"Child2-1", 1})
    fpChild1.Rows.Add(New Object() {"Child2-2", 1})
    fpChild1.Rows.Add(New Object() {"Child2-3", 1})

    ds.Relations.Add("Relation1", fpParent.Columns("Column2"),
    fpChild1.Columns("Column2"))
    fpSpread1.ActiveSheet.DataSource = ds

    ' Expand child hierarchies.
    fpSpread1.ActiveSheet.ExpandRow(0, True)
    fpSpread1.ActiveSheet.ExpandRow(1, True)
End Sub

Private Sub fpSpread1_ChildWorkbookCreated(ByVal sender As Object, ByVal e As
FarPoint.Win.Spread.ChildWorkbookCreatedEventArgs) Handles
FpSpread1.ChildWorkbookCreated
    ' Change its input map for Enter key in child hierarchies (e.Workbook: the target is
child SpreadView).
    Dim im As New FarPoint.Win.Spread.InputMap
    im = e.Workbook.GetInputMap(FarPoint.Win.Spread.InputMapMode.WhenFocused)
    im.Put(New FarPoint.Win.Spread.Keystroke(Keys.Enter, Keys.None),
    FarPoint.Win.Spread.SpreadActions.MoveToNextRow)
    im = e.Workbook.GetInputMap(FarPoint.Win.Spread.InputMapMode.WhenAncestorOfFocused)
    im.Put(New FarPoint.Win.Spread.Keystroke(Keys.Enter, Keys.None),
    FarPoint.Win.Spread.SpreadActions.MoveToNextRow)
End Sub

```

Saving and Loading Map Files

You can load or save the input maps to an XML file. Use the **SaveXmlInputMapFile ('SaveXmlInputMapFile Method' in the on-line documentation)** method to save to a file and the **LoadXmlInputMapFile ('LoadXmlInputMapFile Method' in the on-line documentation)** to load from an input map file.

Spread installs a default Excel compatibility XML file to the input map folder.

Using Code

1. Create an input map.
2. Save the input map to a file.

Example

This example saves an input map to an XML file.

C#

```
FarPoint.Win.Spread.SpreadView sv = fpSpread1.GetRootWorkbook();
    FarPoint.Win.Spread.InputMap im = new FarPoint.Win.Spread.InputMap();
    im.Put(new FarPoint.Win.Spread.KeyStroke(Keys.Enter, Keys.None),
FarPoint.Win.Spread.SpreadActions.MoveToNextRow);
    sv.SetInputMap(FarPoint.Win.Spread.InputMapMode.WhenAncestorOfFocused,
im);
        fpSpread1.SaveXmlInputMapFile("file.xml",
FarPoint.Win.Spread.InputMapMode.WhenAncestorOfFocused,
FarPoint.Win.Spread.OperationMode.Normal);
```

VB

```
Dim sv As FarPoint.Win.Spread.SpreadView = fpSpread1.GetRootWorkbook
    Dim im As New FarPoint.Win.Spread.InputMap()
    im.Put(New FarPoint.Win.Spread.KeyStroke(Keys.Enter, Keys.None),
FarPoint.Win.Spread.SpreadActions.MoveToNextRow)
    sv.SetInputMap(FarPoint.Win.Spread.InputMapMode.WhenAncestorOfFocused,
im)
        fpSpread1.SaveXmlInputMapFile("file.xml",
FarPoint.Win.Spread.InputMapMode.WhenAncestorOfFocused,
FarPoint.Win.Spread.OperationMode.Normal)
```

Using the Spread Designer

1. Select the **Settings** menu and then select the **Input Map** icon located under the **Other Settings** section.
2. From the **Select InputMap** section, select the mode options.
3. Specify the shortcut key and action.
4. Click **Add** to add the new input map.
5. Use the **Save** button to save the input map to a file. Use the **Load Input Maps** button to load an input map file instead.
6. Select **OK** to apply the input map.
7. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Using Excel-compatible Keyboard Shortcuts

By using keyboard shortcuts, you can save your time and increase efficiency. Spread for WinForms provides **ExcelCompatibleKeyboardShortcuts** property which when set to true, allows you to use Excel-compatible keyboard shortcuts.

 **Note:**

- The supported shortcuts and their behavior when **ExcelCompatibleKeyboardShortcuts** is true, may change in next releases to match with Excel behavior.
- If **ExcelCompatibleKeyboardShortcuts** is false, some of the shortcut keys are available on Spread Designer as well.

The below table lists the Excel-compatible keyboard shortcuts:

Shortcut	Spread Designer Support	Description
CTRL + -	No	Delete cells
CTRL + F3	Yes	Show Name Manager dialog
SHIFT + F9	Yes	Calculate the active sheet
F9	Yes	Calculate the entire workbook
CTRL + SHIFT + T	Yes	Show/hide total row of the table
CTRL + T	Yes	Create table
CTRL + U	Yes	Format text as underline
CTRL + I	Yes	Format text as italic
CTRL + B	Yes	Format text as bold
CTRL + SHIFT + %	Yes	Format as a Percent
CTRL + SHIFT + V	No	Paste plain text or paste values
ALT + SHIFT + RIGHT Arrow	Yes	Group data
ALT + SHIFT + LEFT Arrow	Yes	Ungroup data
SHIFT + F2	No	Create cell comment (EnhancedShapeEngine = True)
ALT + UP Arrow	No	Show Data Validation auto complete list when cell in editmode
SHIFT + F3	Yes	Show Insert Function / Function Arguments dialog
F3	No	Show Paste Name dialog
Ctrl+ `	No	Toggle between displaying formulas in cells and displaying calculated values as in Excel
Ctrl+1	No	Show "Format Cells" dialog

Ctrl+Shift+F5	No	Show card (for .NET data object)
Ctrl+K	No	Show Insert Hyperlink dialog
Ctrl+Enter	No	Input data for multiple cells
Tab	No	Move to the next cell at the right In case of a table, if active cell is in the last column of table <ul style="list-style-type: none"> • If the last row of table is selected, add new data row and move active cell to the first cell of new row • Otherwise, move to the first cell of the next row (or the first cell of the first row if current row is the last)
Shift + Tab	No	Move the active cell to the previous cell at the left If the active cell is in the first column of table, move to the last cell of previous row (or the last cell of the last row if current row is the first)
Ctrl + A	No	Select neighbor cells, then worksheet on each subsequent use of Ctrl+A If the cell belongs to Table, select table data range, then table range, then worksheet on each subsequent use of Ctrl+A
Ctrl + Shift + "+"	No	Insert cut or copied cells
Ctrl + Page Up	No	Navigate to the next sheet.
Ctrl + Page Down	No	Navigate to the previous sheet.

The below example code sets **ExcelCompatibleKeyboardShortcuts** property to true which allows you to use above mentioned shortcuts.

C#

```
// Set ExcelCompatibleKeyboardShortcuts features to true
fpSpread1.Features.ExcelCompatibleKeyboardShortcuts = true;
// Set value in Cells
fpSpread1.ActiveSheet.Cells["A1"].Value = 10;
fpSpread1.ActiveSheet.Cells["B1"].Value = 20;
// Set formula in Cell
fpSpread1.ActiveSheet.Cells["C1"].Formula = "SUM(A1: B2)";
```

VB

```
' Set ExcelCompatibleKeyboardShortcuts features to true
fpSpread1.Features.ExcelCompatibleKeyboardShortcuts = True
' Set value in Cells
fpSpread1.ActiveSheet.Cells("A1").Value = 10
fpSpread1.ActiveSheet.Cells("B1").Value = 20
' Set formula in Cell
fpSpread1.ActiveSheet.Cells("C1").Formula = "SUM(A1: B2) "
```

Using the Excel Compatibility Input Maps

You can specify whether to use the default Spread input maps or the Excel compatibility input maps. You can also specify the input map mode and the operation mode. Some actions may not apply to certain modes such as starting cell editing with read-only mode.

Use the **LoadXmlInputMapFile ('LoadXmlInputMapFile Method' in the on-line documentation)** method to load the Excel compatibility file or other input map file. Spread installs a default Excel compatibility file to the product bin folder.

Using Code

Set the options in the **LoadXmlInputMapFile ('LoadXmlInputMapFile Method' in the on-line documentation)** method.

Example

This example specifies the Excel compatibility option.

C#

```
fpSpread1.LoadXmlInputMapFile("ExcelCompatibility.imp");
//fpSpread1.LoadXmlInputMapFile("ExcelCompatibility.imp",
FarPoint.Win.Spread.InputMapMode.WhenAncestorOfFocused,
FarPoint.Win.Spread.OperationMode.Normal);
```

VB

```
fpSpread1.LoadXmlInputMapFile("ExcelCompatibility.imp")
'fpSpread1.LoadXmlInputMapFile("ExcelCompatibility.imp",
FarPoint.Win.Spread.InputMapMode.WhenAncestorOfFocused,
FarPoint.Win.Spread.OperationMode.Normal)
```

Using the Spread Designer

1. Select the **Settings** menu and then select the **Input Map** icon located under the **Other Settings** section.
2. Click the **Load Input Maps** button to load the Excel compatibility file.
3. Select **OK**.
4. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Default Map for Excel Compatibility

The default map for Excel compatibility mode is summarized in this table. This table applies when the input map mode is **WhenAncestorOfFocused** and the state is normal or edit mode.

Key Code	Action Name
Delete	ClearSelectedCellsData ('ClearSelectedCellsData Field' in the on-line documentation)
Ctrl + Up Arrow	MoveToPreviousRowWithData ('MoveToPreviousRowWithData Field' in the on-line documentation)
Ctrl + Down Arrow	MoveToNextRowWithData ('MoveToNextRowWithData Field' in the on-line documentation)

Ctrl + Left Arrow	MoveToPreviousColumnWithData ('MoveToPreviousColumnWithData Field' in the on-line documentation)
Ctrl + Right Arrow	MoveToNextColumnWithData ('MoveToNextColumnWithData Field' in the on-line documentation)
Ctrl + Shift + Up Arrow	ExtendToPreviousRowWithData ('ExtendToPreviousRowWithData Field' in the on-line documentation)
Ctrl + Shift + Down Arrow	ExtendToNextRowWithData ('ExtendToNextRowWithData Field' in the on-line documentation)
Ctrl + Shift + Left Arrow	ExtendToPreviousColumnWithData ('ExtendToPreviousColumnWithData Field' in the on-line documentation)
Ctrl + Shift + Right Arrow	ExtendToNextColumnWithData ('ExtendToNextColumnWithData Field' in the on-line documentation)
F2	StartEditing ('StartEditing Field' in the on-line documentation)
F4	Redo ('Redo Field' in the on-line documentation)
Enter	MoveToNextRow ('MoveToNextRow Field' in the on-line documentation)
Shift + Enter	MoveToPreviousRow ('MoveToPreviousRow Field' in the on-line documentation)
Ctrl + Page Up	MoveToPreviousSheet ('MoveToPreviousSheet Field' in the on-line documentation)
Ctrl + Page Down	MoveToNextSheet ('MoveToNextSheet Field' in the on-line documentation)
Alt + Page Up	MoveToPreviousPageOfColumns ('MoveToPreviousPageOfColumns Field' in the on-line documentation)
Alt + Page Down	MoveToNextPageOfColumns ('MoveToNextPageOfColumns Field' in the on-line documentation)
Ctrl + ;	DateTimeNow ('DateTimeNow Field' in the on-line documentation)
Tab	MoveToNextColumnVisual ('MoveToNextColumnVisual Field' in the on-line documentation)
Shift + Tab	MoveToPreviousColumnVisual ('MoveToPreviousColumnVisual Field' in the on-line documentation)
Ctrl + A	SelectSheet ('SelectSheet Field' in the on-line documentation)
Alt + Backspace	Undo ('Undo Field' in the on-line documentation)

The following table applies when the input map mode is **WhenFocused** and the state is normal.

Key Code	Action Name
Ctrl + C	ClipboardCopy ('ClipboardCopy Field' in the on-line documentation)
Ctrl + Insert	ClipboardCopy ('ClipboardCopy Field' in the on-line documentation)
Ctrl + X	ClipboardCut ('ClipboardCut Field' in the on-line documentation)
Shift + Delete	ClipboardCut ('ClipboardCut Field' in the on-line documentation)
Ctrl + V	ClipboardPasteAll ('ClipboardPasteAll Field' in the on-line documentation)
Shift + Insert	ClipboardPasteAll ('ClipboardPasteAll Field' in the on-line documentation)
Backspace	StartEditing ('StartEditing Field' in the on-line documentation)
=	StartEditingFormula ('StartEditingFormula Field' in the on-line documentation)

- Ctrl + Z **Undo ('Undo Field' in the on-line documentation)**
- Ctrl + Y **Redo ('Redo Field' in the on-line documentation)**

Events from User Actions

This topic summarizes which events are raised for each user action on the Spread component. While it is not a comprehensive list of every action that the user could possibly perform, it details the events for most of the common actions performed by the user.

The types of user actions are organized as follows:

- **Clicking Actions**
- **Selecting Actions**
- **Entering Data Actions**
- **Sheet-Level Actions**
- **Interactivity Actions**
- **Shape Actions**
- **Print Actions**

The lists provided in these sections are an attempt to illustrate the sequence of events that are raised for typical actions. Since some actions occur all the time or repeatedly for any of these actions, we have left off some of these actions from the lists. For example, these lists do not include **MouseMove**, **MouseHover**, **MouseEnter**, **MouseLeave**, **Invalidated**, and **CursorChanged** events.

In general, if you are looking for a way to intercept each change that occurs in a cell, the **EditChange** event is raised for each keystroke the user makes as they are typing data into a cell.

Clicking Actions

Clicking, double-clicking, and right-clicking actions in Spread result in these events:

User Action	List of Events
Click on a general cell	<ul style="list-style-type: none"> • MouseDown • Enter • GotFocus • CellClick ('CellClick Event' in the on-line documentation) • LeaveCell ('LeaveCell Event' in the on-line documentation) • EnterCell ('EnterCell Event' in the on-line documentation) • Paint • MouseUp • MouseCaptureChanged • SelectionChanged ('SelectionChanged Event' in the on-line documentation) • Paint
Click on a combo box cell and select an item	<ul style="list-style-type: none"> • ComboDropDown ('ComboDropDown Event' in the on-line documentation) • ComboSelChange ('ComboSelChange Event' in the on-line documentation) • EditChange ('EditChange Event' in the on-line documentation) • ComboCloseUp ('ComboCloseUp Event' in the on-line documentation) • Paint
Click on a multiple option cell	<ul style="list-style-type: none"> • MouseDown

and select an option

- **CellClick ('CellClick Event' in the on-line documentation)**
- **LeaveCell ('LeaveCell Event' in the on-line documentation)**
- **EnterCell ('EnterCell Event' in the on-line documentation)**
- **EditModeStarting ('EditModeStarting Event' in the on-line documentation)**
- MouseCaptureChanged
- ControlAdded
- **EditModeOn ('EditModeOn Event' in the on-line documentation)**
- LostFocus
- Paint
- **ButtonClicked ('ButtonClicked Event' in the on-line documentation)**

Double-click a general cell (go into edit mode)

- MouseDown
- **CellClick ('CellClick Event' in the on-line documentation)**
- **LeaveCell ('LeaveCell Event' in the on-line documentation)**
- **EnterCell ('EnterCell Event' in the on-line documentation)**
- Paint
- MouseUp
- MouseCaptureChanged
- **SelectionChanged ('SelectionChanged Event' in the on-line documentation)**
- Paint
- MouseDown
- **CellDoubleClick ('CellDoubleClick Event' in the on-line documentation)**
- **EditModeStarting ('EditModeStarting Event' in the on-line documentation)**
- MouseCaptureChanged
- Layout
- ControlAdded
- **EditModeOn ('EditModeOn Event' in the on-line documentation)**
- LostFocus
- Paint

Expanding a hierarchy (the first time)

- MouseDown
- **CellClick ('CellClick Event' in the on-line documentation)**
- **Expand ('Expand Event' in the on-line documentation)**
- Layout
- ControlAdded
- Layout
- **DataColumnConfigure ('DataColumnConfigure Event' in the on-line documentation)**
- **... DataColumnConfigure ('DataColumnConfigure Event' in the on-line documentation)** (repeat for each column in expanded row)
- Layout
- ControlAdded

- Layout
- ... Layout (repeat for each column in expanded row)
- ControlAdded
- Layout
- ... Layout (repeat for each expanded row)
- **ChildWorkbookCreated ('ChildWorkbookCreated Event' in the on-line documentation)**
- **ChildViewCreated ('ChildViewCreated Event' in the on-line documentation)**
- Layout
- ControlRemoved
- Layout
- ControlRemoved
- Layout
- ControlAdded
- Layout
- ... Layout (repeat for each expanded column)
- ControlAdded
- Layout
- Paint
- MouseUp
- MouseCaptureChanged
- Paint

Expanding a hierarchy
(subsequent times)

- MouseDown
- **CellClick ('CellClick Event' in the on-line documentation)**
- **Expand ('Expand Event' in the on-line documentation)**
- Layout
- ControlAdded
- Layout
- ... Layout (repeated)
- ControlAdded
- Layout
- Paint
- MouseUp
- MouseCaptureChanged
- Paint

Collapsing a hierarchy (the first
time or subsequent times)

- MouseDown
- **CellClick ('CellClick Event' in the on-line documentation)**
- **Expand ('Expand Event' in the on-line documentation)**
- Layout
- ControlRemoved
- Layout
- ControlRemoved
- Paint
- MouseUp
- MouseUp

- Paint

Selecting Actions

Actions such as selecting cells and working with selections in Spread result in these events:

User Action

Select a cell - click a general (default) cell

Select range of cells - click a general (default) cell and drag to another cell

Select a row (or column) - click on header cell

List of Events

- MouseDown
 - Enter
 - GotFocus
 - **CellClick ('CellClick Event' in the on-line documentation)**
 - MouseUp
 - MouseCaptureChanged
 - **SelectionChanged ('SelectionChanged Event' in the on-line documentation)**
 - Paint
-
- MouseDown
 - **CellClick ('CellClick Event' in the on-line documentation)**
 - **LeaveCell ('LeaveCell Event' in the on-line documentation)**
 - **EnterCell ('EnterCell Event' in the on-line documentation)**
 - Paint
 - **SelectionChanging ('SelectionChanging Event' in the on-line documentation)**
 - Paint
 - ... (repeating Paint every time you drag over to a cell in another row or column)
 - MouseUp
 - MouseCaptureChanged
 - **SelectionChanged ('SelectionChanged Event' in the on-line documentation)**
 - Paint
-
- MouseDown
 - Enter
 - GotFocus
 - **CellClick ('CellClick Event' in the on-line documentation)**
 - **LeaveCell ('LeaveCell Event' in the on-line documentation)**
 - **EnterCell ('EnterCell Event' in the on-line documentation)**
 - MouseUp
 - MouseCaptureChanged
 - **SelectionChanged ('SelectionChanged Event' in**

the on-line documentation)

- Paint

Entering Data Actions

Actions involved with entering data in Spread result in these events. These are just a few. You can also see what events occur when a formula is entered. Here are the events for entering a value:

User Action

List of Events

Enter a value in a cell

- (see events for click in a cell)
- **EditChange ('EditChange Event' in the on-line documentation)**
- (repeat EditChange for each key pressed)
- MouseDown
- **EditModeOff ('EditModeOff Event' in the on-line documentation)**
- Layout
- ControlRemoved
- **Change ('Change Event' in the on-line documentation)**
- GotFocus
- **CellClick ('CellClick Event' in the on-line documentation)**
- **LeaveCell ('LeaveCell Event' in the on-line documentation)**
- **EnterCell ('EnterCell Event' in the on-line documentation)**
- Paint
- MouseUp
- MouseCaptureChanged
- **SelectionChanged ('SelectionChanged Event' in the on-line documentation)**
- Paint
- LostFocus
- Leave
- Validating
- Validated

Sheet-Level Actions

Sheet-level actions in Spread result in these events:

Action

List of Events

Click on tab of new sheet

- MouseDown
- Enter
- GotFocus
- **SheetTabClick ('SheetTabClick Event' in the on-line documentation)**
- **ActiveSheetChanging ('ActiveSheetChanging Event' in the on-line documentation)**
- **ActiveSheetChanged ('ActiveSheetChanged Event' in the on-line documentation)**

	<ul style="list-style-type: none"> • Paint • MouseUp • MouseCaptureChanged • Paint
Click on the tab of the active sheet	<ul style="list-style-type: none"> • Paint
Remove a sheet	<ul style="list-style-type: none"> • Paint

Interactivity Actions

This table lists the events for the various actions involved with user interactivity in Spread. You can also see what events occur for sorting, filtering, grouping, and searching. Or creating a new viewport or moving a scroll bar by clicking the scroll bar button.

Action	List of Events
Resize a column - drag the column resize cursor	<ul style="list-style-type: none"> • MouseDown • Enter • GotFocus • MouseUp • MouseCaptureChanged • ColumnWidthChanged ('ColumnWidthChanged Event' in the on-line documentation) • Paint

Shape Actions

This table lists the events for the various shape actions in Spread. You can also see what events occur when a shape is deleted.

Action	List of Events
Move Shape	<ul style="list-style-type: none"> • MouseDown • ShapeActivated ('ShapeActivated Event' in the on-line documentation) • Paint • ... Paint (repeated for each cell moved through) • MouseUp • MouseCaptureChanged
Rotate Shape	<ul style="list-style-type: none"> • MouseDown • Paint • ... Paint (repeated for each cell moved through) • MouseUp • MouseCaptureChanged

Print Actions

This table lists the events for the various printing actions in Spread:

Action	List of Events
---------------	-----------------------

Print Preview	<ul style="list-style-type: none"> • PrintMessageBox ('PrintMessageBox Event' in the on-line documentation) • PrintPreviewShowing ('PrintPreviewShowing Event' in the on-line documentation) • LostFocus • PrintBackground ('PrintBackground Event' in the on-line documentation) • PrintAbort ('PrintAbort Event' in the on-line documentation) • PrintMessageBox ('PrintMessageBox Event' in the on-line documentation)
Print a sheet	<ul style="list-style-type: none"> • PrintMessageBox ('PrintMessageBox Event' in the on-line documentation) • PrintMessageBox ('PrintMessageBox Event' in the on-line documentation) • PrintBackground ('PrintBackground Event' in the on-line documentation) • PrintAbort ('PrintAbort Event' in the on-line documentation) • PrintMessageBox ('PrintMessageBox Event' in the on-line documentation) • GotFocus • Paint

File Operations

You can save data from Spread into several different file types and open data files from several different file types into Spread. At design time, you can use the Spread Designer to save the Spread to any of various file types or open previously saved files. With code, you can save the whole component, a particular sheet, or data from a particular range of cells to several different file types or streams. Similarly, you can allow your users to handle file operations for a range of file types.

The procedures for managing file operations include:

- **Saving Data to a File**
- **Opening Existing Files**
- **Using Serialization**
- **Saving and Loading a Skin**
- **Storing Excel Summary and View**

For information about saving and opening files in Spread Designer, refer to **Saving and Opening Design Files (on-line documentation)** in the Spread Designer Guide.

Saving Data to a File

You can save the data, and for some types of files the data and formatting, in the component to a file or stream. Spread provides methods for saving from a Spread file to several file types. Consult the following topics for instructions and more information regarding saving to a file.

- **Saving to a Spread XML File**
- **Saving to an Excel File**
- **Saving to a Text File**
- **Saving to an HTML Table**
- **Saving Spreadsheet Data to Simple XML**

Saving to a Spread XML File

You can save the data or the data and formatting in a Spread component to a Spread XML file or to a stream. All sheets in the component are saved to the file or stream if you use the **Save ('Save Method' in the on-line documentation)** method in the **FpSpread** class. If you choose to save the formatting, the saved data includes formatting characters, such as currency symbols, and other information such as cell types.

For more details on the methods used, refer to the **Save ('Save Method' in the on-line documentation)** methods in the **FpSpread ('FpSpread Class' in the on-line documentation)** class or the **SaveXml ('SaveXml Method' in the on-line documentation)** methods in the **SheetView ('SheetView Class' in the on-line documentation)** class.

For instructions for opening Spread-compatible XML files, see **Opening a Spread XML File**.

Using Code

Use the **Save ('Save Method' in the on-line documentation)** method of the **FpSpread ('FpSpread Class' in the on-line documentation)** component to specify the path and file name of the Spread XML file or the Stream object, and whether to save data only.

Example

This example code saves the data and formatting in a Spread component to a Spread XML file.

C#

```
// Save the data and formatting to an XML file.
fpSpread1.Save("C:\\SpWinFile1.xml", false);
```

VB

```
' Save the data and formatting to an XML file.
fpSpread1.Save("C:\\SpWinFile1.xml", False)
```

Using the Spread Designer

1. From the **File** menu, choose **Save**.
The **Save As** dialog appears.
2. Change the **Save as** type box to XML files (*.xml).
3. Specify the path and file name to which to save the file, and then click **Save**.
If the file is saved successfully, a message appears stating the file has been saved.
4. Click **OK** to close the Spread Designer.

Saving to an Excel File

You can save data to an excel file using one of the **SaveExcel ('SaveExcel Method' in the on-line documentation)** methods of the **FpSpread ('FpSpread Class' in the on-line documentation)** class. This method accepts the path and file name for the file to save, and any additional parameters depending on the particular method.

One of the parameters used in the SaveExcel method is the **ExcelSaveFlags ('ExcelSaveFlags Enumeration' in the on-line documentation)** enumeration. It provides the following formats to save data in Excel.

- Excel Workbook Format (.xlsx files)
- Excel 97-2003 BIFF8 Format (.xls files)

To save data to Excel Workbook format, use the **UseOOXMLFormat ('ExcelSaveFlags Enumeration' in the on-line documentation)** option. If you want to save the spreadsheet data along with all the spreadsheet settings to an Excel file, you need to use the **Exchangeable ('ExcelSaveFlags Enumeration' in the on-line documentation)** option as well. This option helps to retain all the data even after exporting and importing (with the **ExcelOpenFlags Exchangeable ('ExcelOpenFlags Enumeration' in the on-line documentation)** option) the Excel file in Spread.NET. By default when you save to Excel, whatever is stored in the data model of the Spread is written out to a file or stream in BIFF8 format.

It is also possible to store cell data from a single worksheet into an excel file using the **SaveAs** method of **ISheet** interface. This method allows you to specify the path and file name for the file to save, along with any additional parameters that may be necessary based on the method's requirements.

Refer to the following sample code that saves the cell data from a worksheet to an excel file.

C#

```
// Export worksheet to excel with options
TestActiveSheet.SaveAs("WorksheetToExcel.xlsx",
GrapeCity.Spreadsheet.IO.FileFormat.OpenXMLWorkbook, options: ExportOptions.RowHeader);
```

VB

```
' Export worksheet to excel with options
TestActiveSheet.SaveAs("WorksheetToExcel.xlsx",
GrapeCity.Spreadsheet.IO.FileFormat.OpenXMLWorkbook, options:=ExportOptions.RowHeader)
```

Export Image Cell Type to Excel

If you are saving a spreadsheet in an Excel file and it contains an image in an image cell type, the following behavior can be observed:

- If you specify the Exchangeable option, the image cell type will be saved to XLSX file.
 - If you open the exported file in Excel, an empty cell is displayed.
 - If you open the exported file in Spread without Exchangeable option, an empty cell is displayed
 - If you open the exported file in Spread with Exchangeable option, the image cell type cell is displayed.
- If you do not specify the Exchangeable option, the image cell type won't be saved to XLSX file and Spread will export a shape instead.
 - If you open the exported file in Excel or Spread (with or without Exchangeable option), an empty cell and a shape are displayed.

Example

The following example saves the spread data along with all the settings to an Excel file using the SaveExcel method which has ExcelSaveFlags options UseOOXMLFormat and Exchangeable as arguments.

C#

```
// Save spread data to an Excel file (.xlsx).
fpSpread1.SaveExcel("C:\\excelfile.xlsx", FarPoint.Excel.ExcelSaveFlags.UseOOXMLFormat
| FarPoint.Excel.ExcelSaveFlags.Exchangeable);
```

VB

```
' Save spread data to an Excel file (.xlsx).
fpSpread1.SaveExcel("C:\\excelfile.xlsx", FarPoint.Excel.ExcelSaveFlags.UseOOXMLFormat
| FarPoint.Excel.ExcelSaveFlags.Exchangeable)
```

 **Note:** If you put a number or date in an Excel cell and the width of the column is not large enough to display the data, then Excel shows the cell filled with ###. Make sure the width of the column is set wide enough to display the data in the exported Excel-formatted file.

Using the Spread Designer

1. From the **File** menu, choose **Save**.
The **Save As** dialog appears.
2. Change the **Save as** type box to Excel files (*.xls).
3. Specify the path and file name to which to save the file.
4. Select **Exchangeable** option and **UseOOXMLFormat** option from **Select Excel Save Flags** drop-down list, and then click **Save**.
If the file is saved successfully, a message appears stating the file has been saved.
5. Click **OK** to close the Spread Designer.

Example

This example code saves the data in a Spread component to an Excel-formatted file and specifies that both row and column headers are included in the output.

C#

```
// Save the data to an Excel-formatted file, including headers.
fpSpread1.SaveExcel("C:\\excelfile.xls",
FarPoint.Win.Spread.Model.IncludeHeaders.BothCustomOnly);
```

VB

```
' Save the data to an Excel-formatted file, including headers.
fpSpread1.SaveExcel("C:\\excelfile.xls",
FarPoint.Win.Spread.Model.IncludeHeaders.BothCustomOnly)
```

Document Caching

The **DocumentCaching** option in the **ExcelOpenFlags** (**'ExcelOpenFlags Enumeration' in the on-line documentation**) or **ExcelSaveFlags** (**'ExcelSaveFlags Enumeration' in the on-line documentation**) enumeration allows you to open, edit, and save without the loss of advanced document content and formatting. The content can be lossless only if the opening file format is similar to the saving file format. If the advanced document content uses files besides the xls(x) file, then the additional files need to be in the same folder with the xls(x) file. Advanced content could be macros, ActiveX controls, data connections, and so on.

For more information about how the data is saved to an Excel-formatted file, see the **Import and Export Reference (on-line documentation)**.

Saving to a Text File

You can save the data or the data and formatting in a sheet to a text file, using either default tab delimiters or custom delimiters (such as a comma for a csv file). In addition, you can specify whether headers are saved with the data.

Saving to a text file is done for individual sheets. If you want to save all the sheets in the component, you must save each sheet to a text file.

Tab-delimited files saved from the component contain data separated by tabs and carriage returns. Tab-delimited files can be opened, modified, and saved using any standard text editor. Delimited files contain data separated by user-defined delimiters, such as commas, quotation marks, or other delimiters.

You can save the entire sheet or a portion of the sheet data from the component to tab-delimited and delimited files.

If you upgraded from a version prior to version 5, then you may wish to replace any obsolete parameters in the **SaveTextFile** (**'SaveTextFile Method' in the on-line documentation**) and **SaveTextFileRange** (**'SaveTextFileRange Method' in the on-line documentation**) methods.

For more details on the methods and current parameters, refer to the **SaveTextFile** (**'SaveTextFile Method' in the on-line documentation**) and **SaveTextFileRange** (**'SaveTextFileRange Method' in the on-line documentation**) methods of the **SheetView** (**'SheetView Class' in the on-line documentation**) class.

For instructions for opening text files, see **Opening a Custom Text File**.

Using Code

1. To save the entire sheet, use one of the **SheetView** (**'SheetView Class' in the on-line documentation**) class **SaveTextFile** (**'SaveTextFile Method' in the on-line documentation**) methods, specifying the path and file name or stream, whether data or data and formatting is saved, whether headers are saved, and the custom delimiters, depending on the method you choose.
2. To save a portion of a sheet, use one of the **SheetView** (**'SheetView Class' in the on-line documentation**) class **SaveTextFileRange** (**'SaveTextFileRange Method' in the on-line documentation**) methods, specifying the starting row and column, the number of rows and columns to save, the path and file name or stream, whether data or data and formatting is saved, whether headers are saved, and the custom delimiters, depending on the method you choose.

Example

This example code saves a range of data and formatting to a text file, including headers and using custom delimiters.

C#

```
// Save a range of data and formatting to a text file.
fpSpread1.Sheets[0].SaveTextFileRange(1, 1, 1, 2, "C:\\filerange.txt",
FarPoint.Win.Spread.TextFileFlags.Unformatted,
FarPoint.Win.Spread.Model.IncludeHeaders.BothCustomOnly, "#", "%", "^");
```

VB

```
' Save a range of data and formatting to a text file.
fpSpread1.Sheets(0).SaveTextFileRange(1, 1, 1, 2, "C:\\filerange.txt",
FarPoint.Win.Spread.TextFileFlags.Unformatted,
FarPoint.Win.Spread.Model.IncludeHeaders.BothCustomOnly, "#", "%", "^")
```

Saving to an Image File

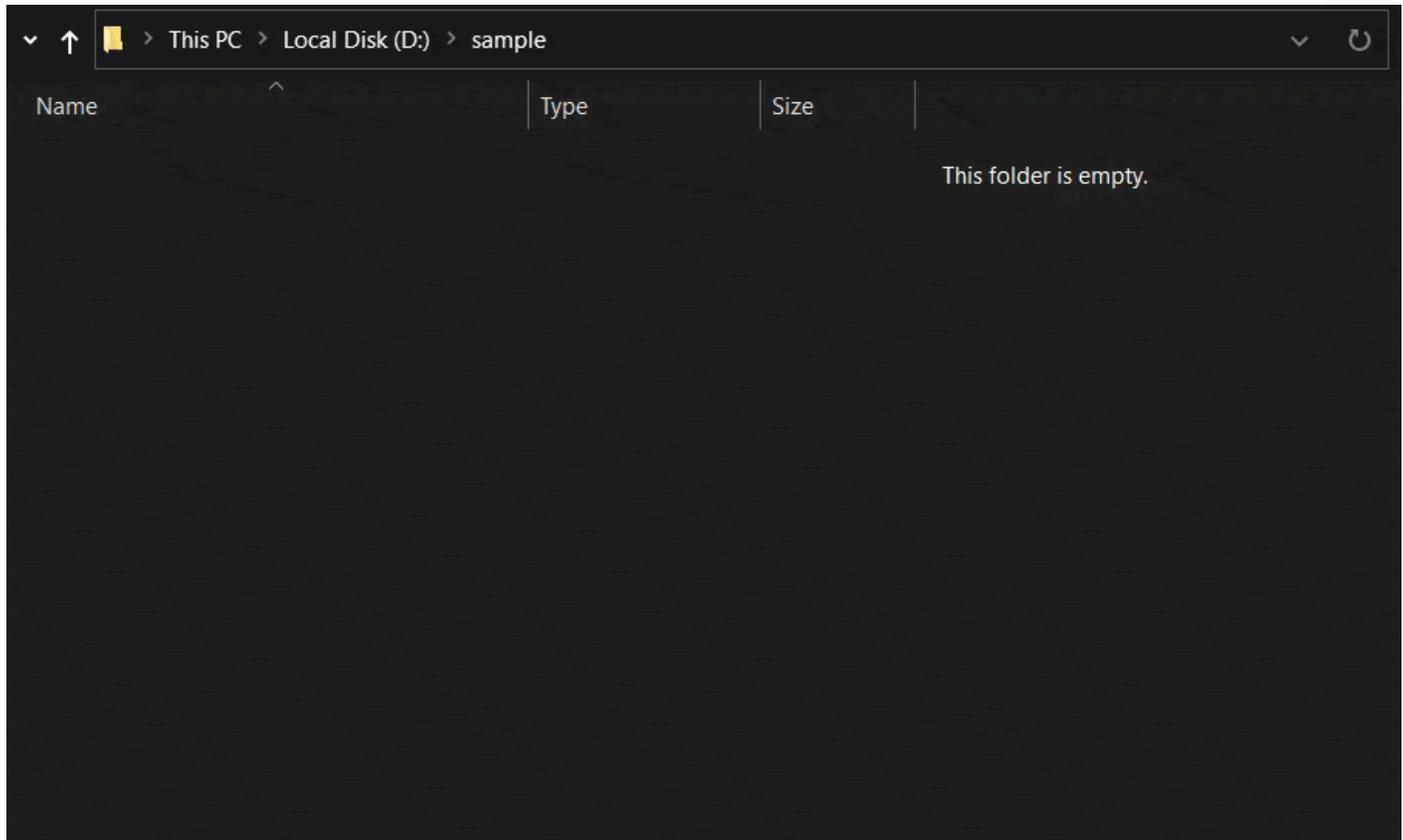
You can save data in an image file by creating a bitmap instance using the **SheetView.SaveImage ('SaveImage Method' in the on-line documentation)** class method. This method saves all the data in the specified cell range in an image.

Users can provide parameters such as row, column, rowcount, colcount, height, and width. A Bitmap instance is returned as output and it represents the image of the range.

The bitmap instance can be saved to an image file by using the **Save** method. It can also be utilized in the following ways:

- Paint as a graphic.
- Use as an image source.

The following GIF illustrates saving a cell range to an image file.



The following code example shows how to create an image using cell ranges in a worksheet and save the image instance in an image format.

C#

```
// Make sure that fpSpread1.LegacyBehaviors does not contain LegacyBehaviors.Style
fpSpread1.Features.EnhancedShapeEngine = true;

fpSpread1.Sheets[0].Cells[0, 0].Text = "Spread";
fpSpread1.Sheets[0].Cells[0, 1].Text = "For";
fpSpread1.Sheets[0].Cells[0, 2].Text = "Winforms";
fpSpread1.BorderCollapse = FarPoint.Win.Spread.BorderCollapse.Enhanced;

var bitmap = fpSpread1.ActiveSheet.SaveImage(0, 0, 3, 3);
bitmap.Save("D:\\sample\\Img.png", System.Drawing.Imaging.ImageFormat.Png);
```

Visual Basic

```
' Make sure that fpSpread1.LegacyBehaviors does Not contain LegacyBehaviors.Style
FpSpread1.Features.EnhancedShapeEngine = True

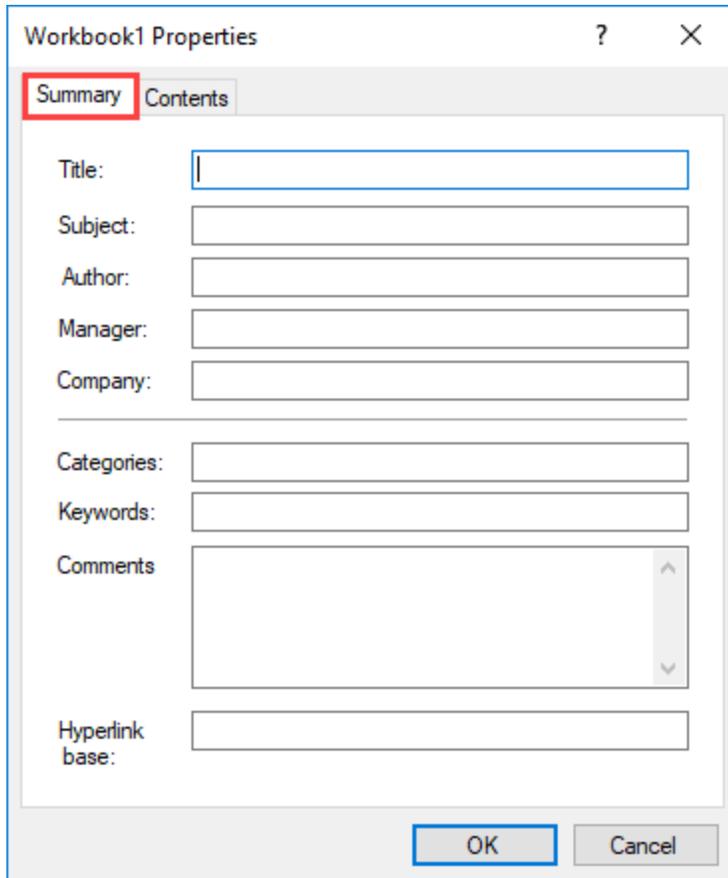
FpSpread1.Sheets(0).Cells(0, 0).Text = "Spread"
FpSpread1.Sheets(0).Cells(0, 1).Text = "For"
FpSpread1.Sheets(0).Cells(0, 2).Text = "Winforms"
FpSpread1.BorderCollapse = FarPoint.Win.Spread.BorderCollapse.Enhanced

Dim bitmap = FpSpread1.ActiveSheet.SaveImage(0, 0, 3, 3)
bitmap.Save("D:\\sample\\Img.png", Imaging.ImageFormat.Png)
```

Storing Excel Summary and View

Storing Excel Summary and View You can store and display the essential information for the workbook using the **Properties** dialog. The following tabs can be used to accomplish this task:

- **Summary Tab** - This tab displays information such as the Title, Subject and Author. You can enter your custom details such as the Manager name and Company name using this menu. Further, you can list the keywords of your choice, apply Comments and add Hyperlink base information to the spreadsheet.
- **Contents Tab** - This tab displays the list of the items present in the workbook such as the number of worksheets.



The screenshot shows the 'Workbook1 Properties' dialog box with the 'Summary' tab selected. The dialog has a title bar with a question mark and a close button. The 'Summary' tab is highlighted with a red box. The dialog contains several input fields: 'Title', 'Subject', 'Author', 'Manager', 'Company', 'Categories', 'Keywords', 'Comments' (a text area with a vertical scrollbar), and 'Hyperlink base'. At the bottom, there are 'OK' and 'Cancel' buttons.

Using the Spread Designer

In order to access the DocumentProperties dialog through Spread Designer, you can select the **Properties** option available in the **File Menu**.

Using Code

This example code shows how to use a Document Properties Form to store the summary and review information.

C#

```
//Get Workbook from fpSpread1
GrapeCity.Spreadsheet.Workbook workbook = fpSpread1.AsWorkbook();
//Call the DocumentPropertiesForm
FarPoint.Win.Spread.DocumentPropertiesForm form = new
```

```
FarPoint.Win.Spread.DocumentPropertiesForm(workbook);
form.Show();
```

VB

```
'Get Workbook from fpSpread1
Dim workbook As GrapeCity.Spreadsheet.Workbook = fpSpread1.AsWorkbook()
'Call the DocumentPropertiesForm
Dim form As FarPoint.Win.Spread.DocumentPropertiesForm = New
FarPoint.Win.Spread.DocumentPropertiesForm(workbook)
form.Show()
```

The **DocumentProperties** (**DocumentProperties Property** in the on-line documentation) property in the **FpSpread** (**FpSpread Class** in the on-line documentation) class represents all the properties of the spread document that can be get or set by the users.

Using Code

This example code shows how to store the Excel summary and review information using **DocumentProperties** (**DocumentProperties Property** in the on-line documentation) property.

C#

```
//Set the view type for the sheetview
fpSpread1.Sheets[0].View = GrapeCity.Spreadsheet.SheetViewType.PageLayout;
//Set the Excel summary properties using Spread's "DocumentProperties" API
fpSpread1.DocumentProperties.Title = "Spread Mescius";
fpSpread1.DocumentProperties.Creator = "Mescius";
fpSpread1.DocumentProperties.Version = "12.0.0.0";
fpSpread1.DocumentProperties.Description = "Test values for Excel summary";
fpSpread1.DocumentProperties.Application = "Test Application";
```

VB

```
'Set the view type for the sheetview
fpSpread1.Sheets(0).View = GrapeCity.Spreadsheet.SheetViewType.PageLayout
'Set the Excel summary properties using Spread's "DocumentProperties" API
fpSpread1.DocumentProperties.Title = "Spread Mescius"
fpSpread1.DocumentProperties.Creator = "Mescius"
fpSpread1.DocumentProperties.Version = "12.0.0.0"
fpSpread1.DocumentProperties.Description = "Test values for Excel summary"
fpSpread1.DocumentProperties.Application = "Test Application"
```

Saving to an HTML Table

You can save an individual sheet or a range of cells in a sheet to an HTML table in a file or stream if you need to display the sheet in a Web browser. This does not save the entire spreadsheet, only an individual sheet.

The HTML export saves as much of the formatting information or presentation-related settings as possible, depending on whether that information can be translated to an HTML element or attribute. The header cells are not saved out to XML; only the data area cells. If you have grouping turned on, only the data area of the sheet (not the grouping bar at the top, and not the group heading rows) is saved.

To save the sheet to an HTML table, use the **SaveHtml** (**SaveHtml Method** in the on-line documentation) methods of the **SheetView** (**SheetView Class** in the on-line documentation) class.

 The HTML export methods will not work with client profiling.

To save a range of cells on a sheet to an HTML table, use the **SaveHtmlRange** (**SaveHtmlRange Method** in the on-line documentation) methods of the **SheetView** (**SheetView Class** in the on-line documentation) class.

In the example below, notice that the entire file is in a single <TABLE> element and that each of the cells, both headings and data area, are translated to table cells. Heading cells are output as <TH> elements (table heading cells) and data area cells are output as <TD> elements (table data cells). All of the formatting information is preserved and kept as attributes in the table cell attributes. A set of <COLGROUP> elements (column groups) define the width of the columns in the table. The appearance of the spreadsheet is reproduced as closely as possible in HTML in this fairly simple organization.

For detailed specifications of the HTML elements and their attributes, refer to the World Wide Web Consortium (W3C) (<https://www.w3.org>) HTML 4.01 reference site (<https://www.w3.org/TR/html4/>).

 The **SaveHTML** methods (**SaveHtml Method** in the on-line documentation) are not supported in .NET Core 3.1 and .NET 6. However, they can be used in a .NET full-framework project.

Using Code

Use the **SaveHtml** (**SaveHtml Method** in the on-line documentation) method.

Example

This example uses the **SaveHtml** (**SaveHtml Method** in the on-line documentation) method.

```
C#
fpSpread1.ActiveSheet.SaveHtml("C:\\testfiles\\FPSpread-SheetToHTML.html");
```

VB

```
fpSpread1.ActiveSheet.SaveHtml("C:\testfiles\FPSpread-SheetToHTML.html")
```

This gives the following result:

```
<table cellpadding="0" cellspacing="0" border="1" style="border-width:1px;border-style:solid;width:548px;border-collapse:collapse;">
<COLGROUP>
  <col width=49px>
  <col width=150px>
  <col width=300px>   <col width=49px>
</COLGROUP>
<tr style="height:20px;">
  <th style="background-color:#A9A9A9;"></th>
  <th align="center" valign="middle" style="color:White;background-color:#A9A9A9;font-family:Arial;font-size:12pt;font-weight:normal;font-style:normal;text-decoration:none;">Artist</th>
  <th align="center" valign="middle" style="color:White;background-color:#A9A9A9;font-family:Arial;font-size:12pt;font-weight:normal;font-style:normal;text-decoration:none;">Website</th>
  <th align="center" valign="middle" style="color:White;background-color:#A9A9A9;font-family:Arial;font-size:12pt;font-weight:normal;font-style:normal;text-decoration:none;">Id</th>
</tr>
<tr style="height:20px;">   <th align="center" valign="middle" style="color:White;background-color:#A9A9A9;"></th>
  <td align="left" valign="top" style="color:Gray;background-color:#F8F8FF;font-family:Arial;font-size:12pt;font-weight:normal;font-style:normal;text-decoration:none;">Aerosmith</td>
  <td align="left" valign="top" style="color:Gray;background-color:#F8F8FF;font-family:Arial;font-size:12pt;font-weight:normal;font-style:normal;text-decoration:none;">http://www.aerosmith.com/detect.html</td>
  <td align="right" valign="top" style="color:Gray;background-color:#F8F8FF;font-family:Arial;font-size:12pt;font-weight:normal;font-style:normal;text-decoration:none;">1</td>
</tr>
<tr style="height:20px;">
  <th align="center" valign="middle" style="color:White;background-color:#A9A9A9;"></th>
  <td align="left" valign="top" style="color:Gray;background-color:#DCDCDC;font-family:Arial;font-size:12pt;font-weight:normal;font-style:normal;text-decoration:none;">Foreigner</td>
  <td align="left" valign="top" style="color:Gray;background-color:#DCDCDC;font-family:Arial;font-size:12pt;font-weight:normal;font-style:normal;text-decoration:none;">http://www.foreigneronline.com/</td>
  <td align="right" valign="top" style="color:Gray;background-color:#DCDCDC;font-family:Arial;font-size:12pt;font-weight:normal;font-style:normal;text-decoration:none;">3</td>
</tr>
<tr style="height:20px;">
  <th align="center" valign="middle" style="color:White;background-color:#A9A9A9;"></th>
  <td align="left" valign="top" style="color:Gray;background-color:#F8F8FF;font-family:Arial;font-size:12pt;font-weight:normal;font-style:normal;text-decoration:none;">Jimi Hendrix</td>
  <td align="left" valign="top" style="color:Gray;background-color:#F8F8FF;font-family:Arial;font-size:12pt;font-weight:normal;font-style:normal;text-decoration:none;">http://www.jimi-hendrix.com/</td>
  <td align="right" valign="top" style="color:Gray;background-color:#F8F8FF;font-family:Arial;font-size:12pt;font-weight:normal;font-style:normal;text-decoration:none;">2</td>
</tr>
<tr style="height:20px;">
  <th align="center" valign="middle" style="color:White;background-color:#A9A9A9;"></th>
  <td align="left" valign="top" style="color:Gray;background-color:#DCDCDC;font-family:Arial;font-size:12pt;font-weight:normal;font-style:normal;text-decoration:none;">Pink Floyd</td>
  <td align="left" valign="top" style="color:Gray;background-color:#DCDCDC;font-family:Arial;font-size:12pt;font-weight:normal;font-style:normal;text-decoration:none;">http://www.pinkfloyd.com/</td>
  <td align="right" valign="top" style="color:Gray;background-color:#DCDCDC;font-family:Arial;font-size:12pt;font-weight:normal;font-style:normal;text-decoration:none;">3</td>
</tr>
<tr style="height:20px;">
  <th align="center" valign="middle" style="color:White;background-color:#A9A9A9;"></th>
  <td align="left" valign="top" style="color:Gray;background-color:#F8F8FF;font-family:Arial;font-size:12pt;font-weight:normal;font-style:normal;text-decoration:none;">The Who</td>
  <td align="left" valign="top" style="color:Gray;background-color:#F8F8FF;font-family:Arial;font-size:12pt;font-weight:normal;font-style:normal;text-decoration:none;">http://www.thewho.net/</td>
  <td align="right" valign="top" style="color:Gray;background-color:#F8F8FF;font-family:Arial;font-size:12pt;font-weight:normal;font-style:normal;text-decoration:none;">4</td>
</tr>
</table>
```

Saving Spreadsheet Data to Simple XML

You can save the data from a sheet to an XML file or stream if you need to process the data further and want the data in a structured format. This does not save the entire spreadsheet nor does it save the formatting information or presentation-related settings. Only the values in the cells are saved to XML. The header cells are not saved out to XML; only the data area cells.

To save the data to XML, use the **SaveXml ('SaveXml Method' in the on-line documentation)** methods of the SheetView class.

You can save the data to one file (or stream) and the XML schema to another file (or stream). An example output is shown below. The top level (or root) element is the sheet (SheetName) and within the sheet element are row elements (Row1, Row2, and so on) and within each row are the column elements (Column1, Column2, and so on) and within each column element is the data. The sheet element uses the name of the sheet. The rows are all generic rows and are not given unique names. The columns are each given a unique name and columns without data are ignored.

The **SaveXml ('SaveXml Method' in the on-line documentation)** method only saves the data for an individual sheet, not for an entire hierarchy that consists of several sheets.

This method only saves the data. For cell type data, see **Understanding How Cell Types Display and Format Data**.

Using Code

Use the **SaveXml ('SaveXml Method' in the on-line documentation)** method to save data to an XML file.

Example

This example saves the data for a sheet named "EastCoastSales" to an XML file and the schema to another file.

C#

```
fpSpread1.ActiveSheet.SaveXml("C:\\testfiles\\FPSpread-SheetToXML2.xml", "C:\\testfiles\\FPSpread-SchemaForXML2.xml");
```

VB

```
fpSpread1.ActiveSheet.SaveXml("C:\\testfiles\\FPSpread-SheetToXML2.xml", "C:\\testfiles\\FPSpread-SchemaForXML2.xml")
```

This gives the following result:

```
<EastCoastSales>
  <Row>
    <Column1>Aerosmith</Column1>
    <Column2>http://www.aerosmith.com/detect.html</Column2>
    <Column3>0</Column3>
  </Row>
  <Row>
    <Column1>Foreigner</Column1>
    <Column2>http://www.foreigneronline.com/</Column2>
    <Column3>1</Column3>
  </Row>
  .
  .
  .
  <Row>
    <Column1>The Who</Column1>
    <Column2>http://www.thewho.net/</Column2>
    <Column3>4</Column3>
  </Row>
</EastCoastSales>
```

The corresponding schema would be something like this:

```
<?xml version="1.0" encoding="utf-16"?><xs:schema
id="Sheet1" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"> <xs:element
name="EastCoastSales" msdata:IsDataSet="true" msdata:UseCurrentLocale="true"> <xs:complexType> <xs:choice
minOccurs="0" maxOccurs="unbounded"> <xs:element name="Row"> <xs:complexType> <xs:sequence> <xs:element
name="Column1" type="xs:string" minOccurs="0" /> <xs:element
name="Column2" type="xs:string" minOccurs="0" /> <xs:element
name="Column3" type="xs:string" minOccurs="0" /> </xs:sequence> </xs:complexType> </xs:element> </xs:choice> </xs:complexType> </xs:element></xs:schema>
```

Opening Existing Files

Spread can open XML files or stream objects that were created by Spread, as well as text and Excel files. Text files must use delimiters that Spread can process to place data into the appropriate cells. Spread provides methods for opening several file types. Consult the following topics for instructions and more information regarding saving to a file.

- **Opening a Spread XML File**
- **Opening an Excel File**
- **Opening a Custom Text File**
- **Opening a Spread COM File**

If you open or load a file or Stream object that contains more rows or columns than the sheet or sheets into which you are opening the file or stream, the component adds rows or columns as needed to the sheet or sheets. If you open or load a file or Stream object with fewer rows or columns than the sheet or sheets into which you are opening the file or stream, the component opens the file and loads the data, and does not remove the additional rows or columns in the sheet or sheets.

Opening existing files using Spread Designer places the data from the file into the design string used to create the component. Longer design strings negatively impact responsiveness, including making page loads slower and increasing response time to editing. Keep this in mind when using Spread Designer to open and load files.

Opening a Spread XML File

Spread can open data or data and formatting from a Spread-compatible XML file or a stream into the Spread component.

For more details on opening a Spread XML file, refer to the **Open ('Open Method' in the on-line documentation)** methods of the **FpSpread ('FpSpread Class' in the on-line documentation)** class or the **Open ('Open Method' in the on-line documentation)** methods of the **SheetView ('SheetView Class' in the on-line documentation)** class.

For instructions for saving Spread XML files, see **Saving to a Spread XML File**.

Using Code

Use the **FpSpread ('FpSpread Class' in the on-line documentation)** class **Open ('Open Method' in the on-line documentation)** method, specifying the path and file name of the Spread XML file to open or the Stream object to open.

Example

This example code opens an existing Spread-compatible XML file in the component.

C#

```
// Open a Spread-compatible XML file.  
fpSpread1.Open("C:\\spreadfile.xml");
```

VB

```
' Open a Spread-compatible XML file.  
fpSpread1.Open("C:\\spreadfile.xml")
```

Using the Spread Designer

1. From the **File** menu, select **Open**.
2. A dialog appears warning you that this overwrites your existing settings if you open a file. Click **Yes** to continue with the file open.
The **Open** dialog appears.
3. Change the **Files of** type box to XML files (*.xml).
4. Specify the path and file name of the file to open, and then click **Open**.
If the file is opened successfully, a message appears stating the file has been opened.
5. Click **OK** to close the Spread Designer.

Opening an Excel File

Opening an Excel File You can open an existing Excel-formatted file (BIFF8 format or xlsx) in Spread. You can open the entire multiple-sheet file into the Spread component or specify a particular sheet (either by name or number) and open it into a specific sheet.

Spread can be used in both bound and unbound modes. When opening an Excel file, Spread is being used in the unbound mode and thus the **DataSource** property returns null (or Nothing in Visual Basic).

Use one of the **OpenExcel ('OpenExcel Method' in the on-line documentation)** methods of the **FpSpread ('FpSpread Class' in the on-line documentation)** class to open all the sheets in the Excel file, providing the path and file name for the file to open and any additional information. You can specify additional open options with the **ExcelOpenFlags ('ExcelOpenFlags Enumeration' in the on-line documentation)** enumeration. This enumeration allows you to determine how frozen columns and rows are imported, if data only is imported, and other options. To open a specific sheet of the Excel file, use one of the **OpenExcel ('OpenExcel Method' in the on-line documentation)** methods of the **SheetView ('SheetView Class' in the on-line documentation)** class, specifying the sheet by name or number.

The Document caching option in the **ExcelOpenFlags** or **ExcelSaveFlags** enumeration allows users to open, edit, and save without the loss of advanced document content and formatting. The content can be lossless only if the opening file format is similar to the saving file format. If the advanced document content uses files besides the xls(x) file, then the additional files need to be in the same folder with the xls(x) file. Advanced content could be macros, ActiveX controls, data connections, and so on.

Note that the sheet index referring to sheets in the Excel file is zero-based, so the first sheet in the Excel file is 0, the second is 1, and so on.

If the Excel file is open in another application (open in Excel for example) when you are trying to open it in Spread, nothing is imported, and the Spread opens without any imported data.

For more information about how the data is imported from an Excel-formatted file, see the **Import and Export Reference (on-line documentation)**.

Using Code

Use one of the **OpenExcel ('OpenExcel Method' in the on-line documentation)** methods of the **FpSpread ('FpSpread Class' in the on-line documentation)** class to an Excel file. To open a specific sheet of the Excel file, use one of the **OpenExcel ('OpenExcel Method' in the on-line documentation)** methods of the **SheetView ('SheetView Class' in the on-line documentation)** class (using the **Sheets** or **ActiveSheet** shortcut).

Example

This example code opens an entire Excel-formatted file using the method in the **FpSpread ('FpSpread Class' in the on-line documentation)** class, and loads the data from the specified Excel sheet into the specified sheet of the Spread component.

C#

```
// Open the fourth sheet of the Excel file.  
fpSpread1.ActiveSheet.OpenExcel("C:\\excelfile.xls", 3);
```

VB

```
' Open the fourth sheet of the Excel file.  
fpSpread1.ActiveSheet.OpenExcel("C:\\excelfile.xls", 3)
```

Using the Spread Designer

1. From the **File** menu, select **Open**.
2. A dialog appears warning you that this overwrites your existing settings if you open a file. Click **Yes** to continue with the file open.
The **Open** dialog appears.
3. Change the **Files of type** box to Excel files (*.xls).
To open a comma-delimited file, change the Files of type box to Comma-delimited files (*.csv).
4. Specify the path and file name of the file to open, and then click **Open**.
If the file is opened successfully, a message appears stating the file has been opened.
5. Click **OK** to close the Spread Designer.

Opening a Custom Text File

You can open existing text files that are delimited, either files saved from Spread or delimited text files from other sources. The data from the file you open is placed in the sheet for which you call the method.

If the file uses custom delimiters (such as commas in csv files), you must specify the delimiters so Opening a Custom Text Filethe component can correctly place the data within the sheet. If your file uses standard tab-delimited format, you need not use a method that lets you specify delimiters.

 If you upgraded from a version prior to version 5, then you may wish to replace any obsolete parameters in the **LoadTextFile** methods.

For more details and current parameters, refer to the **LoadTextFile** ('**LoadTextFile Method**' in the on-line documentation) or **LoadTextFileRange** ('**LoadTextFileRange Method**' in the on-line documentation) methods in the **SheetView** ('**SheetView Class**' in the on-line documentation) class.

For instructions for saving to text files, see **Saving to a Text File**.

Using Code

Use any of the **LoadTextFile** ('**LoadTextFile Method**' in the on-line documentation) methods in the **SheetView** ('**SheetView Class**' in the on-line documentation) class (for the **ActiveSheet** or **Sheets** object).

Example

This example opens a text file and handles the headers and specifies the delimiters.

C#

```
fpSpread1.ActiveSheet.LoadTextFile("C:\\textfile.txt",  
FarPoint.Win.Spread.TextFileFlags.Unformatted,  
FarPoint.Win.Spread.Model.IncludeHeaders.BothCustomOnly, "\\n", ",", "");
```

VB

```
fpSpread1.ActiveSheet.LoadTextFile("C:\textfile.txt",  
FarPoint.Win.Spread.TextFileFlags.Unformatted,  
FarPoint.Win.Spread.Model.IncludeHeaders.BothCustomOnly, Chr(10), ",", "")
```

Using the Spread Designer

1. From the **File** menu, select **Open**.
2. A dialog appears warning you that this overwrites your existing settings if you open a file. Click **Yes** to continue with the file open.
The **Open** dialog appears.
3. Change the **Files of type** box to Custom text files (*.txt).
To open a comma-delimited file, change the **Files of type** box to Comma-delimited files (*.csv).
4. Specify the path and file name of the file to open, and then click **Open**.
If the file is opened successfully, a message appears stating the file has been opened.
5. Click **OK** to close the Spread Designer.

Opening a Spread COM File

You can open an existing file from the COM version of Spread (Spread COM 7.0 or later). For details on opening a Spread COM file, refer to the **OpenSpreadFile** ('**OpenSpreadFile Method**' in the on-line documentation) methods of the **FpSpread** class.

Since the .NET platform is completely different from the COM environment, there are many differences between the Spread file for the products in these two environments. For more information about importing Spread COM files and

understanding compatibility with Spread Windows Forms, refer to the **Version Comparison Reference (on-line documentation)**.

Using Code

Use the **OpenSpreadFile ('OpenSpreadFile Method' in the on-line documentation)** method of the **FpSpread** class, providing the path and file name for the file to open, or use the **OpenSpreadFile ('OpenSpreadFile Method' in the on-line documentation)** method of the **SheetView ('SheetView Class' in the on-line documentation)** class (in the **ActiveSheet** or **Sheets** shortcut object).

Example

This example code opens a Spread COM 7 file in the active sheet.

C#

```
// Open an old Spread 7 file into the active sheet.  
fpSpread1.ActiveSheet.OpenSpreadFile("C:\\oldfile.ss7");
```

VB

```
' Open an old Spread 7 file into the active sheet.  
fpSpread1.ActiveSheet.OpenSpreadFile("C:\\oldfile.ss7")
```

Using the Spread Designer

1. From the **File** menu, select **Open**.
2. A dialog appears warning you that this overwrites your existing settings if you open a file. Click **Yes** to continue with the file open.
The **Open** dialog appears.
3. Change the **Files of type** box to Spread 7 files (*.ss7 or *.ss8).
4. Specify the path and file name of the file to open, and then click **Open**.
If the file is opened successfully, a message appears stating the file has been opened.
5. Click **OK** to close the Spread Designer.

Using Serialization

In Spread, you can serialize objects in the spreadsheet or the entire spreadsheet component using several methods. Be sure you understand the difference between saving with the methods in **FarPoint.Win.Serializer ('Serializer Class' in the on-line documentation)** as opposed to the ones in the model namespace **FarPoint.Win.Spread.Model.SpreadSerializer ('SpreadSerializer Class' in the on-line documentation)**. The **SpreadSerializer ('SpreadSerializer Class' in the on-line documentation)** is intended for saving and loading entire Spread component objects or text files from a sheet, while the **Serializer ('Serializer Class' in the on-line documentation)** contains methods for saving and loading any serializable object using our XML serialization implementation. For more details on serializing and deserializing, refer to these tasks.

- **Implementing a Serializer Class**
- **Parsing Formulas in Custom XML Deserialization**

The method to use for simply saving an object to a file is **SaveObject ('SaveObject Method' in the on-line documentation)**; the method for loading the object back from the file is **LoadObject ('LoadObject Method' in the on-line documentation)**. The methods are overloaded so you can save or load to a stream or use a particular XML serialization interface. For many purposes the overload for **SaveObject ('SaveObject Method' in the on-line documentation)** with only the object, filename, and root element name arguments, and the overload for **LoadObject**

(**'LoadObject Method' in the on-line documentation**) with only the type, filename, and element name should work. The **SaveObject ('SaveObject Method' in the on-line documentation)** method checks to see if the object is in the FarPoint Spread DLL or the FarPoint Win DLL, and if not, then it saves the assembly name in the XML node attributes and uses that name to load the assembly in the **CreateObjectInstanceAndDeserialize ('CreateObjectInstanceAndDeserialize Method' in the on-line documentation)** method.

Design-time serialization is complicated, and Spread uses several custom **CodeDomSerializer** objects to serialize at design time. Those serializers are coded to expect the **FpSpread** and the **SheetView** to be fields in the object being serialized, and if the object is instead returned by a property accessor, the code probably is not generated correctly. For this reason exposing a property of type **FpSpread** or **SheetView** at design time is not recommended.

Implementing a Serializer Class

When using the **Serializer ('Serializer Class' in the on-line documentation)** class, remember these tips:

- You must have a parameterless constructor defined in order for **Serializer.CreateObjectInstanceAndDeserialize ('CreateObjectInstanceAndDeserialize Method' in the on-line documentation)** to create a new instance of your class to deserialize.
- It is a good idea to initialize the fields in the class with some default or dummy values, but it is not necessary if you always save each field to the XML regardless of whether it is set to its default (which is required to make **GetObjectXml ('GetObjectXml Method' in the on-line documentation)** and **SetObjectXml ('SetObjectXml Method' in the on-line documentation)** work properly).
- The **Serializer CanSerializeObject ('CanSerializeObject Method' in the on-line documentation)** method determines whether the specified object can be serialized with **SerializeObject**. This method checks whether the object implements **ICanSerializeXml** (in which case it returns the value of the object's **CanSerializeXml** property), or whether the object implements **ISerializeSupport**, or is a simple type, or if the type's **ISerializable** property returns true.
- Saving the object using elements (see the first subclass example below) requires more coding, but results in more readable XML when the number of properties being saved is high (as in this example of the **LineBorder** class below).
- Saving the object using attributes (the second subclass example below) requires less coding, and results in more readable XML when the number of properties being saved is low (as in this example of the **GradientLineBorder** class below), so the latter subclass example is better.
- If you implement **ISerializeSupport ('ISerializeSupport Interface' in the on-line documentation)** using elements, then you should call the base class implementation first (if there is one), then save and load your properties. You should also return after reading the last end element that you serialize, so that subclasses can also use elements to serialize their properties.
- If you implement **ISerializeSupport ('ISerializeSupport Interface' in the on-line documentation)** using attributes, then you should read your attributes first, then call the base class implementation (if there is one).
- You can use both attributes and elements to serialize your object; there are some Spread objects (mostly the models) which do this. In that case you should do the attributes first, then call the base class, then save and load your elements, returning from **Deserialize** after you reach the last end element.

The examples below show saving simple things like integers and colors, but saving other types is no different. There are methods in **Serializer** for saving and loading Color, DateTime, DateTimeFormatInfo, Enum types, Font, Image, Int32 array, NumberFormatInfo, PointF array, String array, and Object. It is best (though not required) to use the most specific method available. For example, it would work to use **SerializeObject** to save a simple type or a Color value, but it is less efficient and may not result in the same XML.

In general, there is a reason each of the methods was created for its specific type. **DateTimeFormatInfo** and **NumberFormatInfo** must be saved using their respective methods because **SerializeObject** would default to using binary serialization, which does not work across versions of the framework (so a **DateTimeFormatInfo** saved using .NET 1.0 does not load using .NET 1.1 or 2.0).

For simple types (Int32, String, Boolean, etc.) it is best to use **Tostring()** to convert them to a string for saving (passing

CultureInfo.InvariantCulture if possible) and then use **XmlConvert** to convert the strings back into values.

SaveObject is intended for saving objects which implement **ISerializeSupport** (**'ISerializeSupport Interface' in the on-line documentation**), or for certain specific types (intrinsic data types and DataSet), or for generic objects which are serializable but do not implement **ISerializeSupport** (such objects get saved using **EncodeObject**, which uses a **BinaryFormatter** to save the object to a **MemoryStream** and then uses **Convert** to encode the bytes into a base-64 string).

Debug versus Release Build

To get files saved with a debug build to load into a release build that is strong named, you must eliminate or update the assembly attribute in the node where the object is saved. **SerializeObject** will write an assembly attribute if the assembly in which the object's type is defined is different from the calling assembly and different from the executing assembly (FarPoint.Win.dll).

The calling assembly is usually FarPoint.Win.Spread.dll but is possibly the user's assembly if they are defining their own objects which implement **ISerializeSupport** and saving child objects using **SerializeObject**. The tricky case is, unfortunately, likely the most common one: a user creates a custom object to plug into the Spread's object model (for example, a custom data model) and they want to save and load the Spread using the save and load methods in **FpSpread** and use different builds of their assembly.

In that case, the assembly attribute is required to load the correct assembly containing the object's type definition, but the correct attribute for the debug and release builds is different. In that case, the user would need to edit the value in the XML to change the assembly reference (using something like **String.Replace**).

The less common cases involve users who are using **ISerializeSupport** and **Serializer** to save and load their own files containing objects that they define, perhaps in the same assembly or in a different assembly that they are writing in parallel. If the objects are defined in the same assembly, then the assembly tab will not be written out, and if it is a different assembly, the developer can call the overloads for **SerializeObject** and **CreateObjectInstanceAndDeserialize** which take the **callingAssembly** argument and pass the assembly containing the object (even if that is not really the assembly calling into **Serializer**). That will prevent the assembly attribute from being written out, but it is important that this be done on both ends (the call to save and load) so that the type reference can be resolved and an instance of the object created.

Here are some examples of how you might implement **ISerializeSupport**.

Using Code

This example uses the base class.

Example

The following code provides the implementation of **ISerializeSupport** in **FarPoint.Win.LineBorder**:

C#

```
// A constructor with no arguments is required for ISerializeSupport.
// It does not need to be public though.
internal LineBorder()
{
    color = SystemColors.WindowFrame;
    thickness = 1;
    left = top = right = bottom = true;
    inset = new Inset(1);
}
/// <summary>
/// Saves the object to XML.
/// </summary>
/// <param name="w">XmlTextWriter object to which to save the object</param>
```

```
/// <returns>true if successful; false otherwise</returns>
public virtual bool Serialize(XmlTextWriter w)
{
    w.WriteStartElement("Inset");
    w.WriteElementString("Bottom", inset.Bottom.ToString());
    w.WriteElementString("Left", inset.Left.ToString());
    w.WriteElementString("Right", inset.Right.ToString());
    w.WriteElementString("Top", inset.Top.ToString());
    w.WriteEndElement();
    Serializer.SerializeColor(color, "Color", w);
    w.WriteStartElement("Sides");
    w.WriteElementString("Bottom", bottom.ToString());
    w.WriteElementString("Left", left.ToString());
    w.WriteElementString("Right", right.ToString());
    w.WriteElementString("Top", top.ToString());
    w.WriteEndElement();
    w.WriteElementString("Thickness",
thickness.ToString(CultureInfo.InvariantCulture));
    return true;
}

/// <summary>
/// Loads the object from XML.
/// </summary>
/// <param name="r">XmlNodeReader from which to load the object</param>
/// <returns>true if successful; false otherwise</returns>
public virtual bool Deserialize(XmlNodeReader r)
{
    bool inInset = false;
    bool inBottom = false;
    bool inLeft = false;
    bool inRight = false;
    bool inTop = false;
    bool inColor = false;
    bool inSides = false;
    bool inThickness = false;
    int bottom = inset.Bottom;
    int left = inset.Left;
    int right = inset.Right;
    if( r.IsEmptyElement )
        return true;
    while( r.Read() )
    {
        switch( r.NodeType )
        {
            {
            case XmlNodeType.Element:
            if( r.Name.Equals("Inset") )
                inInset = true;
            else if( r.Name.Equals("Color") )
                inColor = true;
            else if( r.Name.Equals("Sides") )
                inSides = true;
            else if( inInset || inSides )
            {
                if( r.Name.Equals("Bottom") )
                    inBottom = true;
                else if( r.Name.Equals("Left") )
```

```

        inLeft = true;
    else if( r.Name.Equals("Right") )
        inRight = true;
    else if( r.Name.Equals("Top") )
        inTop = true;
    }
    else if( r.Name.Equals("Thickness") )
        inThickness = true;
    break;
case XmlNodeType.Text:
    if( inInset )
{
if( inBottom )
bottom = XmlConvert.ToInt32(r.Value);
else if( inLeft )
left = XmlConvert.ToInt32(r.Value);
else if( inRight )
right = XmlConvert.ToInt32(r.Value);
else if( inTop )
inset = new Inset(left, XmlConvert.ToInt32(r.Value), right, bottom);
}
else if( inSides )
{
if( inBottom )
this.bottom = XmlConvert.ToBoolean(r.Value.ToLower(CultureInfo.InvariantCulture));
else if( inLeft )
this.left = XmlConvert.ToBoolean(r.Value.ToLower(CultureInfo.InvariantCulture));
else if( inRight )
this.right = XmlConvert.ToBoolean(r.Value.ToLower(CultureInfo.InvariantCulture));
else if( inTop )
this.top = XmlConvert.ToBoolean(r.Value.ToLower(CultureInfo.InvariantCulture));
}
else if( inColor )
color = Serializer.DeserializeColorValue(r.Value);
else if( inThickness )
thickness = XmlConvert.ToInt32(r.Value);
break;
case XmlNodeType.EndElement:
if( inInset && r.Name.Equals("Inset") )
inInset = false;
else if( inColor && r.Name.Equals("Color") )
inColor = false;
else if( inSides && r.Name.Equals("Sides") )
inSides = false;
else if( inInset || inSides )
{
if( inBottom && r.Name.Equals("Bottom") )
inBottom = false;
else if( inLeft && r.Name.Equals("Left") )
inLeft = false;
else if( inRight && r.Name.Equals("Right") )
inRight = false;
else if( inTop && r.Name.Equals("Top") )
inTop = false;
}
else if( inThickness && r.Name.Equals("Thickness") )
// return here so that subclasses can deserialize

```

```
return true;
break;
}
}
return true;
}
```

Example

This is an example of what a derived class might do in overrides for those methods:

C#

```
// A constructor with no arguments is required for ISerializeSupport.
// It does not need to be public though.
protected GradientLineBorder() : LineBorder(SystemColors.WindowFrame, 1, true, true,
true, true)
{
startColor = Color.Blue;
endColor = Color.Green;
}
/// <summary>
/// Saves the object to XML.
/// </summary>
/// <param name="w">XmlTextWriter object to which to save the object</param>
/// <returns>true if successful; false otherwise</returns>
public override bool Serialize(XmlTextWriter w)
{
if( !base.Serialize(w) )
return false;
Serializer.SerializeColor(startColor, "StartColor", w);
Serailizer.SerializeColor(endColor, "EndColor", w);
return true;
}
/// <summary>
/// Loads the object from XML.
/// </summary>
/// <param name="r">XmlNodeReader from which to load the object</param>
/// <returns>true if successful; false otherwise</returns>
public override bool Deserialize(XmlNodeReader r)
{
bool inStartColor = false;
bool inEndColor = false;

if( r.IsEmptyElement )
return true;
if( !base.Deserialize(r) )
return false;
while( r.Read() )
{
switch( r.NodeType )
{
case XmlNodeType.Element:
if( r.Name.Equals("StartColor") )
inStartColor = true;
else if( r.Name.Equals("EndColor") )
inEndColor = true;
}
```

```

case XmlNodeType.Text:
if( inStartColor )
startColor = Serializer.DeserializeColorValue(r.Value);
else if( inEndColor )
endColor = Serializer.DeserializeColorValue(r.Value);
break;
case XmlNodeType.EndElement:
if( inStartColor && r.Name.Equals("StartColor") )
inStartColor = false;
else if( inEndColor && r.Name.Equals("EndColor") )
// return here so that subclasses can deserialize
return true;
break;
}
}
return true;
}

```

Example

Another (somewhat easier) way to implement it would be this way:

C#

```

/// <summary>
/// Saves the object to XML.
/// </summary>
/// <param name="w">XmlTextWriter object to which to save the object</param>
/// <returns>true if successful; false otherwise</returns>
public override bool Serialize(XmlTextWriter w)
{ w.WriteAttributeString("startColor", Serializer.SerializeColorValue(startColor));
w.WriteAttributeString("endColor", Serializer.SerializeColorValue(endColor));
return base.Serialize(w);
}
/// <summary>
/// Loads the object from XML.
/// </summary>
/// <param name="r">XmlNodeReader from which to load the object</param>
/// <returns>true if successful; false otherwise</returns>
public override bool Deserialize(XmlNodeReader r)
{
if( r.MoveToAttribute("startColor") )
startColor = Serializer.DeserializeColorValue(r.Value);
if( r.MoveToAttribute("endColor") )
endColor = Serializer.DeserializeColorValue(r.Value);
return base.Deserialize(r); }

```

Parsing Formulas in Custom XML Deserialization

With the implementation of cross-sheet references in Spread, formulas can contain references to other sheets. If such formulas are loaded from a file, the Spread component must load sheets and formulas in a specific sequence for the cross-sheet references to work. The parsing of formulas loaded from a file must be delayed until all the sheets in the workbook have been loaded. The **Open** methods in **FpSpread** handle this automatically, loading the XML and parsing in the correct order.

If you have created custom deserialization code, then you have to be careful. If your custom code loads individual sheets and adds them to a workbook, then you will need to add code to parse the formulas after the sheet has been added to the workbook. This can be done with the **LoadFormulas** (**LoadFormulas Method** in the on-line documentation) method, available in the **FpSpread** (**FpSpread Class** in the on-line documentation) class. This method is implemented in the **DefaultSheetDataModel** (**DefaultSheetDataModel Class** in the on-line documentation) and **SheetView** (**SheetView Class** in the on-line documentation) classes as well as the **FpSpread** (**FpSpread Class** in the on-line documentation) class. Using the **LoadFormulas** (**LoadFormulas Method** in the on-line documentation) method at the sheet level calls the method on all the data models for a particular sheet; using the **LoadFormulas** (**LoadFormulas Method** in the on-line documentation) method at the **FpSpread** level calls the method on all the sheets.

For example, the following code (in Visual Basic):

```
FpSpread.Sheets.Add(FarPoint.Win.Serializer.LoadObject(GetType (FarPoint.Win.Spread.SheetView), "C:\SavedSheet.xml", "RootNode"))
```

should be changed to:

```
FpSpread.Sheets.Add(FarPoint.Win.Serializer.LoadObject(GetType (FarPoint.Win.Spread.SheetView), "C:\SavedSheet.xml", "RootNode")) FpSpread.LoadFormulas ()
```

This code should be called after the code that loads the sheet and adds it to the workbook.

For more details, refer to these methods:

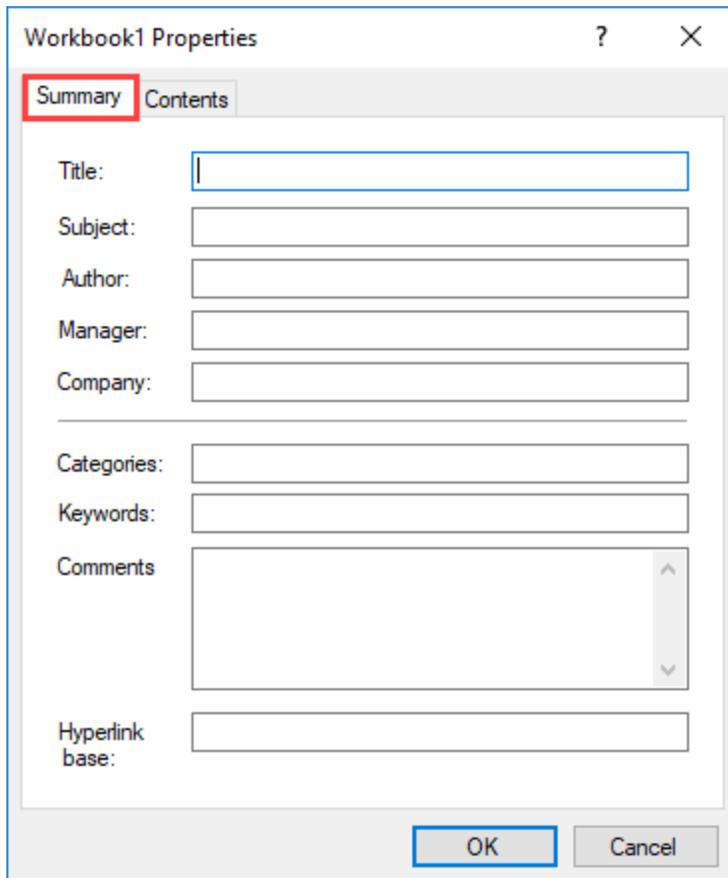
- FpSpread **LoadFormulas** ('LoadFormulas Method' in the on-line documentation) method
- SheetView **LoadFormulas** ('LoadFormulas Method' in the on-line documentation) method
- DefaultSheetDataModel **LoadFormulas** ('LoadFormulas Method' in the on-line documentation) method
- DefaultSheetDataModel **ParseFormula** ('ParseFormula Method' in the on-line documentation) method

For more information about formulas and cross-sheet referencing, refer to the [Formula Reference](#).

Storing Excel Summary and View

Storing Excel Summary and View You can store and display the essential information for the workbook using the **Properties** dialog. The following tabs can be used to accomplish this task:

- **Summary Tab** - This tab displays information such as the Title, Subject and Author. You can enter your custom details such as the Manager name and Company name using this menu. Further, you can list the keywords of your choice, apply Comments and add Hyperlink base information to the spreadsheet.
- **Contents Tab** - This tab displays the list of the items present in the workbook such as the number of worksheets.



Using the Spread Designer

In order to access the DocumentProperties dialog through Spread Designer, you can select the **Properties** option available in the **File Menu**.

Using Code

This example code shows how to use a Document Properties Form to store the summary and review information.

C#

```
//Get Workbook from fpSpread1
GrapeCity.Spreadsheet.Workbook workbook = fpSpread1.AsWorkbook();
//Call the DocumentPropertiesForm
FarPoint.Win.Spread.DocumentPropertiesForm form = new
FarPoint.Win.Spread.DocumentPropertiesForm(workbook);
form.Show();
```

VB

```
'Get Workbook from fpSpread1
Dim workbook As GrapeCity.Spreadsheet.Workbook = fpSpread1.AsWorkbook()
'Call the DocumentPropertiesForm
Dim form As FarPoint.Win.Spread.DocumentPropertiesForm = New
FarPoint.Win.Spread.DocumentPropertiesForm(workbook)
form.Show()
```

The **DocumentProperties** (**DocumentProperties Property** in the on-line documentation) property in the **FpSpread** (**FpSpread Class** in the on-line documentation) class represents all the properties of the spread document that can be get or set by the users.

Using Code

This example code shows how to store the Excel summary and review information using **DocumentProperties** (**DocumentProperties Property** in the on-line documentation) property.

C#

```
//Set the view type for the sheetview
fpSpread1.Sheets[0].View = GrapeCity.Spreadsheet.SheetViewType.PageLayout;
//Set the Excel summary properties using Spread's "DocumentProperties" API
fpSpread1.DocumentProperties.Title = "Spread Mescius";
fpSpread1.DocumentProperties.Creator = "Mescius";
fpSpread1.DocumentProperties.Version = "12.0.0.0";
fpSpread1.DocumentProperties.Description = "Test values for Excel summary";
fpSpread1.DocumentProperties.Application = "Test Application";
```

VB

```
'Set the view type for the sheetview
fpSpread1.Sheets(0).View = GrapeCity.Spreadsheet.SheetViewType.PageLayout
'Set the Excel summary properties using Spread's "DocumentProperties" API
fpSpread1.DocumentProperties.Title = "Spread Mescius"
fpSpread1.DocumentProperties.Creator = "Mescius"
fpSpread1.DocumentProperties.Version = "12.0.0.0"
fpSpread1.DocumentProperties.Description = "Test values for Excel summary"
fpSpread1.DocumentProperties.Application = "Test Application"
```

Printing

You can print a spreadsheet, or parts of a spreadsheet, and use a variety of options to customize printing.

Spread offers you several ways to handle the printing and provides some default handling if you want to keep it simple. You can print various parts of the sheet, you can set options for the appearance of what is printed, you can preview the printing, you can apply printer's setting to print preview, you can show the print preview dialog like Excel, you can print in duplex mode (print on both sides of the paper), you can print more than one page in a sheet of paper and you can provide the printing operation to the end user.

Much of the printing work uses the **PrintSheet ('PrintSheet Method' in the on-line documentation)** method in the **FpSpread ('FpSpread Class' in the on-line documentation)** class. Many of the options available for customizing the printing are in the **PrintInfo ('PrintInfo Property' in the on-line documentation)** class. For each sheet in Spread you assign a **PrintInfo** object. More information on the tasks involved with the management of printing are given in these topics:

- **Specifying What to Print**
- **Customizing the Appearance of the Printing**
- **Optimizing the Printing**
- **Displaying Dialogs for Users**

You can also handle printing within the Spread Designer. For more information on printing and previewing in Spread Designer, refer to **Printing a Sheet from Spread Designer (on-line documentation)**.

Sheets are printed on the current default printer in your Windows environment unless you specify otherwise. You can print sheets on any Windows-supported printer.

Specifying What to Print

You can print any sheet in the workbook or print the entire set of sheets. You can also print any of several parts of the spreadsheet.

- **Printing an Entire Sheet**
- **Printing to PDF**
- **Printing a Child View of a Hierarchical Display**
- **Printing Particular Pages**
- **Printing the Portion of the Sheet with Data**
- **Printing a Range of Cells on a Sheet**
- **Printing an Area of the Sheet**
- **Printing a Sheet with Cell Notes**
- **Printing a Sheet with Shapes**
- **Printing in Duplex Mode**

Printing an Entire Sheet

You can print spreadsheets using the Spread component in the following two ways:

- **Using the PrintSheet() Method**
- **Using the SafePrint() Method**

Using the PrintSheet() Method

You can use this method to print a particular sheet or multiple sheets using the **PrintInfo ('PrintInfo Property' in the on-line documentation)** settings for each sheet. Each sheet can have its own **PrintInfo** object, but you can call

the **PrintSheet** method once to print one or all of the sheets. If the sheet parameter is set to -1, all the sheets in the Spread component will print, with each sheet (with its individual **PrintInfo** settings) as a separate print job. The **PrintDocument ('PrintDocument Event' in the on-line documentation)** event occurs while printing a sheet.

The default setting prints in black and white and automatically determines the best order in which you can print the pages. With the default settings, the following items can be printed using the printer's current orientation setting:

- All the columns and rows in the sheet (but only the cells that have data in them)
- The sheet's border
- The column and row headers
- The header shadows
- The grid lines

To customize these settings, refer to **Understanding the Printing Options** and **Customizing the Printed Page Header or Footer**. To allow Spread to determine the best print settings, refer to **Optimizing the Printing Using Rules**.

You can call the **PrintSheet()** method with different sheet parameters one after the other but calling this method on the same sheet without waiting for the initial print to conclude could produce incorrect results. Before calling the next print for a given sheet, you need to wait for the previous one to be finished. You can do this by catching the **PrintMessageBox** event and querying the **BeginPrinting** parameter to see if it is **False**.

Using Code

Use the **PrintSheet** method in the **FpSpread ('FpSpread Class' in the on-line documentation)** class to print the specified sheet.

Example

This example code prints the second sheet in the component using the **PrintSheet()** method.

C#

```
fpSpread1.PrintSheet(1);
```

VB

```
fpSpread1.PrintSheet(1)
```

Using the SafePrint() Method

The **SafePrint()** method provides users with the flexibility to execute print jobs in a single thread, rather than creating a separate thread. Since it allows printing in the stream synchronously in the same thread, the UI becomes unresponsive till the print operation is complete.

Using this method, you can print spreadsheets in a console application.

Using Code

Use the **SafePrint** method in the **FpSpread ('FpSpread Class' in the on-line documentation)** class to print the spreadsheet in a console application.

Example

This example code prints the second sheet in the console application using the **SafePrint()** method.

C#

```
fpSpread1.SafePrint(fpSpread1,0);
```

VB

```
fpSpread1.SafePrint(fpSpread1,0)
```

You can also use the Spread Designer in order to set properties for printing, and you can also print directly from the Spread Designer. For more information, refer to **Printing a Sheet from Spread Designer (on-line documentation)**.

Printing to PDF

You can print a sheet to a Portable Document Format (PDF, version 1.4) file using the **PrintToPdf ('PrintToPdf Property' in the on-line documentation)** method in the **PrintInfo ('PrintInfo Class' in the on-line documentation)** class. Use the **PdfFileName ('PdfFileName Property' in the on-line documentation)** property to specify the file name and location to which to save the file. Since **PrintInfo** objects are assigned to individual sheets, this method prints an individual sheet.

You can set a password with the **PdfSecurity ('PdfSecurity Property' in the on-line documentation)** property.

The following cell type items are not printed to PDF:

Cell Type	Description
GcTextBoxCellType ('GcTextBoxCellType Class' in the on-line documentation)	LineSpace ('LineSpace Property' in the on-line documentation), Ellipsis ('Ellipsis Property' in the on-line documentation), or DisplayAlignment ('DisplayAlignment Property' in the on-line documentation)
Any Gc cell type	Side button appearance
GcDateTimeCellType ('GcDateTimeCellType Class' in the on-line documentation)	Field appearance
GcNumberCellType ('GcNumberCellType Class' in the on-line documentation)	Field appearance
GcTimeSpanCellType ('GcTimeSpanCellType Class' in the on-line documentation)	Field appearance
GcMaskCellType ('GcMaskCellType Class' in the on-line documentation)	Field appearance
GcCharMaskCellType ('GcCharMaskCellType Class' in the on-line documentation)	Character box appearance
GcComboBoxCellType ('GcComboBoxCellType Class' in the on-line documentation)	Image appearance or ellipsis

To customize print settings, refer to **Understanding the Printing Options** and **Customizing the Printed Page**

Header or Footer. To allow Spread to determine the best print settings, refer to **Optimizing the Printing Using Rules**.

Using Code

Call the **PrintToPdf** ('**PrintToPdf Property**' in the on-line documentation) property in the **PrintInfo** ('**PrintInfo Class**' in the on-line documentation) class to print the specified sheet.

Example

This example code saves the sheet to a PDF file.

C#

```
FarPoint.Win.Spread.PrintInfo printset = new FarPoint.Win.Spread.PrintInfo();
printset.PrintToPdf = true;
printset.PdfFileName = "D:\\results.pdf";
// Assign the printer settings and print
fpSpread1.Sheets[0].PrintInfo = printset;
fpSpread1.PrintSheet(0);
```

VB

```
Dim printset As New FarPoint.Win.Spread.PrintInfo()
printset.PrintToPdf = True
printset.PdfFileName = "D:\\results.pdf"
' Assign the printer settings and print
fpSpread1.Sheets(0).PrintInfo = printset
fpSpread1.PrintSheet(0)
```

Using the Spread Designer

1. Select the **File** menu.
2. Select **Print**.
3. Select **PrintPDF**.
4. Use the **Save** dialog to pick a location and specify a file name.

Printing a Child View of a Hierarchical Display

You can print child sheets of a hierarchy and manage how they are printed. To do this, you specify the specific child view and then use the **PrintSheet** ('**PrintSheet Method**' in the on-line documentation) method as described in **Printing an Entire Sheet**.

For more information about a hierarchical display, refer to **Working with Hierarchical Data Display**.

Using Code

Use the **PrintSheet** ('**PrintSheet Method**' in the on-line documentation) method to print child sheets.

Example

This example prints a child sheet.

C#

```
// Add print code to a command button in a hierarchy example.
FarPoint.Win.Spread.SheetView ss;
ss = fpSpread1.Sheets[0].GetChildView(0, 0);
if (ss != null)
{
    fpSpread1.PrintSheet(ss);
};
```

VB

```
' Add print code to a command button in a hierarchy example.
Dim ss As FarPoint.Win.Spread.SheetView
ss = fpSpread1.Sheets(0).GetChildView(0, 0)
If Not ss Is Nothing Then
    fpSpread1.PrintSheet(ss)
End If
```

Printing Particular Pages

You can print all or some of the pages for the sheet. Specify the pages to print by setting the **PrintType** ('**PrintType Property**' in the on-line documentation), **PageStart** ('**PageStart Property**' in the on-line documentation), and **PageEnd** ('**PageEnd Property**' in the on-line documentation) properties of the **PrintInfo** ('**PrintInfo Class**' in the on-line documentation) object.

You can calculate the number of printed pages for the sheet using the **GetPrintPageCount** ('**GetPrintPageCount Method**' in the on-line documentation) method.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet for which to set the print the page range.
5. In the properties list, double-click the **PrintInfo** property to display the settings for the **PrintInfo** class.
6. Set the **PrintType** property to **PageRange**.
7. Set the **PageStart** and **PageEnd** properties to designate the page range to print.
8. Click **OK** to close the editor.

Using a Shortcut

1. Create a **PrintInfo** ('**PrintInfo Class**' in the on-line documentation) object.
2. Set the **PrintInfo** ('**PrintInfo Class**' in the on-line documentation) object **PrintType** ('**PrintType Property**' in the on-line documentation) property to **PrintType.PageRange**.
If you just want to print the current page, set the **PrintType** ('**PrintType Property**' in the on-line documentation) property to **PrintType.CurrentPage**, and go on to step 4.
3. Set the **PrintInfo** object **PageStart** ('**PageStart Property**' in the on-line documentation) and **PageEnd** ('**PageEnd Property**' in the on-line documentation) properties to designate the page range to print.
4. Set the Sheet shortcut object **PrintInfo** property to the **PrintInfo** object you just created.

Example

This example code prints pages 5 through 10.

C#

```
// Create PrintInfo object and set properties.
FarPoint.Win.Spread.PrintInfo printset = new FarPoint.Win.Spread.PrintInfo();
printset.PrintType = FarPoint.Win.Spread.PrintType.PageRange;
printset.PageStart = 5;
printset.PageEnd = 10;
// Set the PrintInfo property for the first sheet.
fpSpread1.Sheets[0].PrintInfo = printset;
// Print the sheet.
fpSpread1.PrintSheet(0);
```

VB

```
' Create PrintInfo object and set properties.
Dim printset As New FarPoint.Win.Spread.PrintInfo()
printset.PrintType = FarPoint.Win.Spread.PrintType.PageRange
printset.PageStart = 5
printset.PageEnd = 10
' Set the PrintInfo property for the first sheet.
fpSpread1.Sheets(0).PrintInfo = printset
' Print the sheet.
fpSpread1.PrintSheet(0)
```

Using Code

1. Create a **PrintInfo ('PrintInfo Class' in the on-line documentation)** object.
2. Set the **PrintInfo ('PrintInfo Class' in the on-line documentation)** object **PrintType ('PrintType Property' in the on-line documentation)** property to `PrintType.PageRange`.
If you just want to print the current page, set the **PrintType ('PrintType Property' in the on-line documentation)** property to `PrintType.CurrentPage`, and go on to step 4.
3. Set the **PrintInfo** object **PageStart ('PageStart Property' in the on-line documentation)** and **PageEnd ('PageEnd Property' in the on-line documentation)** properties to designate the page range to print.
4. Set the **SheetView** object **PrintInfo** property to the **PrintInfo** object you just created.

Example

This example code prints pages 5 through 10.

C#

```
// Create PrintInfo object and set properties.
FarPoint.Win.Spread.PrintInfo printset = new FarPoint.Win.Spread.PrintInfo();
printset.PrintType = FarPoint.Win.Spread.PrintType.PageRange;
printset.PageStart = 5;
printset.PageEnd = 10;
// Create SheetView object and assign it to the first sheet.
FarPoint.Win.Spread.SheetView SheetToPrint = new FarPoint.Win.Spread.SheetView();
SheetToPrint.PrintInfo = printset;
fpSpread1.Sheets[0] = SheetToPrint;
// Print the sheet.
fpSpread1.PrintSheet(0);
```

VB

```
' Create PrintInfo object and set properties.
```

```

Dim printset As New FarPoint.Win.Spread.PrintInfo()
printset.PrintType = FarPoint.Win.Spread.PrintType.PageRange
printset.PageStart = 5
printset.PageEnd = 10
' Create SheetView object and assign it to the first sheet.
Dim SheetToPrint As New FarPoint.Win.Spread.SheetView()
SheetToPrint.PrintInfo = printset
fpSpread1.Sheets(0) = SheetToPrint
' Set the PrintInfo property for the first sheet.
fpSpread1.Sheets(0).PrintInfo = printset
' Print the sheet.
fpSpread1.PrintSheet(0)

```

Using the Spread Designer

1. Select the sheet tab for the sheet you want to print.
2. From the **Property** window, choose **PrintInfo**.
3. Set **Print Type** to **PageRange**.
4. Set **PageEnd** and **PageStart**.
5. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Printing the Portion of the Sheet with Data

You may not want to print the entire sheet but only the portion of the sheet that has data. Use the **UseMax** (**'UseMax Property' in the on-line documentation**) method of the **PrintInfo** (**'PrintInfo Class' in the on-line documentation**) class to specify whether to print only rows and columns containing data or whether to print all the way to the end of the defined sheet, even if the rows and columns are empty.

If you want to print all of the columns (even if it does not have data in it) but only the rows with data, you would need to set **UseMax** to True to keep from printing rows without data. Then, you would need to programmatically put data in the last column and the last row with data, so the Spread prints all the columns. You could add a line of code similar to the following before calling the **PrintSheet** (**'PrintSheet Method' in the on-line documentation**) method.

C#

```

fpSpread1.Sheets(0).Cells(fpSpread1.Sheets(0).GetLastNonEmptyRow(FarPoint.Win.Spread.NonEmptyItemFlag.Data),
(fpSpread1.Sheets(0).ColumnCount - 1)).Value = ""

```

VB

```

fpSpread1.Sheets[0].Cells[fpSpread1.Sheets[0].GetLastNonEmptyRow(FarPoint.Win.Spread.NonEmptyItemFlag.Data),
fpSpread1.Sheets[0].ColumnCount - 1].Value = "";

```

For more information about methods involved with finding data, refer to **Rows or Columns That Have Data**.

Printing a Range of Cells on a Sheet

You may not want to print the entire sheet but only a specified range of cells on a sheet. You can specify that only a range of cells within a sheet prints, rather than the entire sheet. After specifying the range of cells with the **PrintInfo** (**'PrintInfo Class' in the on-line documentation**) object, use the **PrintSheet** (**'PrintSheet Method' in the on-line documentation**) method as described in **Printing an Entire Sheet**.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet for which to set the print the cell range.

5. In the properties list, double-click the **PrintInfo** property to display the settings for the **PrintInfo** class.
6. Set the **PrintType** property to **PageRange**.
7. Set the **ColStart**, **RowStart**, **ColEnd**, and **RowEnd** properties to designate the cell range to print.
8. Click **OK** to close the editor.

Using a Shortcut

- Create a **PrintInfo** ('PrintInfo Class' in the on-line documentation) object.
- Set the PrintInfo object **PrintType** ('PrintType Property' in the on-line documentation) property to **PrintType.CellRange**.
- Set the PrintInfo object **ColStart** ('ColStart Property' in the on-line documentation), **RowStart** ('RowStart Property' in the on-line documentation), **ColEnd** ('ColEnd Property' in the on-line documentation), and **RowEnd** ('RowEnd Property' in the on-line documentation) properties to designate the cell range to print.
- Set the Sheet shortcut object **PrintInfo** property to the PrintInfo object you just created.

Example

This example code prints cells B2 through D4.

C#

```
// Create PrintInfo object and set properties.
FarPoint.Win.Spread.PrintInfo printset = new FarPoint.Win.Spread.PrintInfo();
printset.PrintType = FarPoint.Win.Spread.PrintType.CellRange;
printset.ColStart = 1;
printset.ColEnd = 3;
printset.RowStart = 1;
printset.RowEnd = 3;
// Set the PrintInfo property for the first sheet.
fpSpread1.Sheets[0].PrintInfo = printset;
// Print the sheet.
fpSpread1.PrintSheet(0);
```

VB

```
' Create PrintInfo object and set properties.
Dim printset As New FarPoint.Win.Spread.PrintInfo()
printset.PrintType = FarPoint.Win.Spread.PrintType.CellRange
printset.ColStart = 1
printset.ColEnd = 3
printset.RowStart = 1
printset.RowEnd = 3
' Set the PrintInfo property for the first sheet.
fpSpread1.Sheets(0).PrintInfo = printset
' Print the sheet.
fpSpread1.PrintSheet(0)
```

Using Code

1. Create a **PrintInfo** ('PrintInfo Class' in the on-line documentation) object.
2. Set the PrintInfo object **PrintType** ('PrintType Property' in the on-line documentation) property to **PrintType.CellRange**.
3. Set the PrintInfo object **ColStart** ('ColStart Property' in the on-line documentation), **RowStart** ('RowStart Property' in the on-line documentation), **ColEnd** ('ColEnd Property' in the on-line documentation), and **RowEnd** ('RowEnd Property' in the on-line documentation) properties to designate the cell range to print.

documentation), and **RowEnd** ('**RowEnd Property**' in the on-line documentation) properties to designate the cell range to print.

4. Set the SheetView object **PrintInfo** property to the PrintInfo object you just created.

Example

This example code prints cells B2 through D4.

C#

```
// Create PrintInfo object and set properties.
FarPoint.Win.Spread.PrintInfo printset = new FarPoint.Win.Spread.PrintInfo();
printset.PrintType = FarPoint.Win.Spread.PrintType.CellRange;
printset.ColStart = 1;
printset.ColEnd = 3;
printset.RowStart = 1;
printset.RowEnd = 3;
// Create SheetView object and assign it to the first sheet.
FarPoint.Win.Spread.SheetView SheetToPrint = new FarPoint.Win.Spread.SheetView();
SheetToPrint.PrintInfo = printset;
fpSpread1.Sheets[0] = SheetToPrint;
// Print the sheet.
fpSpread1.PrintSheet(0);
```

VB

```
' Create PrintInfo object and set properties.
Dim printset As New FarPoint.Win.Spread.PrintInfo()
printset.PrintType = FarPoint.Win.Spread.PrintType.CellRange
printset.ColStart = 1
printset.ColEnd = 3
printset.RowStart = 1
printset.RowEnd = 3
' Create SheetView object and assign it to the first sheet.
Dim SheetToPrint As New FarPoint.Win.Spread.SheetView()
SheetToPrint.PrintInfo = printset
fpSpread1.Sheets(0) = SheetToPrint
' Set the PrintInfo property for the first sheet.
fpSpread1.Sheets(0).PrintInfo = printset
' Print the sheet.
fpSpread1.PrintSheet(0)
```

Using the Spread Designer

1. Select the sheet tab for the sheet you want to print.
2. Select the **Page Layout** option.
3. Select the **PrintTitles** icon, choose **General**.
The **Sheet Print Options** dialog appears.
4. Click the **Output** tab.
5. From the **Output Type** drop-down list box, choose **Cell Range**.
6. Click **OK** to close the **Sheet Print Options** dialog.
7. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.
8. To specify the range of cells,
9. To print the range of cells, follow the instructions in **Printing an Entire Sheet**.

Printing an Area of the Sheet

You may not want to print the entire sheet but only a specified area of the sheet. You can specify an area of the sheet with the **PrintType** ('PrintType Property' in the on-line documentation) property of the **PrintInfo** object or use the **OwnerPrintDraw** ('OwnerPrintDraw Method' in the on-line documentation) method for the control. As with the other printing tasks, such as **Printing an Entire Sheet**, this involves the **PrintSheet** ('PrintSheet Method' in the on-line documentation) method.

Using Code

Set the **PrintType** ('PrintType Property' in the on-line documentation) property and use the **PrintSheet** ('PrintSheet Method' in the on-line documentation) method to print.

Example

This example specifies an area to print.

C#

```
// Create the printer settings object
FarPoint.Win.Spread.PrintInfo printset = new FarPoint.Win.Spread.PrintInfo();
// Allow printing of only 20 columns and 20 rows of cells
printset.ColStart = 1;
printset.ColEnd = 20;
printset.RowStart = 1;
printset.RowEnd = 20;
printset.PrintType = FarPoint.Win.Spread.PrintType.CellRange;
// Assign the printer settings to the sheet and print it
fpSpread1.Sheets[0].PrintInfo = printset;
fpSpread1.PrintSheet(0);
```

VB

```
Dim printset As New FarPoint.Win.Spread.PrintInfo
' Allow printing of only 20 columns and 20 rows of cells
printset.ColEnd = 20
printset.ColStart = 1
printset.RowStart = 1
printset.RowEnd = 20
printset.PrintType = FarPoint.Win.Spread.PrintType.CellRange
' Assign the printer settings to the sheet and print it
fpSpread1.Sheets(0).PrintInfo = printset
fpSpread1.PrintSheet(0)
```

Printing a Sheet with Cell Notes

You can print cell notes as well as the data. Use the **PrintNotes** ('PrintNotes Property' in the on-line documentation) property in the **PrintInfo** ('PrintInfo Class' in the on-line documentation) object to print notes. The notes can be printed on a page either after the data is printed or as displayed on the sheet.

The following figure shows how printing a sticky note as displayed might look.

	A	B	C	D
1		test		
2				
3				
4				A
5				
6				a

For information about adding cell notes, refer to **Adding a Note to a Cell**.

Using Code

1. Use the **PrintNotes** ('**PrintNotes Property**' in the on-line documentation) property to print notes.
2. Use the **PrintSheet** ('**PrintSheet Method**' in the on-line documentation) method to print.

Example

This example prints notes.

C#

```
private void Form1_Load(object sender, System.EventArgs e)
{
    fpSpread1.Sheets[0].RowCount = 20;
    fpSpread1.Sheets[0].ColumnCount = 5;
    fpSpread1.TextTipPolicy = FarPoint.Win.Spread.TextTipPolicy.Fixed;
    fpSpread1.Sheets[0].Cells[0, 0].NoteStyle =
FarPoint.Win.Spread.NoteStyle.StickyNote;
    fpSpread1.Sheets[0].Cells[0, 0].Note = "test";
}

private void button1_Click(object sender, System.EventArgs e)
{
    FarPoint.Win.Spread.PrintInfo pi = new FarPoint.Win.Spread.PrintInfo();
    pi.PrintNotes = FarPoint.Win.Spread.PrintNotes.AsDisplayed;
    fpSpread1.Sheets[0].PrintInfo = pi;
    fpSpread1.PrintSheet(-1);
}
```

VB

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    fpSpread1.Sheets(0).RowCount = 20
    fpSpread1.Sheets(0).ColumnCount = 5
    fpSpread1.TextTipPolicy = FarPoint.Win.Spread.TextTipPolicy.Fixed
    fpSpread1.Sheets(0).Cells(0, 0).NoteStyle =
FarPoint.Win.Spread.NoteStyle.StickyNote
    fpSpread1.Sheets(0).Cells(0, 0).Note = "test"
End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
    Dim pi As New FarPoint.Win.Spread.PrintInfo()
    pi.PrintNotes = FarPoint.Win.Spread.PrintNotes.AsDisplayed
```

```
fpSpread1.Sheets(0).PrintInfo = pi
fpSpread1.PrintSheet(-1)
End Sub
```

Printing a Sheet with Shapes

You can print shapes as well as the data. Use the **PrintShapes ('PrintShapes Property' in the on-line documentation)** property in the **PrintInfo ('PrintInfo Class' in the on-line documentation)** object to include printing the shapes when printing a sheet.

For more information on shapes, refer to **Customizing Drawing**.

Using Code

Use the **PrintShapes ('PrintShapes Property' in the on-line documentation)** property to print shapes.

Example

This example prints shapes.

C#

```
private void button1_Click(object sender, System.EventArgs e)
{
    FarPoint.Win.Spread.PrintInfo pi = new FarPoint.Win.Spread.PrintInfo();
    pi.PrintShapes = true;
    fpSpread1.Sheets[0].PrintInfo = pi;
    fpSpread1.PrintSheet(0);
}
private void Form1_Load(object sender, System.EventArgs e)
{
    FarPoint.Win.Spread.DrawingSpace.ArrowShape arrow = new
    FarPoint.Win.Spread.DrawingSpace.ArrowShape();
    arrow.BackColor = Color.Plum;
    arrow.ForeColor = Color.Pink;
    arrow.SetBounds(0, 0, 200, 100);
    fpSpread1.Sheets[0].AddShape(arrow);
}
```

VB

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
    Dim pi as New FarPoint.Win.Spread.PrintInfo()
    pi.PrintShapes = True
    fpSpread1.Sheets(0).PrintInfo = pi
    fpSpread1.PrintSheet(0)
End Sub
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    Dim arrow As New FarPoint.Win.Spread.DrawingSpace.ArrowShape()
    arrow.BackColor = Color.Plum
    arrow.ForeColor = Color.Pink
    arrow.SetBounds(0, 0, 200, 100)
    fpSpread1.ActiveSheet.AddShape(arrow)
End Sub
```

Printing in Duplex Mode

Printing in duplex mode allows you to print on both sides of the paper.

Using Code

Use the **Duplex ('Duplex Property' in the on-line documentation)** property of the **PrintInfo ('PrintInfo Class' in the on-line documentation)** class to print in duplex mode.

Example

This example shows how to print on both sides of the paper.

C#

```
fpSpread1.ActiveSheet.Cells[1, 1].Value = "1,1";  
fpSpread1.ActiveSheet.Cells[10, 10].Value = "10,10";  
fpSpread1.ActiveSheet.PrintInfo.Duplex = System.Drawing.Printing.Duplex.Default;  
fpSpread1.ActiveSheet.PrintInfo.Preview = true;  
fpSpread1.PrintSheet(fpSpread1.ActiveSheet);
```

VB

```
fpSpread1.ActiveSheet.Cells(1, 1).Value = "1,1"  
fpSpread1.ActiveSheet.Cells(10, 10).Value = "10,10"  
fpSpread1.ActiveSheet.PrintInfo.Duplex = System.Drawing.Printing.Duplex.[Default]  
fpSpread1.ActiveSheet.PrintInfo.Preview = True  
fpSpread1.PrintSheet(fpSpread1.ActiveSheet)
```

 **Note:** Duplex printing mode is not supported if a user tries to print to a PDF file.

Customizing the Appearance of the Printing

There are various aspects of the printing of the spreadsheet that can be customized and most of them reside in the **PrintInfo ('PrintInfo Class' in the on-line documentation)** object. For an overview of the **PrintInfo** object, refer to **Understanding the Printing Options**.

The tasks involved with customizing the printing include:

- **Understanding the Printing Options**
- **Customizing the Print Job Settings**
- **Customizing the Printed Page Layout**
- **Customizing the Printed Page Header or Footer**
- **Customizing the Print Preview Dialog**
- **Repeating Rows or Columns on Printed Pages**
- **Adding a Page Break**
- **Adding a Watermark to a Printed Page**

Most print options are set on the **PrintInfo** object and apply at the sheet level. When you print, you are sending a specified sheet to the printer, which uses those settings. If you want different print settings for different sheets, you may want to reset the **PrintInfo** object and then make the necessary changes between the printing of the sheets.

Understanding the Printing Options

You can customize the printing by setting the properties of a **PrintInfo** ('PrintInfo Class' in the on-line documentation) object and setting the **PrintInfo** ('PrintInfo Property' in the on-line documentation) property of a sheet to that object. The **PrintInfo** ('PrintInfo Class' in the on-line documentation) object has the settings for customizing the printing of a sheet.

The **PrintInfo** ('PrintInfo Class' in the on-line documentation) object provides the following properties for customizing the printing:

Property

AbortMessage ('AbortMessage Property' in the on-line documentation)

BestFitCols ('BestFitCols Property' in the on-line documentation)

BestFitRows ('BestFitRows Property' in the on-line documentation)

Centering ('Centering Property' in the on-line documentation)

Colors ('Colors Property' in the on-line documentation)

ColStart ('ColStart Property' in the on-line documentation) and **ColEnd** ('ColEnd Property' in the on-line documentation)

FirstPageNumber ('FirstPageNumber Property' in the on-line documentation)

Footer ('Footer Property' in the on-line documentation)

Header ('Header Property' in the on-line documentation)

Images ('Images Property' in the on-line documentation)

JobName ('JobName Property' in the on-line documentation)

Margin ('Margin Property' in the on-line documentation)

Opacity ('Opacity Property' in the on-line documentation)

Orientation ('Orientation Property' in the on-line documentation)

Description

Gets or sets the message to display for the abort dialog. See **Displaying an Abort Message for the User**.

Gets or sets whether column widths are adjusted to fit the longest string width for printing. See **Optimizing the Printing Using Size**.

Gets or sets whether row heights are adjusted to fit the tallest string height for printing. See **Optimizing the Printing Using Size**.

Gets or sets whether to center the print out. See **Customizing the Printed Page Layout**.

Gets or sets the list of colors that can be used in a custom header or footer text. See **Customizing the Printed Page Header or Footer**.

Used for printing a portion of the sheet. See **Printing a Range of Cells on a Sheet**.

Gets or sets the page number to print on the first page. See **Customizing the Printed Page Layout**.

Used for providing footers on printed pages. See **Customizing the Printed Page Header or Footer**.

Used for providing headers on printed pages. See **Customizing the Printed Page Header or Footer**.

Gets or sets the list of images that can be used in a custom headers or footers. See **Customizing the Printed Page Header or Footer**.

Gets or sets the name of the print job. See **Customizing the Print Job Settings**.

Gets or sets the margins for printing. See **Customizing the Printed Page Layout**.

Gets or sets the opacity used when printing this sheet; this is used to print a watermark first and then the sheet contents. See **Customizing the Printed Page Layout**.

Gets or sets the page orientation used for printing.

documentation)

PageStart ('PageStart Property' in the on-line documentation) and **PageEnd** ('PageEnd Property' in the on-line documentation)

PageOrder ('PageOrder Property' in the on-line documentation)

PaperSize ('PaperSize Property' in the on-line documentation)

PaperSource ('PaperSource Property' in the on-line documentation)

Preview ('Preview Property' in the on-line documentation)

Printer ('Printer Property' in the on-line documentation)

PrintNotes ('PrintNotes Property' in the on-line documentation)

PrintShapes ('PrintShapes Property' in the on-line documentation)

PrintType ('PrintType Property' in the on-line documentation)

RepeatColStart ('RepeatColStart Property' in the on-line documentation) and **RepeatColEnd** ('RepeatColEnd Property' in the on-line documentation)

RepeatRowStart ('RepeatRowStart Property' in the on-line documentation) and **RepeatRowEnd** ('RepeatRowEnd Property' in the on-line documentation)

RowStart ('RowStart Property' in the on-line documentation) and **RowEnd** ('RowEnd Property' in the on-line documentation)

ShowBorder ('ShowBorder Property' in the on-line documentation)

ShowColor ('ShowColor Property' in the on-line documentation)

ShowColumnHeader ('ShowColumnHeader Property' in the on-line documentation) and **ShowRowHeader** ('ShowRowHeader Property' in the on-line documentation)

ShowGrid ('ShowGrid Property' in the on-line documentation)

ShowPrintDialog ('ShowPrintDialog Property' in the on-line documentation)

See **Customizing the Print Job Settings**.

Used for printing a page range. See **Printing Particular Pages**.

Gets or sets the order in which pages print. See **Customizing the Print Job Settings**.

Gets or sets the paper size to use. See **Customizing the Print Job Settings**.

Gets or sets the paper source to use. See **Customizing the Print Job Settings**.

Used to provide print preview. See **Providing a Preview of the Printing**.

Gets or sets the name of the printer to use for printing. See **Customizing the Print Job Settings**.

Gets or sets whether to print the cell notes. See **Printing a Sheet with Cell Notes**.

Gets or sets whether to print floating objects. See **Printing a Sheet with Shapes**.

Gets or sets what is to be printed. See **Printing Particular Pages**.

Gets or sets whether to print the same set of columns on each page. See **Customizing the Printed Page Header or Footer**.

Gets or sets whether to print the same set of rows on each page. See **Customizing the Printed Page Header or Footer**.

Used for printing a portion of the sheet. See **Printing a Range of Cells on a Sheet**.

Gets or sets whether to print a border around the sheet. See **Customizing the Printed Page Layout**.

Gets or sets whether to print the colors as they appear on the screen. See **Customizing the Printed Page Layout**.

Gets or sets whether to print the column headers and row headers. See **Customizing the Printed Page Layout**.

Gets or sets whether to print the sheet grid lines. See **Customizing the Printed Page Layout**.

Gets or sets whether to display a print dialog before printing. See **Displaying a Print Dialog for the User**.

ShowShadows ('ShowShadows Property' in the on-line documentation)	Gets or sets whether to print the header shadows. See Customizing the Printed Page Layout .
SmartPrintPagesTall ('SmartPrintPagesTall Property' in the on-line documentation)	Gets or sets how many pages tall to print. See Optimizing the Printing Using Rules .
SmartPrintPagesWide ('SmartPrintPagesWide Property' in the on-line documentation)	Gets or sets how many pages wide to print. See Optimizing the Printing Using Rules .
SmartPrintRules ('SmartPrintRules Property' in the on-line documentation)	Used for setting up the rules for optimizing the printing. See Optimizing the Printing Using Rules .
UseMax ('UseMax Property' in the on-line documentation)	Gets or sets whether to print only rows containing data. See Printing an Area of the Sheet .
UseSmartPrint ('UseSmartPrint Property' in the on-line documentation)	Used for turning on the rules for optimizing the printing. See Optimizing the Printing Using Rules .
ZoomFactor ('ZoomFactor Property' in the on-line documentation)	Gets or sets the zoom factor used for printing this sheet. See Customizing the Printed Page Layout .
EnhancePreview ('EnhancePreview Property' in the on-line documentation)	Gets or sets the state to show the enhanced Print Preview Dialog (like Excel). See Customizing the Print Preview Dialog .
ShowPageSetupButton ('ShowPageSetupButton Property' in the on-line documentation)	Gets or sets the state to show the PageSetUpButton on toolbar of the Print Preview Dialog. See Providing a Preview of the Printing .
Duplex ('Duplex Property' in the on-line documentation)	Gets or sets whether to allow a user to print on both sides of the paper. See Printing in Duplex Mode .

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet for which to set the print options.
5. In the properties list, double-click the **PrintInfo** property to display the settings for the **PrintInfo** class.
6. Set the properties listed above as needed to set your print options.
7. Click **OK** to close the editor.

Using a Shortcut

1. Create and set properties for a **PrintInfo** ('PrintInfo Class' in the on-line documentation) object.
2. Set the Sheet shortcut object **PrintInfo** ('PrintInfo Property' in the on-line documentation) property to the **PrintInfo** ('PrintInfo Class' in the on-line documentation) object you just created.

Example

This example code creates a **PrintInfo** ('PrintInfo Class' in the on-line documentation) object, sets properties to specify to not print grid lines or row headers, and to print only cells with data in them.

C#

```
// Create PrintInfo object and set properties.
```

```
FarPoint.Win.Spread.PrintInfo printset = new FarPoint.Win.Spread.PrintInfo();
printset.ShowGrid = false;
printset.ShowRowHeader = FarPoint.Win.Spread.PrintHeader.Hide;
printset.UseMax = true;
// Set the PrintInfo property for the first sheet.
fpSpread1.Sheets[0].PrintInfo = printset;
// Print the sheet.
fpSpread1.PrintSheet(0);
```

VB

```
' Create PrintInfo object and set properties.
Dim printset As New FarPoint.Win.Spread.PrintInfo()
printset.ShowGrid = False
printset.ShowRowHeader = FarPoint.Win.Spread.PrintHeader.Hide
printset.UseMax = True
' Set the PrintInfo property for the first sheet.
fpSpread1.Sheets(0).PrintInfo = printset
' Print the sheet.
fpSpread1.PrintSheet(0)
```

Using Code

1. Create and set properties for a **PrintInfo ('PrintInfo Class' in the on-line documentation)** object.
2. Set the SheetView object **PrintInfo ('PrintInfo Property' in the on-line documentation)** property to the **PrintInfo ('PrintInfo Class' in the on-line documentation)** object you just created.

Example

This example code creates a **PrintInfo ('PrintInfo Class' in the on-line documentation)** object, sets properties to specify to not print grid lines or row headers, and to print only cells with data in them.

C#

```
// Create PrintInfo object and set properties.
FarPoint.Win.Spread.PrintInfo printset = new FarPoint.Win.Spread.PrintInfo();
printset.ShowGrid = false;
printset.ShowRowHeader = FarPoint.Win.Spread.PrintHeader.Hide;
printset.UseMax = true;
// Create SheetView object and assign it to the first sheet.
FarPoint.Win.Spread.SheetView SheetToPrint = new FarPoint.Win.Spread.SheetView();
SheetToPrint.PrintInfo = printset;
fpSpread1.Sheets[0] = SheetToPrint;
// Print the sheet.
fpSpread1.PrintSheet(0);
```

VB

```
' Create PrintInfo object and set properties.
Dim printset As New FarPoint.Win.Spread.PrintInfo()
printset.ShowGrid = False
printset.ShowRowHeader = FarPoint.Win.Spread.PrintHeader.Hide
printset.UseMax = True
' Create SheetView object and assign it to the first sheet.
Dim SheetToPrint As New FarPoint.Win.Spread.SheetView()
SheetToPrint.PrintInfo = printset
```

```
fpSpread1.Sheets(0) = SheetToPrint  
' Print the sheet.  
fpSpread1.PrintSheet(0)
```

Using the Spread Designer

1. Select the sheet tab for the sheet for which you want to set print settings.
2. Select the **Page Layout** option.
3. Select the PrintTitles icon, choose **General**.
The **Sheet Print** dialog appears.
4. Set the print options using the **General, Output, Header/Footer, SmartPrint, Pagination, and Margins** tab options.
5. Click **OK** to close the **Sheet Print Options** dialog.
6. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Customizing the Print Job Settings

Sheets are printed on the current default printer in your Windows environment unless you specify otherwise. You can print sheets on any Windows-supported printer.

The print job settings that you can customize include printer, source, page size and collated settings. Set the **Printer** (**'Printer Property' in the on-line documentation**), **PaperSource** (**'PaperSource Property' in the on-line documentation**), or **PaperSize** (**'PaperSize Property' in the on-line documentation**) on the **PrintInfo** object and Collated settings on the PageSetup object.

Using Code

1. Set various print job settings.
2. Print the sheet.

Example

This example code sets the paper source based on a selection from a combo box and sets the paper size before printing all the sheets.

C#

```
private void Form1_Load(object sender, System.EventArgs e)  
{  
    int i;  
    System.Drawing.Printing.PrinterSettings ps = new  
System.Drawing.Printing.PrinterSettings();  
  
    for (i = 0; i <= ps.PaperSources.Count - 1; i++)  
    {  
        comboBox1.Items.Add(ps.PaperSources[i].SourceName);  
    }  
    comboBox1.Text = ps.PaperSources[0].SourceName;  
}  
  
private void button1_Click(object sender, System.EventArgs e)  
{  
    FarPoint.Win.Spread.PrintInfo pi = new FarPoint.Win.Spread.PrintInfo();  
    System.Drawing.Printing.PrinterSettings ps = new  
System.Drawing.Printing.PrinterSettings();
```

```
pi.PaperSize = new System.Drawing.Printing.PaperSize("Letter", 600, 300);
pi.PaperSource = ps.PaperSources[comboBox1.SelectedIndex];
fpSpread1.Sheets[0].PrintInfo = pi;
fpSpread1.PrintSheet(-1);
}
```

VB

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    Dim ps As New System.Drawing.Printing.PrinterSettings()
    Dim i As Integer
    For i = 0 To ps.PaperSources.Count - 1
        ComboBox1.Items.Add(ps.PaperSources.Item(i).SourceName)
    Next
    ComboBox1.Text = ps.PaperSources.Item(0).SourceName
End Sub
```

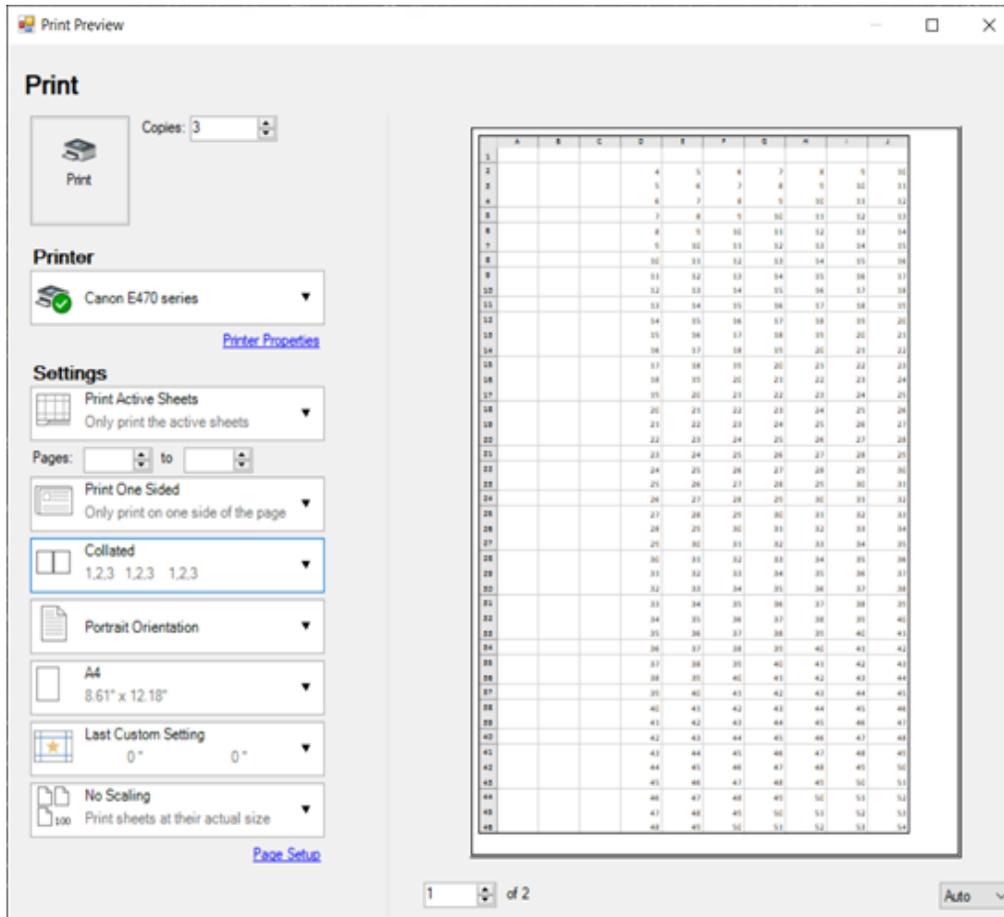
```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
    Dim pi As New FarPoint.Win.Spread.PrintInfo()
    Dim ps As New System.Drawing.Printing.PrinterSettings()
    pi.PaperSize = New System.Drawing.Printing.PaperSize("Letter", 600, 300)
    pi.PaperSource = ps.PaperSources(ComboBox1.SelectedIndex)
    fpSpread1.Sheets(0).PrintInfo = pi
    fpSpread1.PrintSheet(-1)
End Sub
```

Set the Collated Printing

In printing, the term **Collated** refers to printing the first copy in the same order from start to finish, followed by printing the next copy in the same order, whereas **Uncollated** refers to printing multiple copies of the first page of a document, then multiple copies of the second page, and so on.

In Spread.NET, the **Collated** property when set as true uses **IPageSetup** object to implement the collated printing of the spreadsheet.

The following image shows the Collated option in the Print settings that appears on your screen.



Example

Collated printing can be useful for creating employee training materials whereas Uncollated option can be useful for creating separate piles of each page from the document.

Refer to the following code to implement an uncollected printing of the Spreadsheet.

C#

```

IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;
    for (int i = 1; i < 50; i++)
    {
        for (int j = 3; j < 10; j++)
        {
            TestActiveSheet.Cells[i, j].Value = i + j;
        }
    }
    GrapeCity.Spreadsheet.Printing.IPageSetup pageSetup =
fpSpread1.AsWorkbook().ActiveSheet.PageSetup;
    pageSetup.Collated = false;
    pageSetup.NumberOfCopies = 3;
    fpSpread1.Sheets[0].PrintInfo.Preview = true;
    fpSpread1.Sheets[0].PrintInfo.EnhancePreview = true;
    fpSpread1.PrintSheet(0);
    
```

VB

```
Private Sub SurroundingSub()  
    Dim TestActiveSheet As IWorksheet = fpSpread1.AsWorkbook().ActiveSheet  
    For i = 1 To 49  
        For j = 3 To 9  
            TestActiveSheet.Cells(i, j).Value = i + j  
        Next  
    Next  
    Dim pageSetup As GrapeCity.Spreadsheet.Printing.IPageSetup =  
fpSpread1.AsWorkbook().ActiveSheet.PageSetup  
    pageSetup.Collated = False  
    pageSetup.NumberOfCopies = 3  
    fpSpread1.Sheets(0).PrintInfo.Preview = True  
    fpSpread1.Sheets(0).PrintInfo.EnhancePreview = True  
    fpSpread1.PrintSheet(0)  
End Sub
```



The Collated/Uncollated option cannot be used for printing PDFs, as NumberOfCopies property is not applicable for PDF.

Customizing the Printed Page Layout

You can customize the layout of the printed page in any of several ways. The layout settings that you can customize include: color, borders, grid lines and opacity, centering and margins, orientation, shadows, and zooming. These customizations are possible with properties of the **PrintInfo** object.

Setting the Printed Page Color, Border, and Grid

There are several appearance customizations that can be set for the printed page, including setting the color, the border, and the grid.

You can set the **PrintInfo ShowColor** (**ShowColor Property** in the on-line documentation), **ShowBorder** (**ShowBorder Property** in the on-line documentation), and **ShowGrid** (**ShowGrid Property** in the on-line documentation) properties.

Setting the Printed Page Orientation

There are several ways to control the orientation (landscape or portrait) of the page when printing.

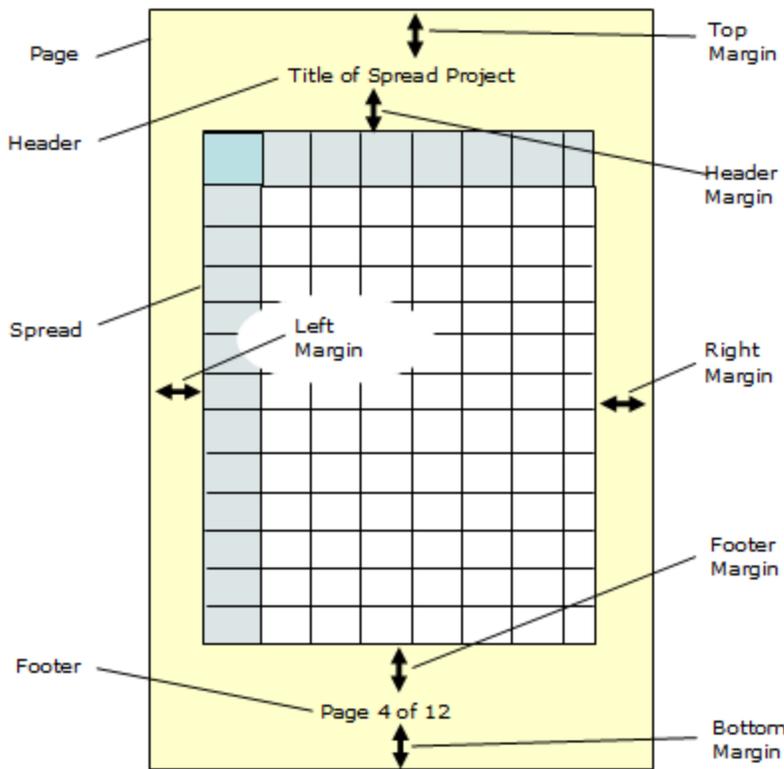
You can set the **PrintInfo Orientation** (**Orientation Property** in the on-line documentation) property.

You can use LandscapeRule, which is a SmartPrint rule. For more information, refer to **Optimizing the Printing Using Rules**.

Setting the Printed Page White Space

You can customize the margins of the printed page and you can center the printed page.

Refer to the following figure for definitions of the properties of the **PrintMargin** (**PrintMargin Class** in the on-line documentation) object.



The **PrintInfo Margin** ('Margin Property' in the on-line documentation) property uses a **PrintMargin** ('PrintMargin Class' in the on-line documentation) object. When you use this object in code, the units of measure are 100-ths of an inch, so $3/4$ of an inch would be 75.

When you use the Spread Designer and set the margins in the **Sheet > Print Settings > Margins** tab, the units of measure are inches, so $3/4$ of an inch would be 0.75.

There are several ways to control the justification of contents on the page when printing.

You can set the **PrintInfo Centering** ('Centering Property' in the on-line documentation) property.

Creating a Shadow for the Printed Page

One of the customizations of the page layout is the ability to print shadows.

You can set the **PrintInfo ShowShadows** ('ShowShadows Property' in the on-line documentation) property.

Scaling the Printing

There are several ways to control the scaling of the printing.

You can set the **PrintInfo ZoomFactor** ('ZoomFactor Property' in the on-line documentation) property.

The **ZoomFactor** and the zoom within the preview are different scaling mechanisms. The **ZoomFactor** in the layout effects the size of the actual display and the print out size. The zoom within the print preview dialog is simply a temporary zoom in the display but is not saved for any printing.

Currently there is no means of adjusting the default zoom in the preview.

You can use **ScaleRule**, which is a **SmartPrint** rule. For more information, refer to **Optimizing the Printing Using Rules**.

Customizing the Printed Page Header or Footer

You can provide header and footers that appears on the printed pages. Using the **Header ('Header Property' in the on-line documentation)** property and **Footer ('Footer Property' in the on-line documentation)** property of the **PrintInfo ('PrintInfo Class' in the on-line documentation)** class, which may include special control commands, you can specify text and variables, such as page numbers, as well as specify the font settings. The font related commands begin with "f".

The control commands that can be inserted in headers and footers are listed in this table:

Control Character	Full Command	Action in Printed Page Header or Footer
/	/	Inserts a literal forward slash character (/)
/c	/c	Center justifies the item
/cl	/cl"n"	Sets the font color for text, with the zero-based index of the color, n, in quotes (n can be 0 or more); see the Colors ('Colors Property' in the on-line documentation) property
/dl	/dl	Inserts the date, using the long form
/ds	/ds	Inserts the date, using the short form
/f	/f"n"	Recalls the previously saved font settings (see /fs in this table), with the zero-based index, n, in quotes (n can be 0 or more)
/fb	/fbo	Turns off bold font type
	/fb1	Turns on bold font type
/fi	/fio	Turns off italics font type
	/fi1	Turns on italics font type
/fk	/fko	Turns off strikethrough
	/fk1	Turns on strikethrough
/fn	/fn"name"	Sets the name of the font face, with the name of the font in quotes
/fs	/fs"n"	Saves the font settings for re-use, with the zero-based index of the font settings, n, in quotes (see /f in this table)
/fu	/fu0	Turns off underline
	/fu1	Turns on underline
/fz	/fz"n"	Set the size of the font
/g	/g"n"	Inserts a graphic (image), with the zero-based index of the image, n, in quotes (n can be zero or more); see the Images ('Images Property' in the on-line documentation) property
/l	/l	Left justifies the item (that is the letter l or L, as in Left)
/n	/n	Inserts a new line
/p	/p	Inserts a page number
/pc	/pc	Inserts a page count (the total number of pages in the print job)
/r	/r	Right justifies the item
/sn	/sn	Inserts the sheet name

/tl /tl Inserts the time, using the long form
 /ts /ts Inserts the time, using the short form

If you use multiple control characters, do not put spaces between them. The letters can be lower or upper case; all commands and examples are shown here in lower case for simplicity.

Define the headers and footers (set the **Header ('Header Property' in the on-line documentation)** and **Footer ('Footer Property' in the on-line documentation)** properties) before printing the sheet (running the **PrintSheet ('PrintSheet Method' in the on-line documentation)** method).

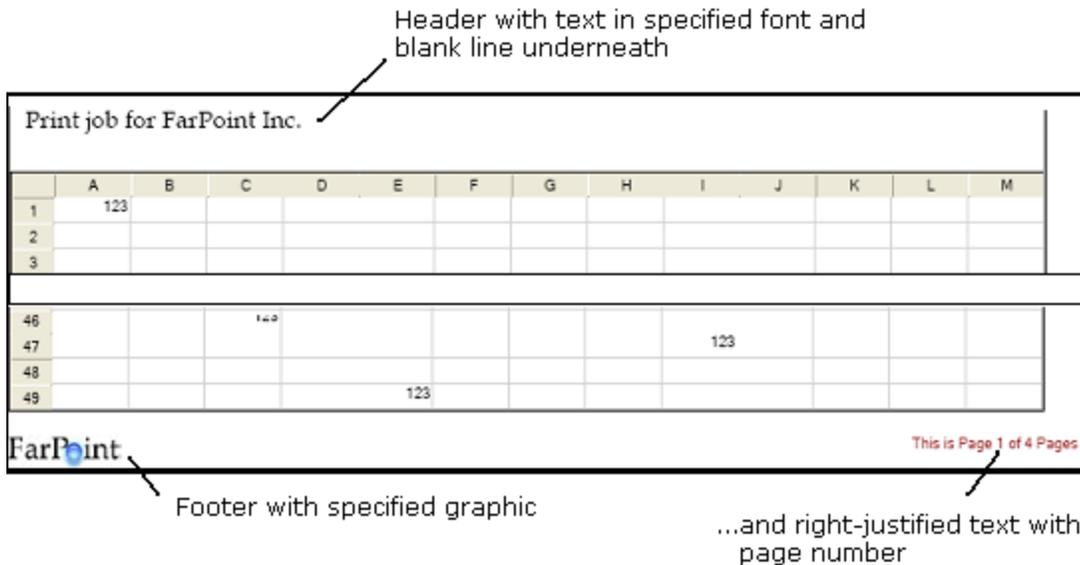
You can specify a color for the text from a list of colors if the color is previously defined in the **Colors** property.

You can specify an image if the image is previously defined in the **Images** property.

You can add text including the page number and the total number of pages printed.

You can save the font settings to re-use them later in the header or footer.

Here is the result of the example code given below.



Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Members** list, select the sheet for which to set the header and footer text.
5. In the properties list, double-click the **PrintInfo** property to display the settings for the **PrintInfo** class.
6. Set the **Header** and **Footer** properties to set the header and footer text.
7. Click **OK** to close the editor.

Using a Shortcut

1. Create and set the **Header ('Header Property' in the on-line documentation)** and **Footer ('Footer Property' in the on-line documentation)** properties for a **PrintInfo ('PrintInfo Class' in the on-line documentation)** object.
2. Set the Sheet shortcut object **PrintInfo ('PrintInfo Property' in the on-line documentation)** property to the **PrintInfo ('PrintInfo Class' in the on-line documentation)** object you just created.

Example

This example code prints the sheet with the specified header and footer text.

C#

```
// Create PrintInfo object and set properties.
FarPoint.Win.Spread.PrintInfo printset = new FarPoint.Win.Spread.PrintInfo();
printset.Colors = new Color[] {Color.Green, Color.Yellow, Color.Gold, Color.Indigo,
Color.Brown};
printset.Images = new Image[] {Image.FromFile("C:\\images\\point.jpg"),
Image.FromFile("C:\\images\\logo.gif"), Image.FromFile("C:\\images\\icon.jpg")};
printset.Header = "/fn\"Book Antiqua\" /fz\"14\" Print job for FarPoint Inc./n ";
printset.Footer = "/g\"1\"/r/cl\"4\"This is page /p of /pc";
// Set the PrintInfo property for the first sheet.
fpSpread1.Sheets[0].PrintInfo = printset;
// Print the sheet.
fpSpread1.PrintSheet(0);
```

VB

```
' Create PrintInfo object and set properties.
Dim printset As New FarPoint.Win.Spread.PrintInfo()
printset.Colors = New Color() {Color.Green, Color.Yellow, Color.Gold, Color.Indigo,
Color.Brown}
printset.Images = New Image() {Image.FromFile("D:\images\point.jpg"),
Image.FromFile("D:\images\logo.gif"), Image.FromFile("C:\images\icon.jpg")}
printset.Header = "/fn\"Book Antiqua\" /fz\"14\" Print job for FarPoint Inc./n "
printset.Footer = "/g\"1\"/r/cl\"4\"This is page /p of /pc"
' Set the PrintInfo property for the first sheet.
fpSpread1.Sheets(0).PrintInfo = printset
' Print the sheet.
fpSpread1.PrintSheet(0)
```

Using Code

1. Create and set the **Header ('Header Property' in the on-line documentation)** and **Footer ('Footer Property' in the on-line documentation)** properties for a **PrintInfo ('PrintInfo Class' in the on-line documentation)** object.
2. Set the SheetView object **PrintInfo ('PrintInfo Property' in the on-line documentation)** property to the **PrintInfo ('PrintInfo Class' in the on-line documentation)** object you just created.

Example

This example code prints the sheet with the specified header and footer colors and images.

C#

```
// Create PrintInfo object and set properties.
FarPoint.Win.Spread.PrintInfo pi = new FarPoint.Win.Spread.PrintInfo();
pi.Footer = "This is Page /p/nof /pc Pages";
pi.Header = "Print Job For /nFPT Inc.";
pi.Colors = new Color[] {Color.Red, Color.Blue};
pi.Images = new Image[] {Image.FromFile("D:\Corporate.jpg"),
Image.FromFile("D:\Building.jpg")};
pi.RepeatColEnd = 25;
```

```

pi.RepeatColStart = 1;
pi.RepeatRowEnd = 25;
pi.RepeatRowStart = 1;
fpSpread1.ActiveSheet.PrintInfo = pi;

```

VB

```

' Create PrintInfo object and set properties.
Dim pi As New FarPoint.Win.Spread.PrintInfo
pi.Footer = "This is Page /p/nof /pc Pages"
pi.Header = "Print Job For /nFPT Inc."
pi.Colors = New Color() {Color.Red, Color.Blue}
pi.Images = New Image() {Image.FromFile("D:\Corporate.jpg"),
Image.FromFile("D:\Building.jpg")}
pi.RepeatColEnd = 25
pi.RepeatColStart = 1
pi.RepeatRowEnd = 25
pi.RepeatRowStart = 1
fpSpread1.ActiveSheet.PrintInfo = pi

```

Using Code

1. Create and set the **Header ('Header Property' in the on-line documentation)** and **Footer ('Footer Property' in the on-line documentation)** properties for a **PrintInfo ('PrintInfo Class' in the on-line documentation)** object.
2. Set the SheetView object **PrintInfo ('PrintInfo Property' in the on-line documentation)** property to the **PrintInfo ('PrintInfo Class' in the on-line documentation)** object you just created.

Example

This example code prints the sheet with the specified header and footer text.

C#

```

// Create PrintInfo object and set properties.
FarPoint.Win.Spread.PrintInfo printset = new FarPoint.Win.Spread.PrintInfo();
printset.Header = "/lJobName";
printset.Footer = "/r/p of /pc";
// Create SheetView object and assign it to the first sheet.
FarPoint.Win.Spread.SheetView SheetToPrint = new FarPoint.Win.Spread.SheetView();
SheetToPrint.PrintInfo = printset;
// Set the PrintInfo property for the first sheet.
fpSpread1.Sheets[0] = SheetToPrint;
// Print the sheet.
fpSpread1.PrintSheet(0);

```

VB

```

' Create PrintInfo object and set properties.
Dim printset As New FarPoint.Win.Spread.PrintInfo()
printset.Header = "/lJobName"
printset.Footer = "/r/p of /pc"
' Create SheetView object and assign it to the first sheet.
Dim SheetToPrint As New FarPoint.Win.Spread.SheetView()
SheetToPrint.PrintInfo = printset
fpSpread1.Sheets(0) = SheetToPrint

```

```
' Set the PrintInfo property for the first sheet.  
fpSpread1.Sheets(0).PrintInfo = printset  
' Print the sheet.  
fpSpread1.PrintSheet(0)
```

Using the Spread Designer

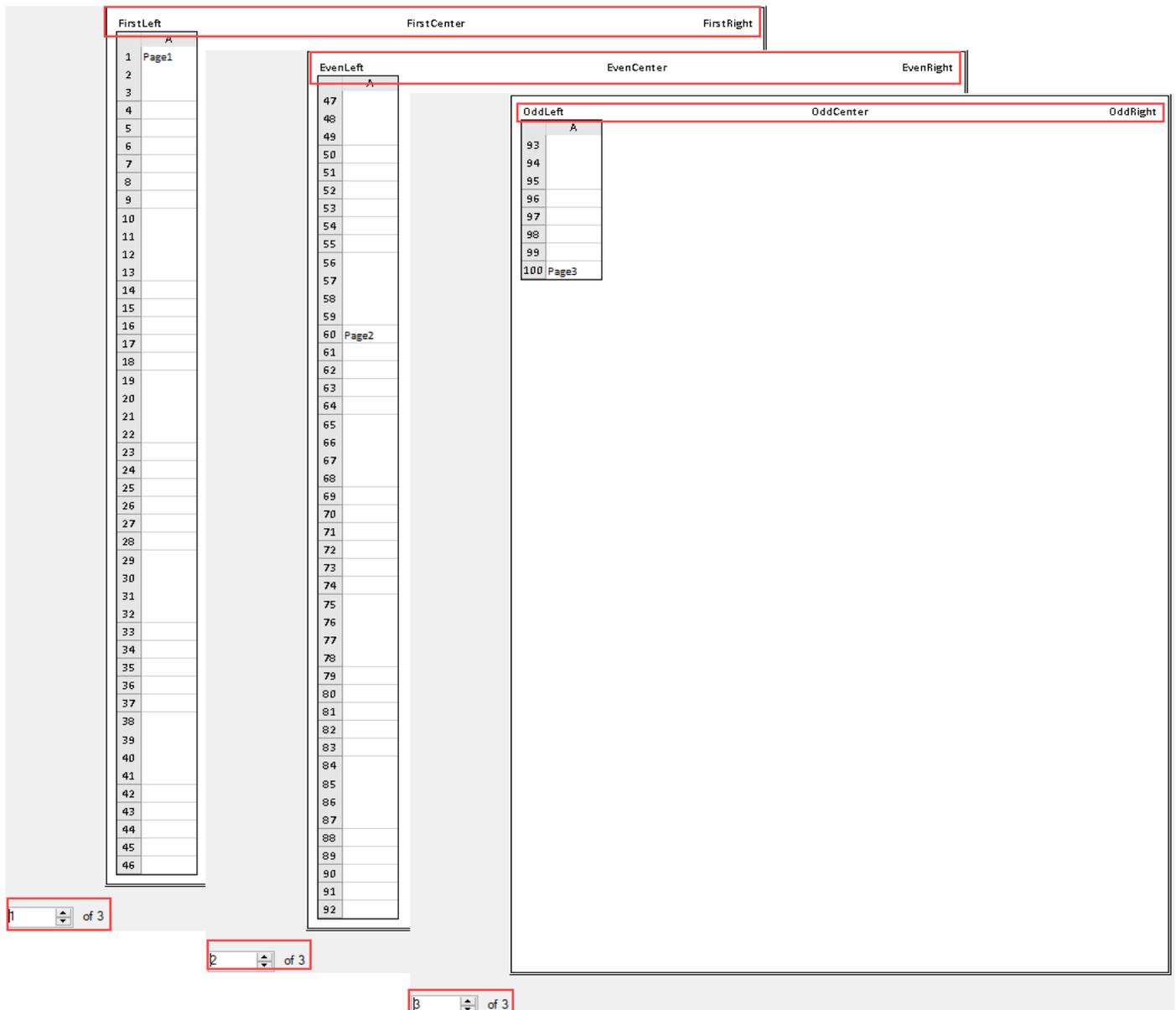
1. Select the sheet tab for the sheet for which you want to set print settings.
2. Select the **Page Layout** option.
3. Choose **Print Titles**.
The **Sheet Print** dialog appears.
4. Click the **Header/Footer** tab.
5. Select the **Header** or **Footer** radio button to indicate whether you are setting the header or footer text.
6. Type the text for your header or footer in the text box.
You can insert control characters in your text by selecting them from the **Control Characters** drop-down list box below the text box and then clicking the **Add** button.
7. Click **OK** to close the **Sheet Print Options** dialog.
8. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Different Header/Footer for Different Printed Pages

You can also add different headers and footers on the first page, last page, odd pages and even pages by setting the **OddAndEvenPagesHeaderFooter** and **DifferentFirstPageHeaderFooter** properties of **PrintInfo** class to true.

On setting the **OddAndEvenPagesHeaderFooter** property to true, Spread.NET allows you to set the different value for header and footer using **EvenFooter**, **FirstFooter**, **FirstHeader**, and other properties.

The following image shows how different headers and footers appear on different printed pages.



Refer to the following code to implement the different header and footer using **PrintInfo** class.

C#

```
// Different header/footer printing
IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;
TestActiveSheet.Cells["A1"].Value = "Page1";
TestActiveSheet.Cells["A60"].Value = "Page2";
TestActiveSheet.Cells["A100"].Value = "Page3";
fpSpread1.ActiveSheet.PrintInfo.Colors = new System.Drawing.Color[] {
System.Drawing.Color.Purple };
fpSpread1.ActiveSheet.PrintInfo.FirstColors = new System.Drawing.Color[] {
System.Drawing.Color.Blue };
fpSpread1.ActiveSheet.PrintInfo.EvenColors = new System.Drawing.Color[] {
System.Drawing.Color.Red };
fpSpread1.ActiveSheet.PrintInfo.Footer = "/1/c1\"0\"OddLeft/cOddCenter/rOddRight";
```

```
// Set DifferentFirstPageHeaderFooter property to true for different first and last
page header footer
fpSpread1.ActiveSheet.PrintInfo.DifferentFirstPageHeaderFooter = true;
fpSpread1.ActiveSheet.PrintInfo.FirstFooter =
"/lFirstLeft/c/cl\"0\"FirstCenter/rFirstRight";
// Set OddAndEvenPagesHeaderFooter property to true for different Odd and Even page
headers
fpSpread1.ActiveSheet.PrintInfo.OddAndEvenPagesHeaderFooter = true;
fpSpread1.ActiveSheet.PrintInfo.EvenFooter =
"/lEvenLeft/cEvenCenter/r/cl\"0\"EvenRight";
fpSpread1.Sheets[0].PrintInfo.Preview = true;
fpSpread1.Sheets[0].PrintInfo.EnhancePreview = true;
fpSpread1.Sheets[0].PrintInfo.ShowColor = true;
fpSpread1.PrintSheet(0);
```

VB

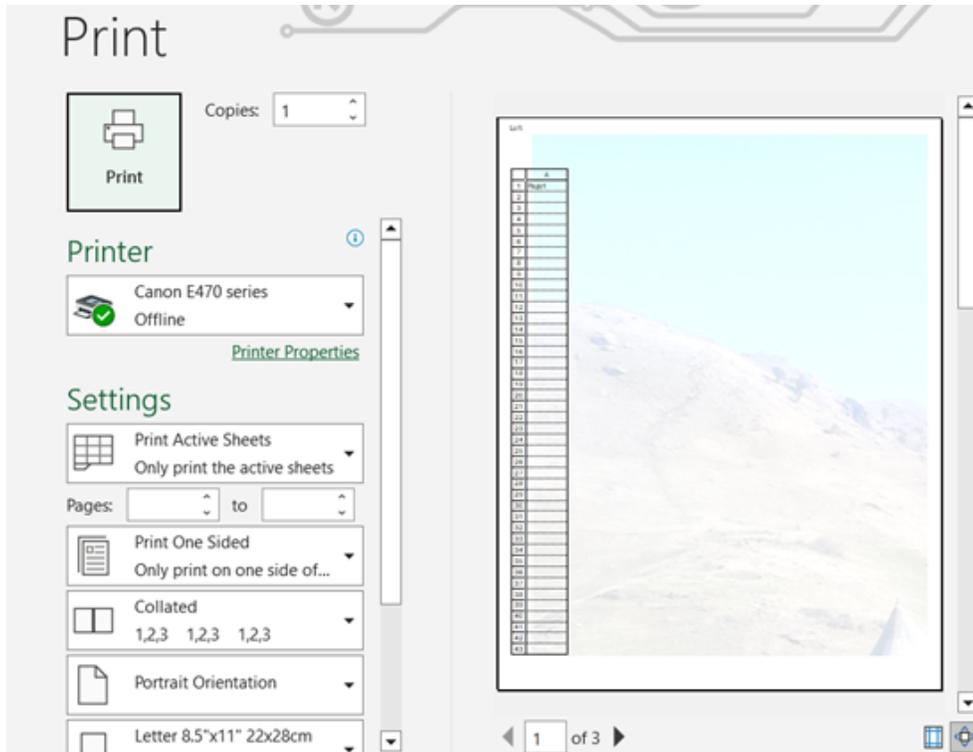
```
' Different header/footer printing
Dim TestActiveSheet As IWorksheet = fpSpread1.AsWorkbook().ActiveSheet
TestActiveSheet.Cells("A1").Value = "Page1"
TestActiveSheet.Cells("A60").Value = "Page2"
TestActiveSheet.Cells("A100").Value = "Page3"
fpSpread1.ActiveSheet.PrintInfo.Colors = New Drawing.Color() {Drawing.Color.Purple}
fpSpread1.ActiveSheet.PrintInfo.FirstColors = New Drawing.Color() {Drawing.Color.Blue}
fpSpread1.ActiveSheet.PrintInfo.EvenColors = New Drawing.Color() {Drawing.Color.Red}
fpSpread1.ActiveSheet.PrintInfo.Footer = "/l/cl\"0\"OddLeft/cOddCenter/rOddRight"
' Set DifferentFirstPageHeaderFooter property to true for different first and last page
header footer
fpSpread1.ActiveSheet.PrintInfo.DifferentFirstPageHeaderFooter = True
fpSpread1.ActiveSheet.PrintInfo.FirstFooter =
"/lFirstLeft/c/cl\"0\"FirstCenter/rFirstRight"
' Set OddAndEvenPagesHeaderFooter property to true for different Odd and Even page
headers
fpSpread1.ActiveSheet.PrintInfo.OddAndEvenPagesHeaderFooter = True
fpSpread1.ActiveSheet.PrintInfo.EvenFooter =
"/lEvenLeft/cEvenCenter/r/cl\"0\"EvenRight"
fpSpread1.Sheets(0).PrintInfo.Preview = True
fpSpread1.Sheets(0).PrintInfo.EnhancePreview = True
fpSpread1.Sheets(0).PrintInfo.ShowColor = True
fpSpread1.PrintSheet(0)
```

PageSetup class also supports **OddAndEvenPagesHeaderFooter** and **DifferentFirstPageHeaderFooter** properties. However, if you are using the **PageSetup** class, you should only use the properties included in this class to set the values for headers and footers.

Export/Import of Excel with an Image in Header or Footer

Spread.NET also supports export/import of Excel with the image in the headers and footers of the print page.

The following image shows the print preview of exported Excel with image as a watermark.



Refer to the following code to add image in the header and export the Spread to Excel.

C#

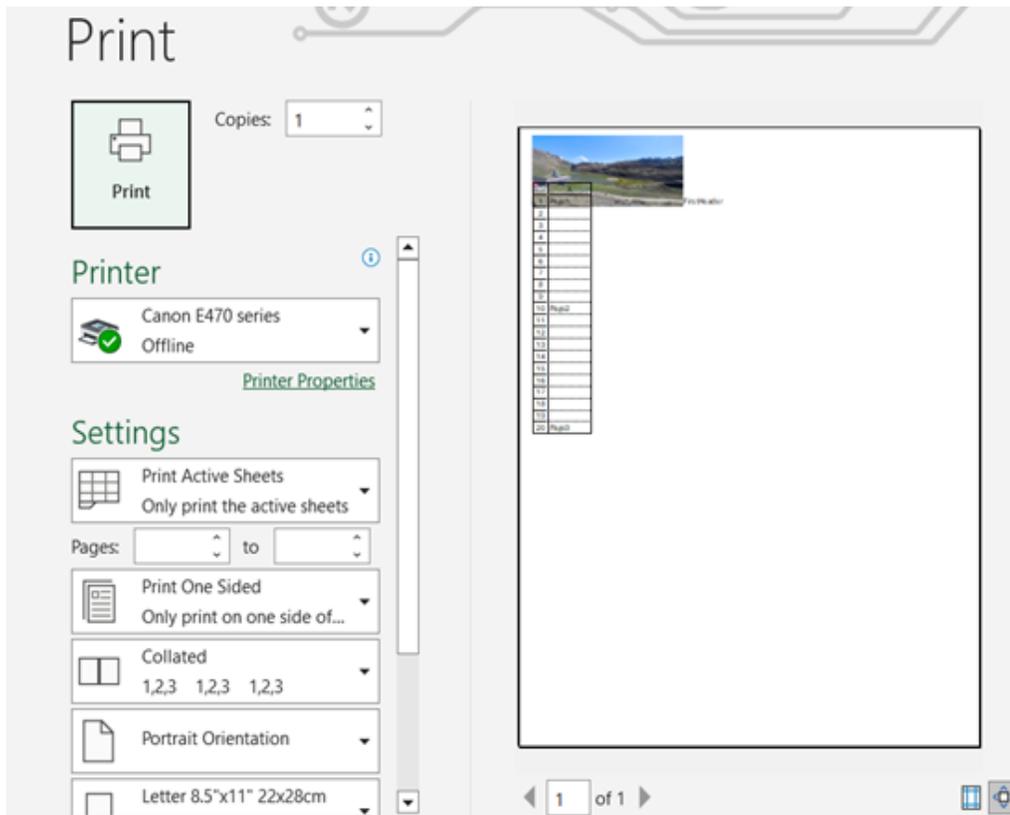
```
TestActiveSheet.PageSetup.LeftHeaderPicture.FileName = @"Image.jpg";
TestActiveSheet.PageSetup.LeftHeader = "Left&G";
TestActiveSheet.PageSetup.TopMargin = 1;
TestActiveSheet.PageSetup.LeftHeaderPicture.ColorType = PictureColorType.Watermark;
TestActiveSheet.PageSetup.LeftHeaderPicture.CropLeft = -40;
TestActiveSheet.PageSetup.LeftHeaderPicture.CropTop = 30;
fpSpread1.ActiveSheet.PrintInfo.Preview = true;
fpSpread1.ActiveSheet.PrintInfo.ShowColor = true;
fpSpread1.ActiveSheet.PrintInfo.EnhancePreview = true;
fpSpread1.PrintSheet(fpSpread1.ActiveSheet);
fpSpread1.SaveExcel(@"excel.xlsx", FarPoint.Excel.ExcelSaveFlags.UseOOXMLFormat);
```

VB

```
TestActiveSheet.PageSetup.LeftHeaderPicture.FileName = "Image.jpg"
TestActiveSheet.PageSetup.LeftHeader = "Left&G"
TestActiveSheet.PageSetup.TopMargin = 1
TestActiveSheet.PageSetup.LeftHeaderPicture.ColorType = PictureColorType.Watermark
TestActiveSheet.PageSetup.LeftHeaderPicture.CropLeft = -40
TestActiveSheet.PageSetup.LeftHeaderPicture.CropTop = 30
fpSpread1.ActiveSheet.PrintInfo.Preview = True
fpSpread1.ActiveSheet.PrintInfo.ShowColor = True
fpSpread1.ActiveSheet.PrintInfo.EnhancePreview = True
fpSpread1.PrintSheet(fpSpread1.ActiveSheet)
fpSpread1.SaveExcel("excel.xlsx", FarPoint.Excel.ExcelSaveFlags.UseOOXMLFormat)
```

However, if you have more than one image in the header or footer, only the first image used is exported to Excel in the header or Footer, as Excel supports only one image.

The following image shows the print preview of an exported Excel file where the first used image is placed in the header.



Refer to the following code to add multiple images in the header and export the Spread to Excel.

C#

```
fpSpread1.ActiveSheet.PrintInfo.OddAndEvenPagesHeaderFooter = true;
fpSpread1.ActiveSheet.PrintInfo.Margin.Top = 100;
fpSpread1.ActiveSheet.PrintInfo.EvenImages = new System.Drawing.Image[] {
    System.Drawing.Image.FromFile("Image.png"), System.Drawing.Image.FromFile("Image.tiff")
};
fpSpread1.ActiveSheet.PrintInfo.EvenHeader = "/l/g\"0\"/g\"1\"EventHeader";
fpSpread1.ActiveSheet.PrintInfo.DifferentFirstPageHeaderFooter = true;
fpSpread1.ActiveSheet.PrintInfo.FirstImages = new System.Drawing.Image[] {
    System.Drawing.Image.FromFile("Image.jpg"), System.Drawing.Image.FromFile("Image.ico")
};
fpSpread1.ActiveSheet.PrintInfo.FirstHeader = "/l/g\"0\"/g\"1\"FirstHeader";
fpSpread1.ActiveSheet.PrintInfo.Images = new System.Drawing.Image[] {
    System.Drawing.Image.FromFile("Image.bmp"), System.Drawing.Image.FromFile("Image.jpeg")
};
fpSpread1.ActiveSheet.PrintInfo.Header = "/l/g\"0\"/g\"1\"OddHeader";
fpSpread1.ActiveSheet.PrintInfo.Preview = true;
fpSpread1.ActiveSheet.PrintInfo.ShowColor = true;
fpSpread1.ActiveSheet.PrintInfo.EnhancePreview = true;
// fpSpread1.PrintSheet(fpSpread1.ActiveSheet);
fpSpread1.SaveExcel(@"excel_printinfo.xlsx", ExcelSaveFlags.UseOOXMLFormat);
```

VB

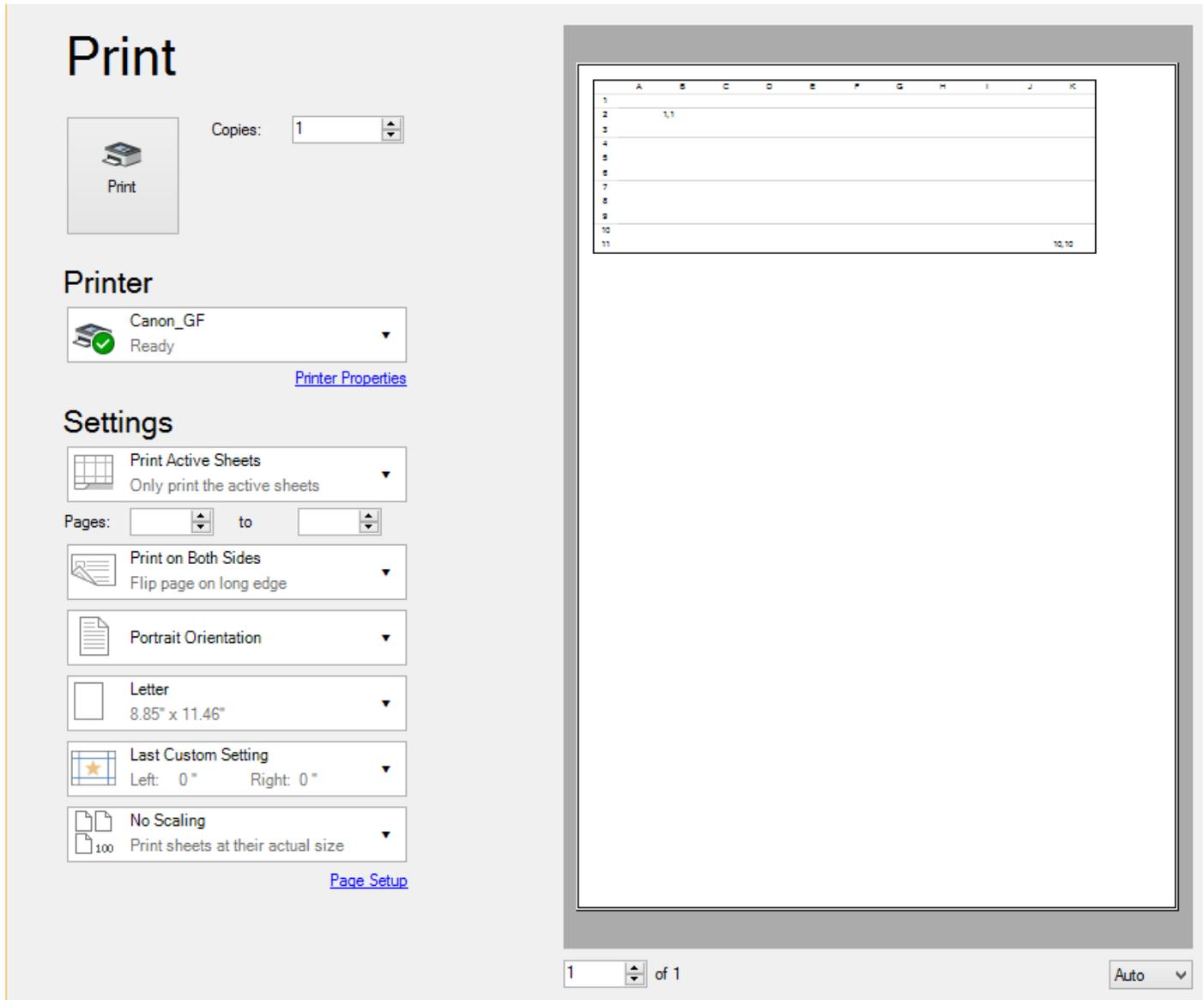
```
fpSpread1.ActiveSheet.PrintInfo.OddAndEvenPagesHeaderFooter = True
```

```
fpSpread1.ActiveSheet.PrintInfo.Margin.Top = 100
fpSpread1.ActiveSheet.PrintInfo.EvenImages = New Drawing.Image()
{Drawing.Image.FromFile("Image.png"), Drawing.Image.FromFile("Image.tiff")}
fpSpread1.ActiveSheet.PrintInfo.EvenHeader = "/l/g""0""/g""1""EventHeader"
fpSpread1.ActiveSheet.PrintInfo.DifferentFirstPageHeaderFooter = True
fpSpread1.ActiveSheet.PrintInfo.FirstImages = New Drawing.Image()
{Drawing.Image.FromFile("Image.jpg"), Drawing.Image.FromFile("Image.ico")}
fpSpread1.ActiveSheet.PrintInfo.FirstHeader = "/l/g""0""/g""1""FirstHeader"
fpSpread1.ActiveSheet.PrintInfo.Images = New Drawing.Image()
{Drawing.Image.FromFile("Image.bmp"), Drawing.Image.FromFile("Image.jpeg")}
fpSpread1.ActiveSheet.PrintInfo.Header = "/l/g""0""/g""1""OddHeader"
fpSpread1.ActiveSheet.PrintInfo.Preview = True
fpSpread1.ActiveSheet.PrintInfo.ShowColor = True
fpSpread1.ActiveSheet.PrintInfo.EnhancePreview = True
' fpSpread1.PrintSheet(fpSpread1.ActiveSheet);
fpSpread1.SaveExcel("excel_printinfo.xlsx", ExcelSaveFlags.UseOOXMLFormat)
```

Customizing the Print Preview Dialog

You can customize the Print Preview settings to show the **Print Preview Dialog like Excel** by using the **EnhancePreview ('EnhancePreview Property' in the on-line documentation)** property of the **PrintInfo ('PrintInfo Class' in the on-line documentation)** class.

The following image shows the printing options that appear on your screen after setting the EnhancePreview property to true.



Using Code

Set the **EnhancePreview** ('**EnhancePreview Property**' in the **on-line documentation**) property of the **PrintInfo** ('**PrintInfo Class**' in the **on-line documentation**) class to true in order to customize the print preview setting to show the Print Preview Dialog like Excel.

Example

This example shows how to set the **EnhancePreview** ('**EnhancePreview Property**' in the **on-line documentation**) property.

C#

```
//Code to show print preview dialog like Excel

fpSpread1.ActiveSheet.Cells[1, 1].Value = "1,1";
fpSpread1.ActiveSheet.Cells[10, 10].Value = "10,10";
```

```
fpSpread1.ActiveSheet.PrintInfo.EnhancePreview = true;
fpSpread1.PrintSheet(fpSpread1.ActiveSheet);
```

VB

```
'Code to show print preview dialog like Excel
fpSpread1.ActiveSheet.Cells(1, 1).Value = "1,1"
fpSpread1.ActiveSheet.Cells(10, 10).Value = "10,10"
fpSpread1.ActiveSheet.PrintInfo.EnhancePreview = True
fpSpread1.PrintSheet(fpSpread1.ActiveSheet)
```

Repeating Rows or Columns on Printed Pages

You can specify that rows appear at the top of every printed page or specify that columns appear on the left side of every printed page. Use the **RepeatRowStart** ('RepeatRowStart Property' in the on-line documentation), **RepeatRowEnd** ('RepeatRowEnd Property' in the on-line documentation), **RepeatColStart** ('RepeatColStart Property' in the on-line documentation), and **RepeatColEnd** ('RepeatColEnd Property' in the on-line documentation) properties of the **PrintInfo** ('PrintInfo Class' in the on-line documentation) object.

Using Code

1. Use the repeat properties of the **PrintInfo** object.
2. Use the **PrintSheet** ('PrintSheet Method' in the on-line documentation) method to print the sheet.

Example

This example sets the repeat options and uses a preview dialog.

C#

```
FarPoint.Win.Spread.PrintInfo printset = new FarPoint.Win.Spread.PrintInfo();
printset.RepeatColStart = 0;
printset.RepeatColEnd = 2;
printset.RepeatRowStart = 0;
printset.RepeatRowEnd = 2;
printset.Preview = true;
fpSpread1.Sheets[0].PrintInfo = printset;
fpSpread1.PrintSheet(0);
```

VB

```
Dim printset As New FarPoint.Win.Spread.PrintInfo
printset.RepeatColStart = 0
printset.RepeatColEnd = 2
printset.RepeatRowStart = 0
printset.RepeatRowEnd = 2
printset.Preview = True
fpSpread1.Sheets(0).PrintInfo = printset
fpSpread1.PrintSheet(0)
```

Adding a Page Break

You can add a force page break before a specified column or row. Page breaks are not displayed on the screen but force

page breaks when the sheet is printed. A column page break occurs to the left of the specified column. A row page break occurs above the specified row. To add or set the page breaks, use the **SetRowPageBreak ('SetRowPageBreak Method' in the on-line documentation)** and **SetColumnPageBreak ('SetColumnPageBreak Method' in the on-line documentation)** methods.

You can also retrieve the number of the next column or row in the sheet where a page break occurs. To find the page breaks that are already set, use the **GetRowPageBreaks ('GetRowPageBreaks Method' in the on-line documentation)** method to return the number of row page breaks and use the **GetColumnPageBreaks ('GetColumnPageBreaks Method' in the on-line documentation)** method to return the number of column page breaks.

You can calculate the number of printed pages for the sheet using the **GetPrintPageCount ('GetPrintPageCount Method' in the on-line documentation)** method.

Using Code

1. Use the **SetRowPageBreak ('SetRowPageBreak Method' in the on-line documentation)** method on the sheet to set the row page break.
2. Use the **GetRowPageBreaks ('GetRowPageBreaks Method' in the on-line documentation)** method to get the page breaks.

Example

This example sets a row page break and gets the row numbers for the row page breaks.

C#

```
// Add this code to the form load.
FarPoint.Win.Spread.PrintInfo pi = new FarPoint.Win.Spread.PrintInfo();
pi.UseMax = false;
fpSpread1.Sheets[0].PrintInfo = pi;
fpSpread1.Sheets[0].SetRowPageBreak(5, true);
// Add this code to a button click event.
int []i;
i = fpSpread1.GetRowPageBreaks(0);
foreach (object o in i)
{
    listBox1.Items.Add(o);
}
```

VB

```
' Add this code to the form load event.
Dim pi As New FarPoint.Win.Spread.PrintInfo()
pi.UseMax = False
fpSpread1.Sheets(0).PrintInfo = pi
fpSpread1.Sheets(0).SetRowPageBreak(5, True)
' Add this code to a button click event.
Dim i() As Integer
Dim o As Object
i = fpSpread1.GetRowPageBreaks(0)
For Each o In i
    ListBox1.Items.Add(o)
Next
```

Adding a Watermark to a Printed Page

You can print a background image or a watermark when the spreadsheet is printed.

Set the **PrintBackground** ('**PrintBackground Event**' in the on-line documentation) event to fire when printing, and specify the graphic with the **PrintBackground** event, and the opacity with the **PrintInfo.Opacity** ('**Opacity Property**' in the on-line documentation) property, so the printing of the spreadsheet has no watermark if opacity is highest (transparency is lowest) and shows the watermark through the spreadsheet if the opacity is low (transparency is high).

Using Code

Use the **PrintInfo** ('**PrintInfo Class**' in the on-line documentation) object to specify the opacity for printing a watermark.

Example

This example shows how to print a watermark.

C#

```
private void fpSpread1_PrintBackground(object sender,
FarPoint.Win.Spread.PrintBackgroundEventArgs e)
{
    FarPoint.Win.Picture pic = new
FarPoint.Win.Picture(System.Drawing.Image.FromFile("D:\\Files\\cover.jpg"),
FarPoint.Win.RenderStyle.Normal);
    pic.AlignHorz = FarPoint.Win.HorizontalAlignment.Left;
    pic.AlignVert = FarPoint.Win.VerticalAlignment.Top;
    pic.Paint(e.Graphics, e.SheetRectangle);
}

private void button1_Click(object sender, System.EventArgs e)
{
    FarPoint.Win.Spread.PrintInfo pi = new FarPoint.Win.Spread.PrintInfo();
    pi.Opacity = 100;
    fpSpread1.ActiveSheet.PrintInfo = pi;
    fpSpread1.PrintSheet(0);
}
```

VB

```
Private Sub fpSpread1_PrintBackground(ByVal sender As Object, ByVal e As
FarPoint.Win.Spread.PrintBackgroundEventArgs) Handles FpSpread1.PrintBackground
    Dim pic As New FarPoint.Win.Picture(Image.FromFile("D:\\Files\\cover.jpg"),
FarPoint.Win.RenderStyle.Normal)
    pic.AlignHorz = FarPoint.Win.HorizontalAlignment.Left
    pic.AlignVert = FarPoint.Win.VerticalAlignment.Top
    pic.Paint(e.Graphics, e.SheetRectangle)
End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
    Dim pi As New FarPoint.Win.Spread.PrintInfo()
    pi.Opacity = 100
    fpSpread1.ActiveSheet.PrintInfo = pi
    fpSpread1.PrintSheet(0)
End Sub
```

Optimizing the Printing

There are a few ways to let Spread handle the printing and optimize the printing for you. You can let Spread determine the optimum way using rules, a set of algorithms to calculate the best way to scale and fit your printing, or let Spread determine the biggest column or row and handle the printing based on that.

The tasks involved with optimizing the printing include:

- **Optimizing the Printing Using Rules**
- **Optimizing the Printing Using Size**

Optimizing the Printing Using Rules

Spread provides a way to automatically determine the best way to print your sheet. By using rules that you can choose, it can decide, for example, whether it is best to print your sheet on landscape- or portrait-oriented pages.

The rules, that you can turn on or off, that optimize the printing can be customized by setting the properties of these rule objects:

Rule Object	Description
LandscapeRule ('LandscapeRule Class' in the on-line documentation)	Determines whether to print the sheet in landscape or portrait orientation.
ScaleRule ('ScaleRule Class' in the on-line documentation)	Determines the best scale at which to print the sheet, starting with 100% (Start Factor = 1), and decreasing at set intervals to a minimum size (End Factor). Default settings are Start Factor = 1, End Factor = 0.6, and Interval = 0.1.
BestFitColumnRule ('BestFitColumnRule Class' in the on-line documentation)	Determines how best to fit the columns in the sheet on the page.

By default, optimizing the printing of the sheet uses the following logic:

- If the information can be printed without making any changes to the settings that you have defined in the **PrintInfo** (**'PrintInfo Class' in the on-line documentation**) object, the sheet prints in portrait mode.
- If the sheet is wider than a portrait page, the sheet prints in landscape mode.
- If the information does not fit in landscape mode, but does fit in landscape mode if the sheet is reduced up to 60% of its original size, the sheet is scaled to fit within the page.
- If the information cannot be scaled to fit, the sheet tries to reduce column widths to accommodate the widest string within each column.
- If all attempts to make the sheet print within a page fail, printing resumes normally in the current printer orientation with no reductions.

You can customize how this logic is applied through the rule objects. If you customize the rule object, the default rules are ignored and only the custom rules are used for printing. You can set up a collection of these rules with the **SmartPrintRulesCollection** (**'SmartPrintRulesCollection Class' in the on-line documentation**) object and set whether to use these rules with the **UseSmartPrint** (**'UseSmartPrint Property' in the on-line documentation**) property.

Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.

4. In the **Members** list, select the sheet for which to use SmartPrint.
5. In the properties list, double-click the **PrintInfo** property to display the settings for the **PrintInfo** class.
6. Set the **UseSmartPrint** property to true.
7. If you want to customize the SmartPrint rules, select the SmartPrintRules and click the button to display the **SmartPrintRule Collection Editor**. Customize the rules as you want, then click **OK** to close the **SmartPrintRule Collection Editor**.
8. Click **OK** to close the **SheetView Collection** editor.

Using a Shortcut

1. Create a **PrintInfo ('PrintInfo Class' in the on-line documentation)** object.
2. If you want to change how SmartPrint determines how best to print the sheet, create a new **SmartPrintRulesCollection ('SmartPrintRulesCollection Class' in the on-line documentation)** object.
3. Set the **PrintInfo ('PrintInfo Class' in the on-line documentation)** object **UseSmartPrint ('UseSmartPrint Property' in the on-line documentation)** property to true.
4. Set the Sheets shortcut object **PrintInfo ('PrintInfo Property' in the on-line documentation)** property to the **PrintInfo ('PrintInfo Class' in the on-line documentation)** object you just created.

Example

This example code prints using customized print rules, set up in the **SmartPrintRulesCollection ('SmartPrintRulesCollection Class' in the on-line documentation)** object. In this example, if the sheet does fit on a page by shrinking columns to the longest text string, it prints with the columns shrunk. If it does not fit with the columns shrunk, it keeps them shrunk and tries to print in landscape orientation. If it does not fit with the columns shrunk and in landscape orientation, it keeps these settings and tries to scale the sheet, starting at 100%, then decreasing by 20% intervals down to 40%.

C#

```
// Create the print rules.
FarPoint.Win.Spread.SmartPrintRulesCollection printrules = new
FarPoint.Win.Spread.SmartPrintRulesCollection();
printrules.Add(new
FarPoint.Win.Spread.BestFitColumnRule(FarPoint.Win.Spread.ResetOption.None));
printrules.Add(new
FarPoint.Win.Spread.LandscapeRule(FarPoint.Win.Spread.ResetOption.None));
printrules.Add(new FarPoint.Win.Spread.ScaleRule(FarPoint.Win.Spread.ResetOption.All,
1.0f, .4f, .2f));
// Create a PrintInfo object and set the properties.
FarPoint.Win.Spread.PrintInfo printset = new FarPoint.Win.Spread.PrintInfo();
printset.SmartPrintRules = printrules;
printset.UseSmartPrint = true;
// Set the PrintInfo property for the first sheet.
fpSpread1.Sheets[0].PrintInfo = printset;
// Print the sheet.
fpSpread1.PrintSheet(0);
```

VB

```
' Create the print rules.
Dim printrules As New FarPoint.Win.Spread.SmartPrintRulesCollection()
printrules.Add(New
FarPoint.Win.Spread.BestFitColumnRule(FarPoint.Win.Spread.ResetOption.None))
printrules.Add(New
FarPoint.Win.Spread.LandscapeRule(FarPoint.Win.Spread.ResetOption.None))
```

```

printrules.Add(New FarPoint.Win.Spread.ScaleRule(FarPoint.Win.Spread.ResetOption.All,
1.0F, 0.4F, 0.2F))
' Create a PrintInfo object and set the properties.
Dim printset As New FarPoint.Win.Spread.PrintInfo()
printset.SmartPrintRules = printrules
printset.UseSmartPrint = True
' Set the PrintInfo property for the first sheet.
fpSpread1.Sheets(0).PrintInfo = printset
' Print the sheet.
fpSpread1.PrintSheet(0)

```

Using Code

1. Create a **PrintInfo** ('PrintInfo Class' in the on-line documentation) object.
2. If you want to change how SmartPrint determines how best to print the sheet, create a new **SmartPrintRulesCollection** ('SmartPrintRulesCollection Class' in the on-line documentation) object.
3. Set the **PrintInfo** ('PrintInfo Class' in the on-line documentation) object **UseSmartPrint** ('UseSmartPrint Property' in the on-line documentation) property to true.
4. Set the **SheetView** ('SheetView Class' in the on-line documentation) object **PrintInfo** ('PrintInfo Property' in the on-line documentation) property to the **PrintInfo** ('PrintInfo Class' in the on-line documentation) object you just created.

Example

This example code prints using customized print rules, set up in the **SmartPrintRulesCollection** ('SmartPrintRulesCollection Class' in the on-line documentation) object. In this example, if the sheet does fit on a page by shrinking columns to the longest text string, it prints with the columns shrunk. If it does not fit with the columns shrunk, it keeps them shrunk and tries to print in landscape orientation. If it does not fit with the columns shrunk and in landscape orientation, it keeps these settings and tries to scale the sheet, starting at 100%, then decreasing by 20% intervals down to 40%.

C#

```

// Create the print rules.
FarPoint.Win.Spread.SmartPrintRulesCollection printrules = new
FarPoint.Win.Spread.SmartPrintRulesCollection();
printrules.Add(new
FarPoint.Win.Spread.BestFitColumnRule(FarPoint.Win.Spread.ResetOption.None));
printrules.Add(new
FarPoint.Win.Spread.LandscapeRule(FarPoint.Win.Spread.ResetOption.None));
printrules.Add(new FarPoint.Win.Spread.ScaleRule(FarPoint.Win.Spread.ResetOption.All,
1.0f, .4f, .2f));
// Create a PrintInfo object and set the properties.
FarPoint.Win.Spread.PrintInfo printset = new FarPoint.Win.Spread.PrintInfo();
printset.SmartPrintRules = printrules;
printset.UseSmartPrint = true;
// Create a SheetView object and assign it to the first sheet.
FarPoint.Win.Spread.SheetView SheetToPrint = new FarPoint.Win.Spread.SheetView();
SheetToPrint.PrintInfo = printset;
fpSpread1.Sheets[0] = SheetToPrint;
// Print the sheet.
fpSpread1.PrintSheet(0);

```

VB

```
' Create the print rules.
Dim printrules As New FarPoint.Win.Spread.SmartPrintRulesCollection()
printrules.Add(New
FarPoint.Win.Spread.BestFitColumnRule(FarPoint.Win.Spread.ResetOption.None))
printrules.Add(New
FarPoint.Win.Spread.LandscapeRule(FarPoint.Win.Spread.ResetOption.None))
printrules.Add(New FarPoint.Win.Spread.ScaleRule(FarPoint.Win.Spread.ResetOption.All,
1.0F, 0.4F, 0.2F))
' Create a PrintInfo object and set the properties.
Dim printset As New FarPoint.Win.Spread.PrintInfo()
printset.SmartPrintRules = printrules
printset.UseSmartPrint = True
' Create a SheetView object and assign it to the first sheet.
Dim SheetToPrint As New FarPoint.Win.Spread.SheetView()
SheetToPrint.PrintInfo = printset
fpSpread1.Sheets(0) = SheetToPrint
' Print the sheet.
fpSpread1.PrintSheet(0)
```

Using the Spread Designer

1. Select the sheet tab for the sheet for which you want to set print settings.
2. Select the **Page Layout** option.
3. Click the **SmartPrint** tab.
4. Select the **Use SmartPrint** check box if you want to print using the SmartPrint feature.
5. The default printing rules for SmartPrint are listed in the text box. If you want to do so, delete those rules by clicking the **Delete** button, then use the controls below the text box to add rules and rule options.
6. Click **OK** to close the **Sheet Print Options** dialog.
7. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Optimizing the Printing Using Size

Spread provides a way to automatically determine the best way to print your sheet. By using rules that you can choose, it can decide, for example, whether it is best to print your sheet on landscape- or portrait-oriented pages. See **Optimizing the Printing Using Rules** for more information.

You can set **BestFitCols** (**'BestFitCols Property' in the on-line documentation**) and **BestFitRows** (**'BestFitRows Property' in the on-line documentation**) properties.

Displaying Dialogs for Users

Spread provides a way for you to display a print dialog and a print preview dialog to allow your end users to set various options for printing.

- **Displaying a Print Dialog for the User**
- **Displaying an Abort Message for the User**
- **Providing a Preview of the Printing**

Displaying a Print Dialog for the User

Spread provides a way for you to display a print dialog to allow your end user to set various options on the printing. Set the **ShowPrintDialog** (**'ShowPrintDialog Property' in the on-line documentation**) property of the **PrintInfo**

(**PrintInfo Class** in the on-line documentation) object.

Displaying an Abort Message for the User

Spread provides a way for you to display an abort message to allow your end user a chance to cancel or continue with the printing.

For more information, refer to the following:

- **PrintAbortEventArgs** (**PrintAbortEventArgs Class** in the on-line documentation) class
- **PrintMessageBoxEventArgs** (**PrintMessageBoxEventArgs Class** in the on-line documentation) class
- **PrintInfo** (**PrintInfo Class** in the on-line documentation) class, **AbortMessage** property
- **FpSpread** (**FpSpread Class** in the on-line documentation) class, **PrintMessageBox** event, **PrintAbort** Event, and **PrintCancelled** Event

Using Code

1. Create and set the **AbortMessage** property for a **PrintInfo** (**PrintInfo Class** in the on-line documentation) object.
2. Set the **SheetView** object **PrintInfo** (**PrintInfo Property** in the on-line documentation) property to the **PrintInfo** (**PrintInfo Class** in the on-line documentation) object you just created.

Example

This example brings up the cancel dialog.

C#

```
FarPoint.Win.Spread.PrintInfo pi = new FarPoint.Win.Spread.PrintInfo();
pi.AbortMessage = "Do you want to cancel printing?";
fpSpread1.ActiveSheet.PrintInfo = pi;
fpSpread1.PrintSheet(0);
```

VB

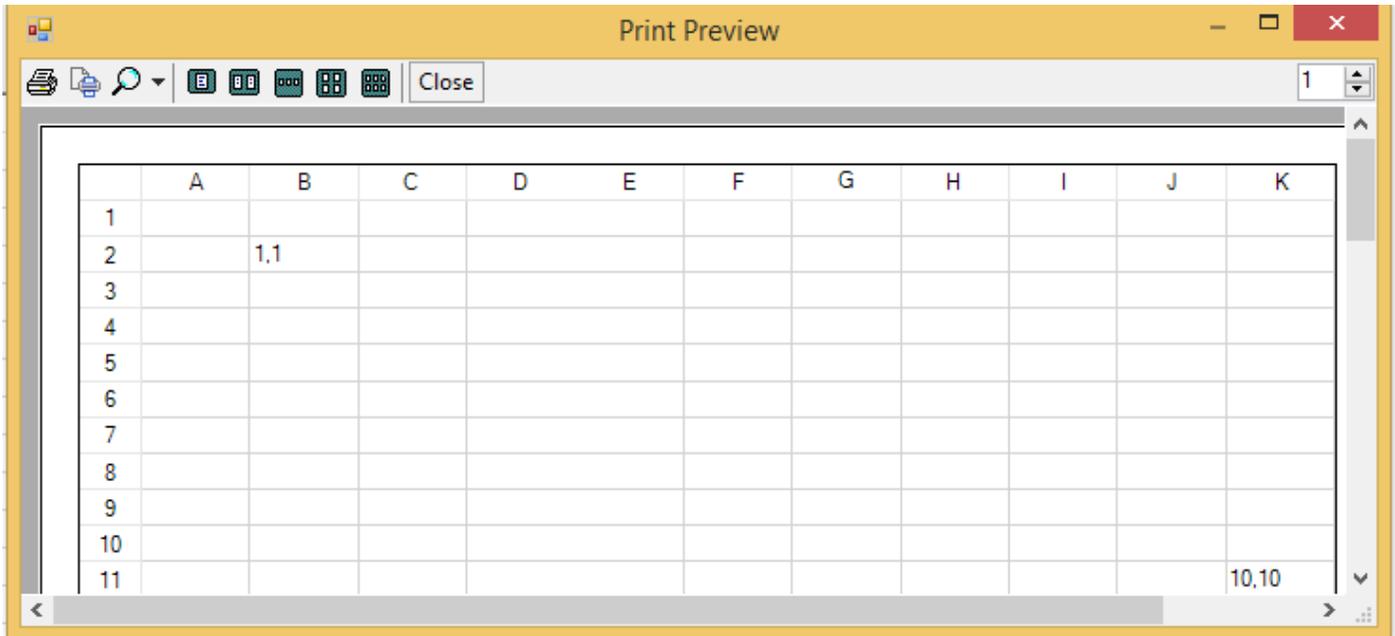
```
Dim pi As New FarPoint.Win.Spread.PrintInfo
pi.AbortMessage = "Do you want to cancel printing??"
fpSpread1.Sheets(0).PrintInfo = pi
' Print the sheet
fpSpread1.PrintSheet(0)
```

Providing a Preview of the Printing

You can preview what the printed pages will look like for a sheet and you can allow your end user to preview the printing.

You can use the **Preview** (**Preview Property** in the on-line documentation) property in the **PrintInfo** (**PrintInfo Class** in the on-line documentation) class to preview a sheet.

The following image shows the basic Print Preview Dialog that appears on your screen.



Use the **OwnerPrintDraw** ('**OwnerPrintDraw Method**' in the on-line documentation) method of the **FpSpread** ('**FpSpread Class**' in the on-line documentation) class to provide a print preview dialog with options for previewing the pages before printing.

Use the **ShowPageSetupButton** ('**ShowPageSetupButton Property**' in the on-line documentation) property of the **PrintInfo** ('**PrintInfo Class**' in the on-line documentation) class to show the PageSetUp button on the toolbar of the Print Preview Dialog. With this button, you can set the orientation (portrait and landscape) and margins etc. of the page you want to print.

Two additional printing features are: the **PrintPreviewShowing** ('**PrintPreviewShowing Event**' in the on-line documentation) event and the ability to set your own print preview dialog with the **SetPrintPreview** ('**SetPrintPreview Method**' in the on-line documentation) method (and the corresponding **GetPrintPreview** ('**GetPrintPreview Method**' in the on-line documentation) method), all members of the **FpSpread** ('**FpSpread Class**' in the on-line documentation) class. The **PrintPreviewShowing** ('**PrintPreviewShowing Event**' in the on-line documentation) event fires prior to displaying the dialog and supplies you with both the *PreviewDialog* and the *PreviewControl* in its event parameter list so you can make on-the-fly modifications to the **PrintPreviewDialog** and the **PrintPreviewControl** objects.

The print preview dialog allows you to zoom in and out to change the scale of what you see in the preview. You can set the preview to display one, two, four, or six pages of the spreadsheet per printed page. You can print from the print preview dialog or close the dialog when done.

You can also print and preview the printing within the Spread Designer. For more information on printing and previewing in Spread Designer, refer to **Printing a Sheet from Spread Designer** (on-line documentation) and **Previewing a Sheet in Spread Designer** (on-line documentation).

You can also customize the Print Preview settings to show the Print Preview Dialog like Excel. For more information, refer to **Customizing the Print Preview Dialog**.

Using Code

Set the **Preview** ('**Preview Property**' in the on-line documentation) property of the **PrintInfo** ('**PrintInfo Class**' in the on-line documentation) class to true in order to allow users to preview spreadsheets and see the modifications concurrently while the settings of the printer are being changed.

Example

This example shows how to apply printer setting to show the print preview dialog.

C#

```
//Code to show the print preview dialog

fpSpread1.ActiveSheet.Cells[1, 1].Value = "1,1";
fpSpread1.ActiveSheet.Cells[10, 10].Value = "10,10";
fpSpread1.ActiveSheet.PrintInfo.Preview = true;
fpSpread1.PrintSheet(fpSpread1.ActiveSheet);
```

VB

```
'Code to show print preview dialog
fpSpread1.ActiveSheet.Cells(1, 1).Value = "1,1"
fpSpread1.ActiveSheet.Cells(10, 10).Value = "10,10"
fpSpread1.ActiveSheet.PrintInfo.Preview = True
fpSpread1.PrintSheet(fpSpread1.ActiveSheet)
```

Chart Control

The following topics explain the basics of the Chart control and how to use the control:

- **Understanding Charts**
 - **Chart User Interface Elements**
 - **Chart Object Model**
 - **Chart Types and Views**
 - **Plot Types**
 - **Y Plot Types**
 - **Bar Charts**
 - **Area Charts**
 - **Box Whisker Charts**
 - **Funnel Charts**
 - **Histogram Charts**
 - **Line Charts**
 - **Market Data (High-Low) Charts**
 - **Pareto Charts**
 - **Point Charts**
 - **Stripe Charts**
 - **Waterfall Charts**
 - **XY Plot Types**
 - **XY Bubble Charts**
 - **XY Point Charts**
 - **XY Line Charts**
 - **XY Stripe Charts**
 - **XYZ Plot Types**
 - **XYZ Point Charts**
 - **XYZ Line Charts**
 - **XYZ Surface Charts**
 - **XYZ Stripe Charts**
 - **Pie Plot Types**
 - **Doughnut Charts**
 - **Pie Charts**
 - **Polar Plot Types**
 - **Polar Point Charts**
 - **Polar Line Charts**
 - **Polar Area Charts**
 - **Polar Stripe Charts**
 - **Pie Charts**
 - **Radar Plot Types**
 - **Radar Point Charts**
 - **Radar Line Charts**
 - **Radar Area Charts**
 - **Radar Stripe Charts**
 - **Data Plot Types**
 - **Plot area**

- Series
- Walls
- Axis
- Chart Line Style
- Elevation and Rotation
- Lighting, Shapes, and Borders
- Labels
- Multi-Level Category Labels (on-line documentation)
- Legends
- **Creating Charts**
 - **Using the Chart Control on sheet**
 - **Allowing the User to Change the Chart**
 - **Saving or Loading a Chart**
 - **Using the Chart Control**
 - **Save/Load Chart Control**
 - **Creating Plot Types**
 - **Creating a Y Plot**
 - **Creating an XY Plot**
 - **Creating an XYZ Plot**
 - **Creating a Pie Plot**
 - **Creating a Polar Plot**
 - **Creating a Radar Plot**
 - **Creating a Sunburst Chart**
 - **Creating a Treemap Chart**
 - **Combining different types of plots**
 - **Connecting to Data**
 - **Using a Bound Data Source**
 - **Using an UnBound Data Source**
 - **Binding with cell range**
 - **Advanced chart settings**
 - **Fill Effects**
 - **View Type**

Understanding Charts

The following topics explain the various chart elements and formatting options.

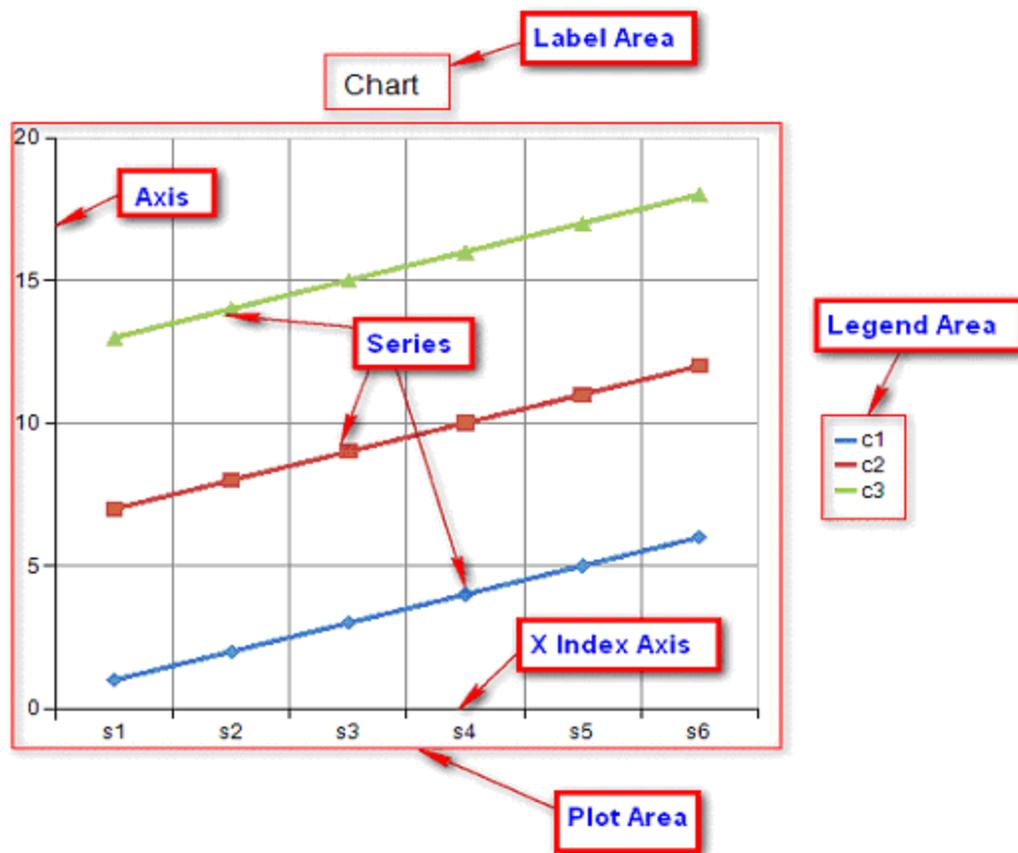
- **Chart User Interface Elements**
- **Chart Object Model**
- **Chart Types and Views**
- **Plot Types**
 - **Y Plot Types**
 - **Bar Charts**
 - **Area Charts**
 - **Box Whisker Charts**
 - **Funnel Charts**
 - **Histogram Charts**

- **Line Charts**
- **Market Data (High-Low) Charts**
- **Pareto Charts**
- **Point Charts**
- **Stripe Charts**
- **Waterfall Charts**
- **XY Plot Types**
 - **XY Bubble Charts**
 - **XY Point Charts**
 - **XY Line Charts**
 - **XY Stripe Charts**
- **XYZ Plot Types**
 - **XYZ Point Charts**
 - **XYZ Line Charts**
 - **XYZ Surface Charts**
 - **XYZ Stripe Charts**
- **Pie Plot Types**
 - **Doughnut Charts**
 - **Pie Charts**
- **Polar Plot Types**
 - **Polar Point Charts**
 - **Polar Line Charts**
 - **Polar Area Charts**
 - **Polar Stripe Charts**
 - **Pie Charts**
- **Radar Plot Types**
 - **Radar Point Charts**
 - **Radar Line Charts**
 - **Radar Area Charts**
 - **Radar Stripe Charts**
- **Data Plot Types**
- **Plot area**
 - **Series**
 - **Walls**
 - **Axis**
 - **Chart Line Style**
 - **Elevation and Rotation**
 - **Lighting, Shapes, and Borders**
- **Labels**
- **Multi-Level Category Labels (on-line documentation)**
- **Legends**

Chart User Interface Elements

Here is a brief summary of the elements of the canvas (chart view) of the chart.

This diagram shows the parts of the canvas:



There are three main elements in the chart:

- **LabelArea ('LabelArea Class' in the on-line documentation)** - Labels contain the plot title and the axis labels.
- **LegendArea ('LegendArea Class' in the on-line documentation)** - Legends contain identifiers for each of the series of data.
- **PlotArea ('PlotArea Class' in the on-line documentation)** - The plot consists of data displayed in one of several plot types. For more information about plot types, refer to Plot Types. On the plot are several graphical elements such as grid lines, tick marks, stripes, and walls.

 For information on various plot types, see **Plot Types**.

Elements are positioned using a relative location, where (0,0) is the upper left corner of the chart and (1,1) is the lower right corner of the chart and a relative alignment, where (0,0) is the upper left corner of the element and (1,1) is the lower right corner of the element.

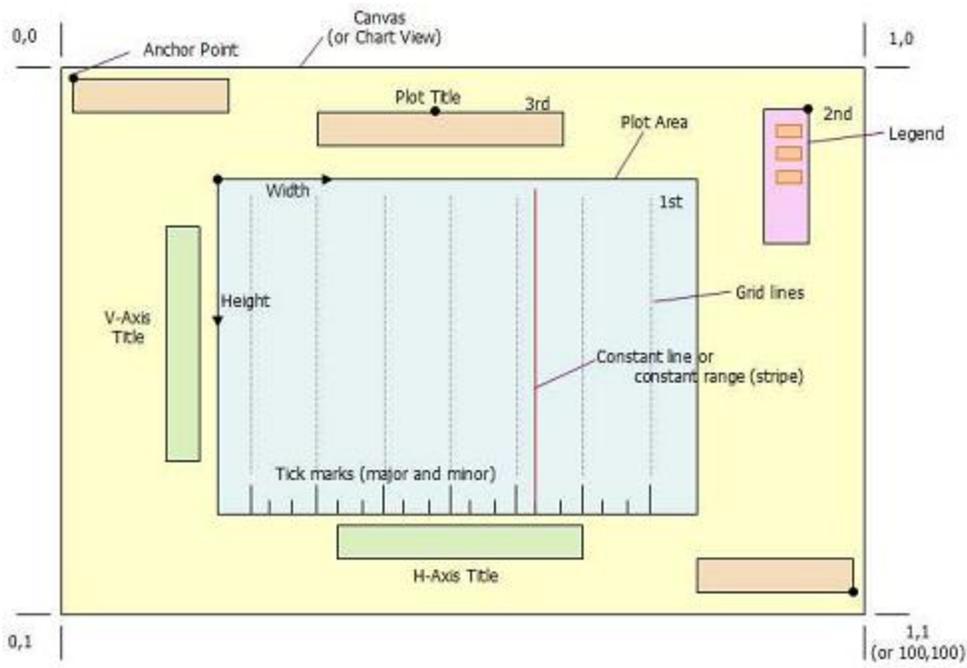
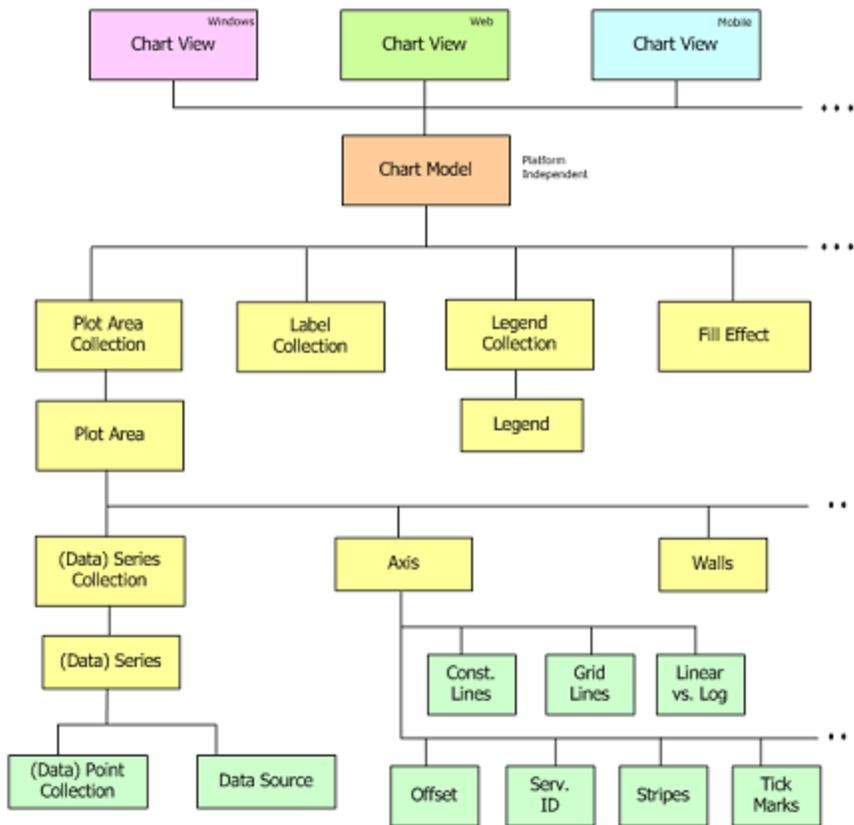


Chart Object Model

Here is a brief summary of the object model of the chart.
 This diagram shows the object model:



Basically there is a chart view that can be platform specific. This chart view is dependent on the chart model, which is platform independent.

The model in turn is made up of four major objects:

- **PlotAreaCollection** ('PlotAreaCollection Class' in the on-line documentation)
- **LabelAreaCollection** ('LabelAreaCollection Class' in the on-line documentation)
- **LegendAreaCollection** ('LegendAreaCollection Class' in the on-line documentation)
- **Fill** ('Fill Class' in the on-line documentation)

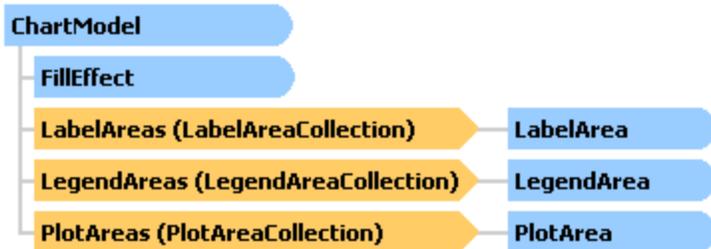


Chart Types and Views

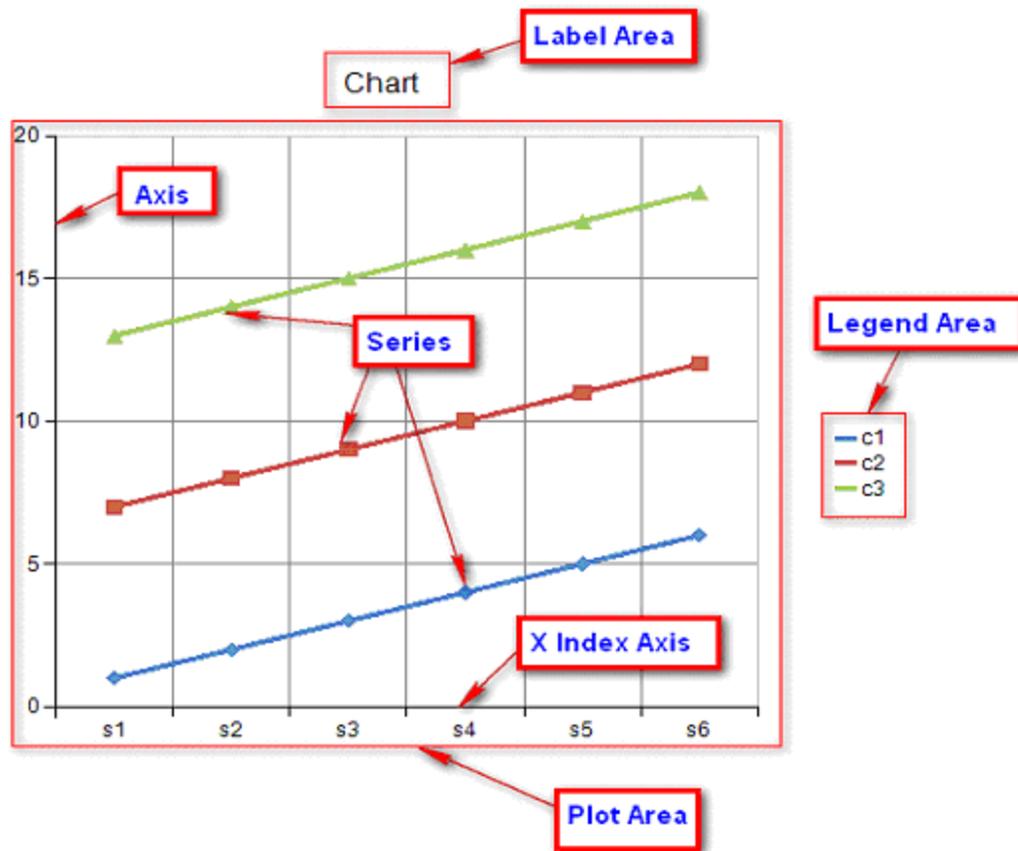
The chart control has several chart types and each type has additional views.

The following is a list of the chart types:

Chart **Type of view**

Column	The column type has the following types of views - Clustered Column, Stacked Column, 100% Stacked Column, High Low Column, 3D Clustered Column, 3D Stacked Column, 100% 3D Stacked Column, 3D Column, 3D High Low Column, Clustered Cylinder, Stacked Cylinder, 100% Stacked Cylinder, 3D Cylinder, High Low Column Cylinder, Clustered Full Cone, Stacked Full Cone, 100% Stacked Full Cone, 3D Full Cone, High Low Column Full Cone, Clustered Full Pyramid, Stacked Full Pyramid, 100% Stacked Full Pyramid, 3D Pyramid, and High Low Column Pyramid.
Line	The line type has the following types of views - Line, Stacked Line, 100% Stacked Line, Line with Markers, Stacked Line with Markers, 100% Stacked Line with Markers, and 3D Line.
Pie	The pie type has the following types of views - 2D Pie, 3D Pie, 2D Exploded Pie, and 3D Exploded Pie.
Bar	The bar type has the following types of views - Clustered Bar, Stacked Bar, 100% Stacked Bar, High Low Bar, 3D Clustered Bar , 3D Stacked Bar, 100% 3D Stacked Bar, 3D High Low Bar, Clustered Horizontal Cylinder, Stacked Horizontal Cylinder, 100% Stacked Horizontal Cylinder, High Low Bar Cylinder, Clustered Horizontal Full Cone, Stacked Horizontal Full Cone, 100% Stacked Horizontal, High Low Bar Full Cone, Clustered Horizontal Full Pyramid, Stacked Horizontal Full Pyramid, 100% Stacked Horizontal, and High Low Bar Pyramid.
Area	The area type has the following types of views - Area, Stacked Area, 100% Stacked Area, High Low Area, 3D Area, 3D Stacked Area, 100% 3D Stacked Area, and 3D High Low Area.
XY	The XY type has the following types of views - XY Point, XY Line, and XY Line with Marker.
Bubble	The bubble type has the following types of views - 2D Bubble and 3D Bubble.
Stock	The stock type has the following types of views - High Low Close, Open High Low Close, and Candle Stick.
XYZ	The XYZ type has the following types of views - XYZ Line, XYZ Line with Marker, XYZ Point, and Surface.
Doughnut	The doughnut type has the following types of views - Doughnut and Exploded Doughnut.
Radar	The radar type has the following types of views - Radar Line, Radar Line with Marker, Radar Point, and Radar Area.
Polar	The polar type has the following types of views - Polar Line, Polar Line with Marker, Polar Point, and Polar Area.
Treemap	Treemap
Sunburst	Sunburst
Histogram	The Histogram type has the following types of views - Histogram, Pareto chart.
Box Whisker	Box Whisker
Waterfall	Waterfall
Funnel	Funnel

There are several visual elements to a chart such as the plot, legend, and label areas, the axis, and the series. The label area contains additional information about the chart. The legend can be used to help end users identify different chart elements such as the series. The axis displays the scale for a single dimension of a plot area. Each series is a collection of data points. The plot area is the area in which data points are drawn.



Plot Types

A plot area is the area in which data points (bars, points, lines, etc) are drawn. A plot area will contain a collection of series.

Each series is a collection of data points. A plot area may also contain an axis(s) and wall(s). The axis(s) and wall(s) enhance the visual appearance of the plot area and are usually painted just outside the plot area.

A plot area can be assigned an anchor view location and a view size. The anchor view location is specified in normalized units ((0,0) = left upper corner of chart view, (1,1) = right lower corner of chart view). The view size of the plot area is specified in normalized units ((0,0) = zero size, (1,1) = full size of chart view). The left top corner of the plot area is aligned with the anchor location.

A plot area can be assigned a model size using width, height, and depth properties. The properties use model units.

There are several plot types: Y, XY, XYZ, Pie, Polar, Radar, Data, Sunburst, and Treemap.

- **Y Plot Types**
- **XY Plot Types**
- **XYZ Plot Types**
- **Pie Plot Types**
- **Polar Plot Types**
- **Radar Plot Types**

Y Plot Types

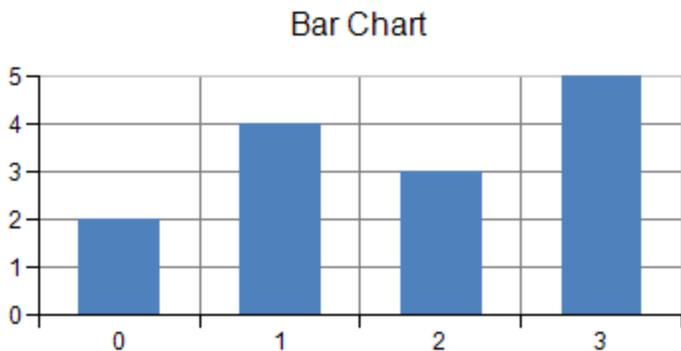
The Y plot area contains series that have values in one dimension.

When visualized in 2D, a Y plot area takes the form of a rectangle with the x-axis representing categories and the y-axis representing values.

When visualized in 3D, a Y plot area takes the form of a cube with the x-axis representing categories, the y-axis representing values, and the z-axis (depth) representing series.

A Y plot area can be oriented vertically or horizontally. When oriented vertically, the x-axis is horizontal and the y-axis is vertical. When oriented horizontally, the x-axis is vertical and the y-axis is horizontal.

The following image depicts a bar chart.



You can have any of these types of Y plots.

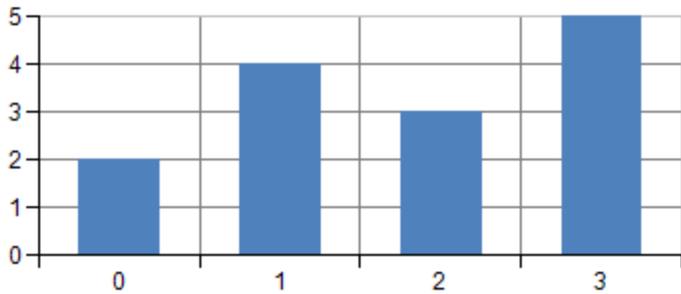
- **Bar Charts**
- **Area Charts**
- **Box Whisker Charts**
- **Funnel Charts**
- **Histogram Charts**
- **Line Charts**
- **Market Data (High-Low) Charts**
- **Pareto Charts**
- **Point Charts**
- **Stripe Charts**
- **Waterfall Charts**

For more information, see the **YPlotArea** ('**YPlotArea Class**' in the on-line documentation) class.

Bar Charts

The Bar chart is a basic one-dimensional Cartesian plot such as the one shown in this figure.

Bar Chart



Each data point contains a single value; how the bar for that point is displayed can be customized. Bar borders and bar fill effects can be assigned for the series or for a point in the series with null (Nothing in VB) indicating unassigned. Bar width and bar depth are measured relative to the floor grid cell (with a range of 0 to 1). Bar origin can be automatically generated or manually assigned.

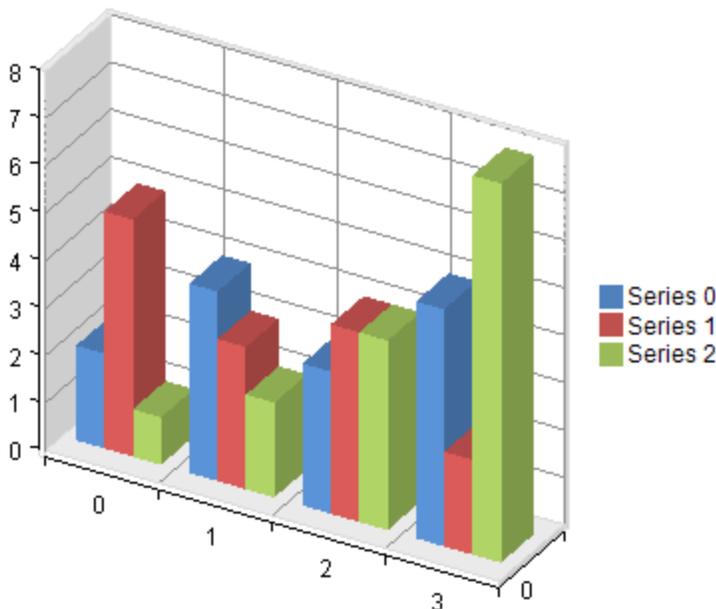
For more information on the bar series object in the API, refer to the **BarSeries ('BarSeries Class' in the on-line documentation)** class.

You can have any of these types of bar charts, which represent different ways of displaying the series data.

- (standard) Bar
- Multiple Bar
- Cluster Bar
- Stacked Bar
- Stacked 100% (normalized) Bar

A Cluster Bar chart aggregates bars by series and places the bars in the set adjacent to each other.

Cluster Bar Chart



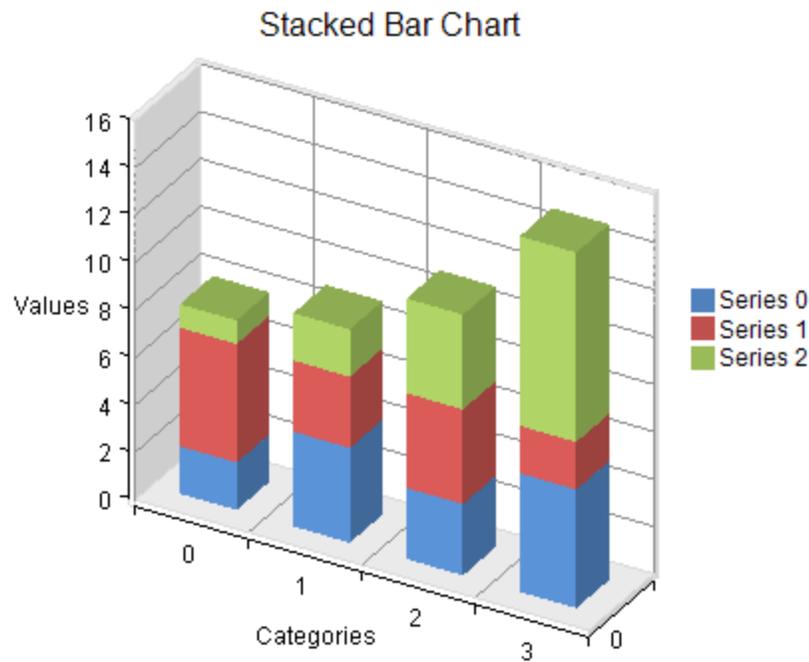
A cluster bar series is a composite series that groups together two or more bar series. A cluster bar series can be assigned a border, fill effect, width, depth, and an origin for the bars as well as a width for the group. Assigning null for a border or fill effect indicates that the property is unset. Group width is measured relative to the floor grid cell (0 = no width, 1 = width of floor grid cell). Bar width is measured relative to the width reserved for the group divided by the number of series in the group (0 = no width, 1 = width reserved for the group divided by the number of series in the group). Bar

depth is measured relative to the floor grid cell (0 = no depth, 1 = depth of floor grid cell). The origin can be marked for auto generation by the chart view, in which case the assigned origin is ignored.

Each bar series in the cluster bar series can be assigned a border and a fill effect for the bars.

Each point in a bar series in the cluster bar series has a single data value. Each point is visualized as a bar. All the bars for a given category are placed side by side. Each point in a bar series in a cluster bar series can have a border and a fill effect for the bar.

The stacked bar shows the bars vertically stacked:



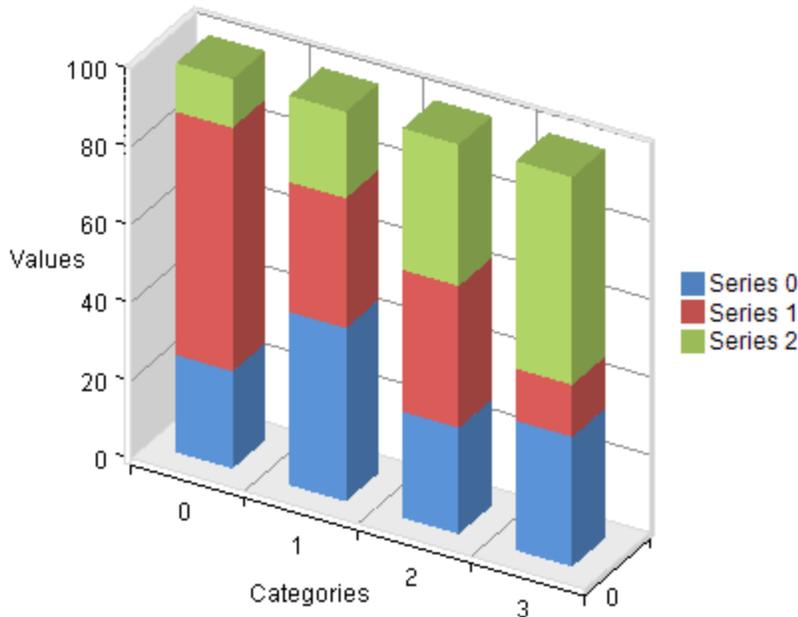
A stacked bar series is a composite series that groups together two or more bar series. A stacked bar series can be assigned a border, fill effect, width, and a depth for the bars. You can also specify whether the group should be displayed 100% stacked. Assigning null for a border or fill effect indicates that the property is unset. Width and depth are measured relative to the floor grid cell (0 = no width or depth, 1 = width or depth of floor grid cell).

Each bar series in the stacked bar series can be assigned a border and a fill effect for the bars.

Each point in a bar series in a stacked bar series has a single data value. Each point is visualized as a bar. All the bars in a given category are stacked vertically. Each point in a bar series in a stacked bar series can have a border and a fill effect for the bar.

The stacked 100% bar chart shows the bars vertically stacked and spread all the way to the top (100%) but spread proportionately; otherwise, it is similar to a stacked bar chart. The height of each bar in a group indicates its percentage of the total.

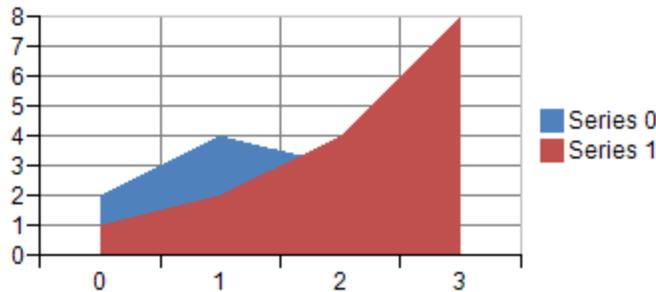
100% Stacked Bar Chart



Area Charts

The Area chart can be a basic one-dimensional Cartesian plot such as the one shown in this figure.

Area Chart



Each point in an area series contains a single data value. The data value is visualized as a point on an area.

An area series can have a border, fill effect, depth, or an origin for the area. You can also specify whether the area is jagged or smooth, and whether drop lines are displayed. Assigning null for the border or fill effect indicates that the property is unset. Depth is measured relative to the floor grid cell (0 = no depth, 1 = depth of floor grid cell). The origin can be marked for auto generation by the chart view, in which case the assigned origin is ignored.

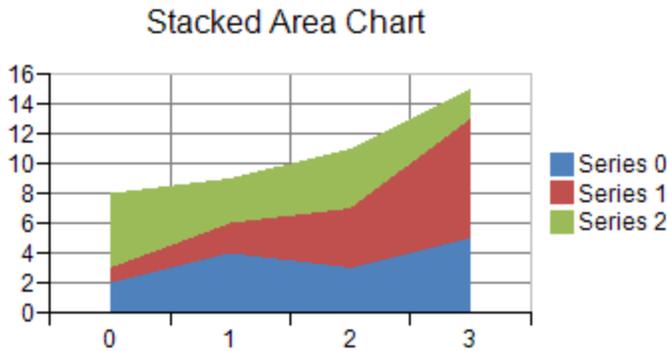
Each point in an area series can be assigned a border and a fill effect for the area.

For more information on the area series object in the API, refer to the **AreaSeries ('AreaSeries Class' in the online documentation)** class.

You can have any of these types of area charts, which represent different ways of displaying the series data.

- (standard) Area
- Stacked Area
- Stacked 100% (normalized) Area

The stacked area chart shows the points vertically stacked.

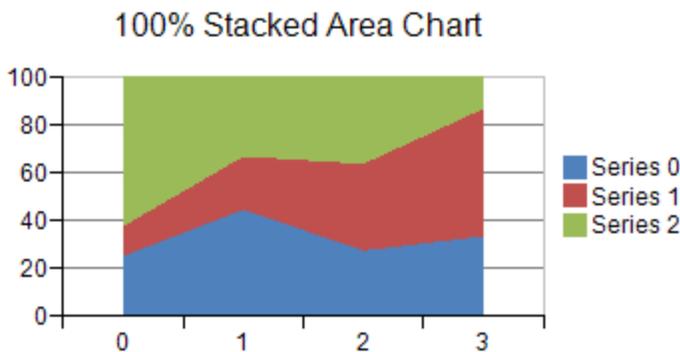


A stacked area series is a composite series that groups together two or more area series.

A stacked area series can be assigned a border, fill effect, or depth for the areas. You can also specify whether the area is jagged or smooth and whether drop lines are displayed. Assigning null for a border or fill effect indicates that the property is unset. Each area series in a stacked area series can be assigned a border and a fill effect for the area.

Each point in an area series in a stacked area series has a single data value. The data value is visualized as a point on an area. Each point in an area series in a stacked area series can be assigned a border and a fill effect for the area.

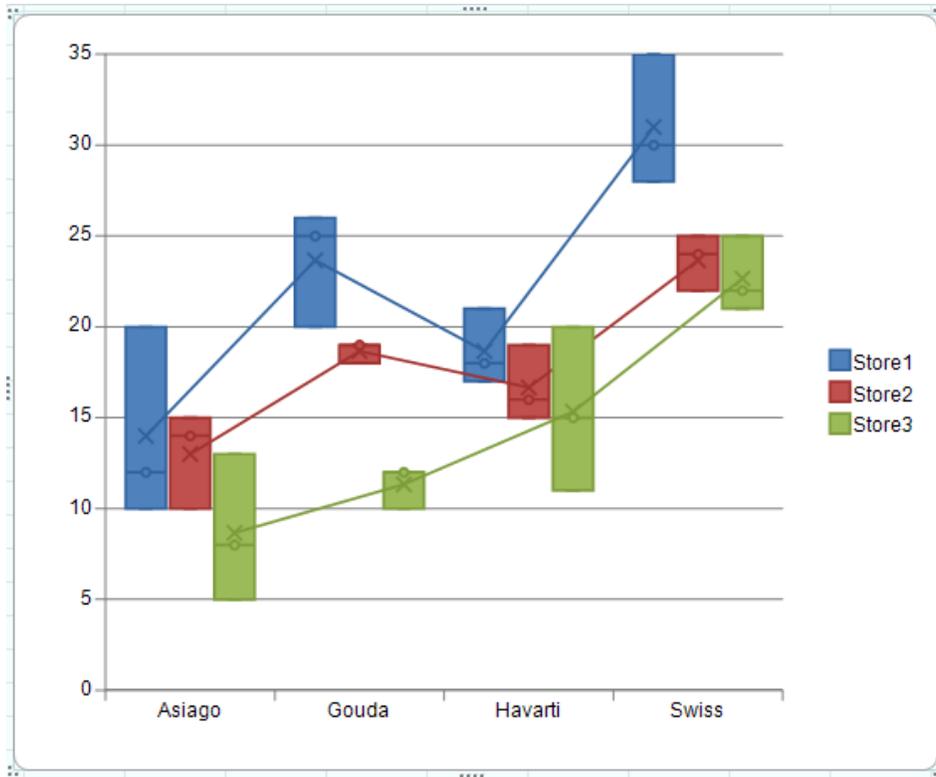
The stacked 100% area chart shows the points vertically stacked and spread all the way to the top (100%) but spread proportionately; otherwise, it is similar to a stacked area chart.



Box Whisker Charts

A box whisker chart displays the distribution of data into quartiles. The boxes may also have lines extending vertically (whiskers). The lines indicate variability outside the upper and lower quartiles. A point outside the whisker is an outlier point. Box whisker charts are commonly used in statistical analysis.

You can use the **BoxWhiskerSeries** ('[BoxWhiskerSeries Class](#)' in the on-line documentation), the **ClusteredBoxWhiskerSeries** ('[ClusteredBoxWhiskerSeries Class](#)' in the on-line documentation), and the **YPlotArea** ('[YPlotArea Class](#)' in the on-line documentation) classes to create a box whisker chart with multiple series as shown in the following image.



For information about creating charts in the Spread Designer or the Chart Designer, refer to **Using the Chart Control on sheet** or **Adding a Chart Control ('Adding Charts' in the on-line documentation)**.

Using Code

1. Add data for the chart.
2. Add the chart.
3. Set any additional properties such as whether to show the mean lines.

Example

This example creates a box whisker chart.

C#

```

fpSpread1.ActiveSheet.Cells[0, 1].Text = "Cheese";
fpSpread1.ActiveSheet.Cells[0, 2].Text = "Store1";
fpSpread1.ActiveSheet.Cells[0, 3].Text = "Store2";
fpSpread1.ActiveSheet.Cells[0, 4].Text = "Store3";
fpSpread1.ActiveSheet.Cells[1, 1].Text = "Asiago";
fpSpread1.ActiveSheet.Cells[2, 1].Text = "Gouda";
fpSpread1.ActiveSheet.Cells[3, 1].Text = "Havarti";
fpSpread1.ActiveSheet.Cells[4, 1].Text = "Swiss";
fpSpread1.ActiveSheet.Cells[5, 1].Text = "Asiago";
fpSpread1.ActiveSheet.Cells[6, 1].Text = "Gouda";
fpSpread1.ActiveSheet.Cells[7, 1].Text = "Havarti";
fpSpread1.ActiveSheet.Cells[8, 1].Text = "Swiss";
fpSpread1.ActiveSheet.Cells[9, 1].Text = "Asiago";
fpSpread1.ActiveSheet.Cells[10, 1].Text = "Gouda";
fpSpread1.ActiveSheet.Cells[11, 1].Text = "Havarti";
fpSpread1.ActiveSheet.Cells[12, 1].Text = "Swiss";
fpSpread1.ActiveSheet.Cells[1, 2].Value = 20;
fpSpread1.ActiveSheet.Cells[2, 2].Value = 25;
fpSpread1.ActiveSheet.Cells[3, 2].Value = 21;
fpSpread1.ActiveSheet.Cells[4, 2].Value = 30;
fpSpread1.ActiveSheet.Cells[5, 2].Value = 10;
    
```

```

fpSpread1.ActiveSheet.Cells[6, 2].Value = 26;
fpSpread1.ActiveSheet.Cells[7, 2].Value = 18;
fpSpread1.ActiveSheet.Cells[8, 2].Value = 28;
fpSpread1.ActiveSheet.Cells[9, 2].Value = 12;
fpSpread1.ActiveSheet.Cells[10, 2].Value = 20;
fpSpread1.ActiveSheet.Cells[11, 2].Value = 17;
fpSpread1.ActiveSheet.Cells[12, 2].Value = 35;
fpSpread1.ActiveSheet.Cells[1, 3].Value = 15;
fpSpread1.ActiveSheet.Cells[2, 3].Value = 18;
fpSpread1.ActiveSheet.Cells[3, 3].Value = 19;
fpSpread1.ActiveSheet.Cells[4, 3].Value = 22;
fpSpread1.ActiveSheet.Cells[5, 3].Value = 10;
fpSpread1.ActiveSheet.Cells[6, 3].Value = 19;
fpSpread1.ActiveSheet.Cells[7, 3].Value = 15;
fpSpread1.ActiveSheet.Cells[8, 3].Value = 25;
fpSpread1.ActiveSheet.Cells[9, 3].Value = 14;
fpSpread1.ActiveSheet.Cells[10, 3].Value = 19;
fpSpread1.ActiveSheet.Cells[11, 3].Value = 16;
fpSpread1.ActiveSheet.Cells[12, 3].Value = 24;
fpSpread1.ActiveSheet.Cells[1, 4].Value = 5;
fpSpread1.ActiveSheet.Cells[2, 4].Value = 12;
fpSpread1.ActiveSheet.Cells[3, 4].Value = 20;
fpSpread1.ActiveSheet.Cells[4, 4].Value = 25;
fpSpread1.ActiveSheet.Cells[5, 4].Value = 8;
fpSpread1.ActiveSheet.Cells[6, 4].Value = 10;
fpSpread1.ActiveSheet.Cells[7, 4].Value = 11;
fpSpread1.ActiveSheet.Cells[8, 4].Value = 22;
fpSpread1.ActiveSheet.Cells[9, 4].Value = 13;
fpSpread1.ActiveSheet.Cells[10, 4].Value = 12;
fpSpread1.ActiveSheet.Cells[11, 4].Value = 15;
fpSpread1.ActiveSheet.Cells[12, 4].Value = 21;
fpSpread1.ActiveSheet.AddChart(new FarPoint.Win.Spread.Model.CellRange(0, 1, 13, 4),
typeof(FarPoint.Win.Chart.ClusteredBoxWhiskerSeries), 550, 450, 300, 0);
FarPoint.Win.Chart.ClusteredBoxWhiskerSeries cboxseries =
(FarPoint.Win.Chart.ClusteredBoxWhiskerSeries)fpSpread1.Sheets[0].Charts[0].Model.PlotAreas[0].Series[0];
foreach (FarPoint.Win.Chart.BoxWhiskerSeries boxseries in cboxseries.Series)
{
    boxseries.ShowInnerPoints = true;
    boxseries.ShowMeanLine = true;
    boxseries.ShowMeanMarkers = true;
    boxseries.ShowOutlierPoints = true;
}

```

VB

```

fpSpread1.ActiveSheet.Cells(0, 1).Text = "Cheese"
fpSpread1.ActiveSheet.Cells(0, 2).Text = "Store1"
fpSpread1.ActiveSheet.Cells(0, 3).Text = "Store2"
fpSpread1.ActiveSheet.Cells(0, 4).Text = "Store3"
fpSpread1.ActiveSheet.Cells(1, 1).Text = "Asiago"
fpSpread1.ActiveSheet.Cells(2, 1).Text = "Gouda"
fpSpread1.ActiveSheet.Cells(3, 1).Text = "Havarti"
fpSpread1.ActiveSheet.Cells(4, 1).Text = "Swiss"
fpSpread1.ActiveSheet.Cells(5, 1).Text = "Asiago"
fpSpread1.ActiveSheet.Cells(6, 1).Text = "Gouda"
fpSpread1.ActiveSheet.Cells(7, 1).Text = "Havarti"
fpSpread1.ActiveSheet.Cells(8, 1).Text = "Swiss"
fpSpread1.ActiveSheet.Cells(9, 1).Text = "Asiago"
fpSpread1.ActiveSheet.Cells(10, 1).Text = "Gouda"
fpSpread1.ActiveSheet.Cells(11, 1).Text = "Havarti"
fpSpread1.ActiveSheet.Cells(12, 1).Text = "Swiss"
fpSpread1.ActiveSheet.Cells(1, 2).Value = 20
fpSpread1.ActiveSheet.Cells(2, 2).Value = 25
fpSpread1.ActiveSheet.Cells(3, 2).Value = 21
fpSpread1.ActiveSheet.Cells(4, 2).Value = 30
fpSpread1.ActiveSheet.Cells(5, 2).Value = 10
fpSpread1.ActiveSheet.Cells(6, 2).Value = 26

```

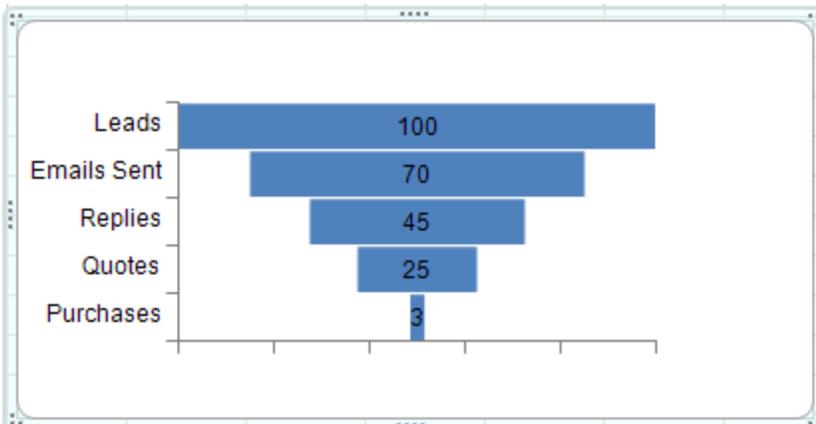
```

fpSpread1.ActiveSheet.Cells(7, 2).Value = 18
fpSpread1.ActiveSheet.Cells(8, 2).Value = 28
fpSpread1.ActiveSheet.Cells(9, 2).Value = 12
fpSpread1.ActiveSheet.Cells(10, 2).Value = 20
fpSpread1.ActiveSheet.Cells(11, 2).Value = 17
fpSpread1.ActiveSheet.Cells(12, 2).Value = 35
fpSpread1.ActiveSheet.Cells(1, 3).Value = 15
fpSpread1.ActiveSheet.Cells(2, 3).Value = 18
fpSpread1.ActiveSheet.Cells(3, 3).Value = 19
fpSpread1.ActiveSheet.Cells(4, 3).Value = 22
fpSpread1.ActiveSheet.Cells(5, 3).Value = 10
fpSpread1.ActiveSheet.Cells(6, 3).Value = 19
fpSpread1.ActiveSheet.Cells(7, 3).Value = 15
fpSpread1.ActiveSheet.Cells(8, 3).Value = 25
fpSpread1.ActiveSheet.Cells(9, 3).Value = 14
fpSpread1.ActiveSheet.Cells(10, 3).Value = 19
fpSpread1.ActiveSheet.Cells(11, 3).Value = 16
fpSpread1.ActiveSheet.Cells(12, 3).Value = 24
fpSpread1.ActiveSheet.Cells(1, 4).Value = 5
fpSpread1.ActiveSheet.Cells(2, 4).Value = 12
fpSpread1.ActiveSheet.Cells(3, 4).Value = 20
fpSpread1.ActiveSheet.Cells(4, 4).Value = 25
fpSpread1.ActiveSheet.Cells(5, 4).Value = 8
fpSpread1.ActiveSheet.Cells(6, 4).Value = 10
fpSpread1.ActiveSheet.Cells(7, 4).Value = 11
fpSpread1.ActiveSheet.Cells(8, 4).Value = 22
fpSpread1.ActiveSheet.Cells(9, 4).Value = 13
fpSpread1.ActiveSheet.Cells(10, 4).Value = 12
fpSpread1.ActiveSheet.Cells(11, 4).Value = 15
fpSpread1.ActiveSheet.Cells(12, 4).Value = 21
fpSpread1.ActiveSheet.AddChart(New FarPoint.Win.Spread.Model.CellRange(0, 1, 13, 4),
GetType(FarPoint.Win.Chart.ClustereBoxWhiskerSeries), 550, 450, 300, 0)
Dim cboxseries As FarPoint.Win.Chart.ClustereBoxWhiskerSeries =
DirectCast(fpSpread1.Sheets(0).Charts(0).Model.PlotAreas(0).Series(0),
FarPoint.Win.Chart.ClustereBoxWhiskerSeries)
For Each boxseries As FarPoint.Win.Chart.BoxWhiskerSeries In cboxseries.Series
    boxseries.ShowInnerPoints = True
    boxseries.ShowMeanLine = True
    boxseries.ShowMeanMarkers = True
    boxseries.ShowOutlierPoints = True
Next

```

Funnel Charts

A funnel chart shows values across multiple stages. The values typically decrease at each stage.



Funnel charts are useful for sales and other types of data. For example, determining how many web site visits lead to

product purchases.

You can use the **FunnelSeries ('FunnelSeries Class' in the on-line documentation)** class and the **YPlotArea ('YPlotArea Class' in the on-line documentation)** class to add a funnel chart.

For information about creating charts in the Spread Designer or the Chart Designer, refer to **Using the Chart Control on sheet** or **Adding a Chart Control ('Adding Charts' in the on-line documentation)**.

Using Code

1. Add data for the chart with the **FunnelSeries ('FunnelSeries Class' in the on-line documentation)** class.
2. Specify the plot area using the **YPlotArea ('YPlotArea Class' in the on-line documentation)** class.
3. Set the position and size of the plot area.
4. Add a series to the plot area.
5. Create a chart model and add the plot area to this model.
6. Create a chart and set a chart model for this chart.
7. Add the chart to SPREAD.

Example

This example creates a funnel chart.

C#

```
FarPoint.Win.Chart.FunnelSeries funnel= new FarPoint.Win.Chart.FunnelSeries();
funnel.Values.Add(100);
funnel.Values.Add(70);
funnel.Values.Add(45);
funnel.Values.Add(25);
funnel.Values.Add(3);
funnel.CategoryNames.Add("Leads");
funnel.CategoryNames.Add("Emails Sent");
funnel.CategoryNames.Add("Replies");
funnel.CategoryNames.Add("Quotes");
funnel.CategoryNames.Add("Purchases");

FarPoint.Win.Chart.YPlotArea plotArea = new FarPoint.Win.Chart.YPlotArea();
plotArea.Location = new PointF(0.2f, 0.2f);
plotArea.Size = new SizeF(0.6f, 0.6f);
plotArea.Series.Add(funnel);
FarPoint.Win.Chart.ChartModel model = new FarPoint.Win.Chart.ChartModel();
model.PlotAreas.Add(plotArea);
FarPoint.Win.Spread.Chart.SpreadChart chart = new
FarPoint.Win.Spread.Chart.SpreadChart();
chart.Model = model;
chart.Left = 0;
chart.Top = 150;
chart.Size = new Size(400, 200);
fpSpread1.ActiveSheet.Charts.Add(chart);
```

VB

```
Dim funnel As New FarPoint.Win.Chart.FunnelSeries()
funnel.Values.Add(100)
funnel.Values.Add(70)
funnel.Values.Add(45)
funnel.Values.Add(25)
```

```

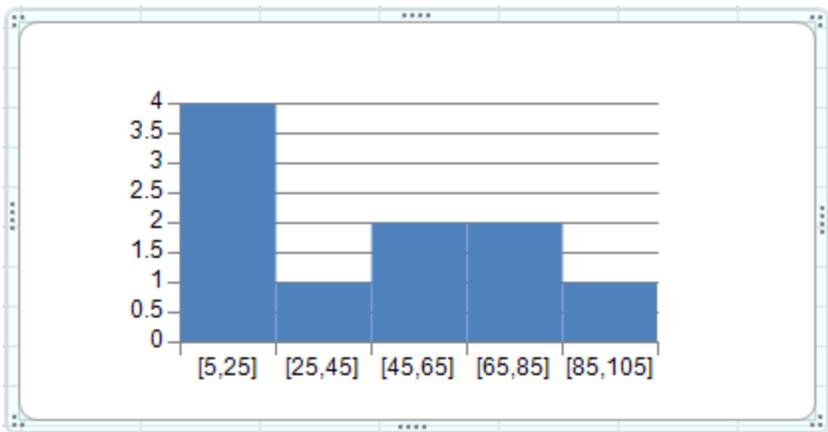
funnel.Values.Add(3)
funnel.CategoryNames.Add("Leads")
funnel.CategoryNames.Add("Emails Sent")
funnel.CategoryNames.Add("Replies")
funnel.CategoryNames.Add("Quotes")
funnel.CategoryNames.Add("Purchases")

Dim plotArea As New FarPoint.Win.Chart.YPlotArea()
plotArea.Location = New PointF(0.2F, 0.2F)
plotArea.Size = New SizeF(0.6F, 0.6F)
plotArea.Series.Add(funnel)
Dim model As New FarPoint.Win.Chart.ChartModel()
model.PlotAreas.Add(plotArea)
Dim chart As New FarPoint.Win.Spread.Chart.SpreadChart()
chart.Model = model
chart.Left = 0
chart.Top = 150
chart.Size = New Size(400, 200)
fpSpread1.ActiveSheet.Charts.Add(chart)

```

Histogram Charts

A histogram chart shows the frequency of data. The chart uses two types of data (data to be analyzed and the bin numbers that represent the frequency intervals of the data).



You can specify the following options for the chart:

Properties

AutoOverflow
(**'AutoOverflow Property'** in the on-line documentation)

AutoUnderFlow
(**'AutoUnderFlow Property'** in the on-line documentation)

BinCount (**'BinCount Property'** in the on-line documentation)

Description

Specifies that the bin width is calculated using Scott's normal reference rule. Scott's normal reference rule tries to minimize the bias in variance of the histogram compared with the data set, while assuming normally distributed data.

Specifies the number of bins for the histogram (including overflow and underflow bins).

OverflowValue (OverflowValue Property in the on-line documentation) IsOverflowBin (IsOverflowBin Property in the on-line documentation)	Creates a bin for all values above the specified value.
IsUnderflowBin (IsUnderflowBin Property' in the on-line documentation) UnderFlowValue (UnderFlowValue Property' in the on-line documentation)	Creates a bin for all values below or equal to the specified value.
BinType (BinType Property' in the on-line documentation)	Specifies whether to use categories or numbers for the bins.

You can add data to the control or use the **HistogramSeries** (**HistogramSeries Class' in the on-line documentation**) class to create data for the chart. Use the **BinSize** (**BinSize Property' in the on-line documentation**) property to specify the bin size.

For information about creating charts in the Spread Designer or the Chart Designer, refer to **Using the Chart Control on sheet** or **Adding a Chart Control** (**Adding Charts' in the on-line documentation**).

Using Code

1. Add data for the chart with the **HistogramSeries** (**HistogramSeries Class' in the on-line documentation**) class.
2. Specify the plot area using the **YPlotArea** (**YPlotArea Class' in the on-line documentation**) class.
3. Set the position and size of the plot area.
4. Add a series to the plot area.
5. Create a chart model and add the plot area to this model.
6. Create a chart and set a chart model for this chart.
7. Add the chart to SPREAD.

Example

This example creates a histogram chart with a bin size of 20.

C#

```
FarPoint.Win.Chart.HistogramSeries hs = new FarPoint.Win.Chart.HistogramSeries();
hs.SeriesName = "Histogram Sample";
hs.Values.Add(23);
hs.Values.Add(5);
hs.Values.Add(79);
hs.Values.Add(11);
hs.Values.Add(23);
hs.Values.Add(55);
hs.Values.Add(88);
hs.Values.Add(67);
hs.Values.Add(42);
```

```
hs.Values.Add(56);
hs.BinOption.BinSize = 20;

FarPoint.Win.Chart.YPlotArea plotArea = new FarPoint.Win.Chart.YPlotArea();
plotArea.Location = new PointF(0.2f, 0.2f);
plotArea.Size = new SizeF(0.6f, 0.6f);
plotArea.Series.Add(hs);

FarPoint.Win.Chart.ChartModel model = new FarPoint.Win.Chart.ChartModel();
model.PlotAreas.Add(plotArea);

FarPoint.Win.Spread.Chart.SpreadChart chart = new
FarPoint.Win.Spread.Chart.SpreadChart();
chart.Model = model;
chart.Left = 0;
chart.Top = 150;
chart.Size = new Size(400, 200);
fpSpread1.ActiveSheet.Charts.Add(chart);
```

VB

```
Dim hs As New FarPoint.Win.Chart.HistogramSeries()
hs.SeriesName = "Histogram Sample"
hs.Values.Add(23)
hs.Values.Add(5)
hs.Values.Add(79)
hs.Values.Add(11)
hs.Values.Add(23)
hs.Values.Add(55)
hs.Values.Add(88)
hs.Values.Add(67)
hs.Values.Add(42)
hs.Values.Add(56)
hs.BinOption.BinSize = 20

Dim plotArea As New FarPoint.Win.Chart.YPlotArea()
plotArea.Location = New PointF(0.2F, 0.2F)
plotArea.Size = New SizeF(0.6F, 0.6F)
plotArea.Series.Add(hs)

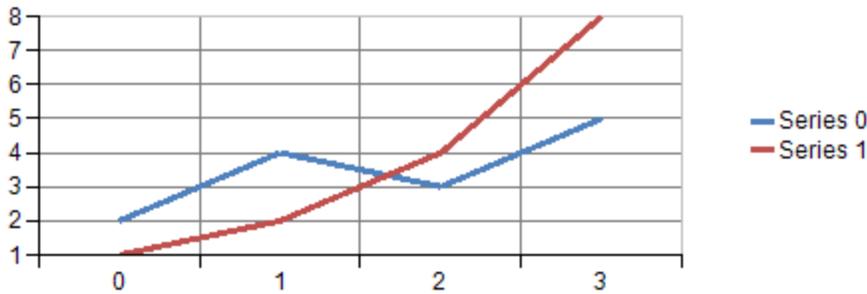
Dim model As New FarPoint.Win.Chart.ChartModel()
model.PlotAreas.Add(plotArea)

Dim chart As New FarPoint.Win.Spread.Chart.SpreadChart()
chart.Model = model
chart.Left = 0
chart.Top = 150
chart.Size = New Size(400, 200)
fpSpread1.ActiveSheet.Charts.Add(chart)
```

Line Charts

The line chart can be a basic one-dimensional Cartesian plot such as the one shown in this figure.

Line Chart



Each point in a line series contains a single data value. The data values are visualized as points on a line.

A line series can be assigned a border, fill effect, or depth for the line. The line can also be jagged or smooth or display drop lines. Assigning null for the border or fill effect indicates that the property is unset. Depth is measured relative to the floor grid cell (0 = no depth, 1 = depth of floor grid cell).

Each point in a line series can have a border or a fill effect for the line.

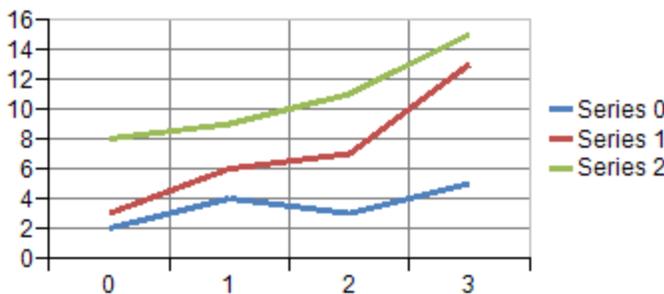
For more information on the line series object in the API, refer to the **LineSeries ('LineSeries Class' in the on-line documentation)** class.

You can have any of these types of line charts, which represent different ways of displaying the series data.

- (standard) Line
- Stacked Line
- Stacked 100% (normalized) Line

The stacked line chart shows the points vertically stacked.

Stacked Line Chart

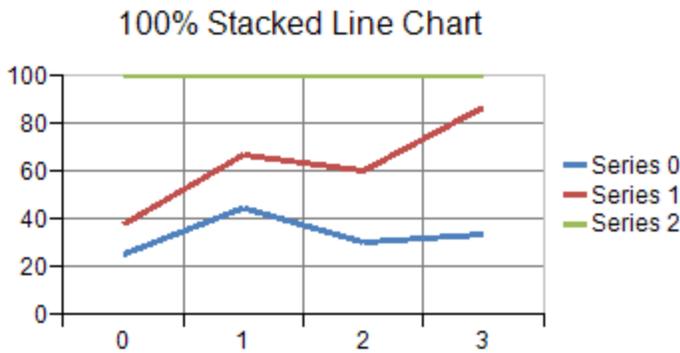


A stacked line series is a composite series that groups together two or more line series. A stacked line series can have a border, fill effect, or depth for the lines. You can also specify whether the line is jagged or smooth and whether drop lines are displayed. Assigning null for a border or fill effect indicates that the property is unset. Depth is measured relative to the floor grid cell (0 = no depth, 1 = depth of floor grid cell).

Each line series in a stacked line series can be assigned a border and a fill effect for the line.

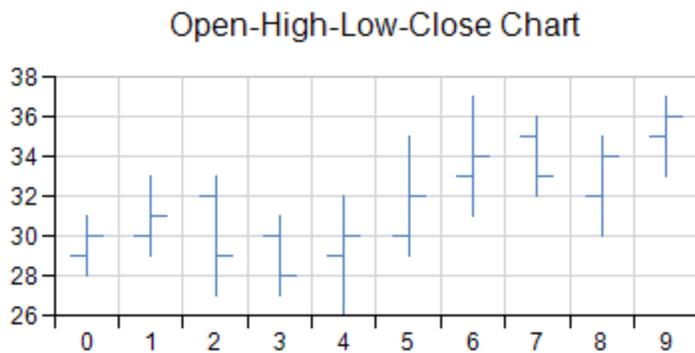
Each point in a line series in a stacked line series has a single data value. The data value is visualized as a point on a line. Each point in a line series in a stacked line series can be assigned a border and a fill effect for the line.

The stacked 100% line chart shows the points vertically stacked and spread all the way to the top (100%) but spread proportionately; otherwise, it is similar to a stacked line chart.



Market Data (High-Low) Charts

The market data (high-low) charts are another version of the one-dimensional Cartesian plot (Y plot) specifically designed for displaying market data often with high and low values as well as market open and market close values. For example:

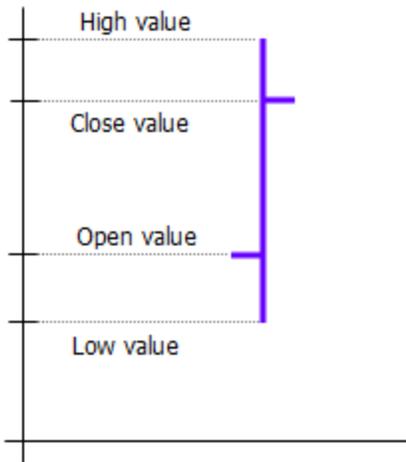


The normal high-low series are displayed as vertical lines with the high value being the vertically highest point on the line and the low value being the lowest point on the line. The opening market value is the small horizontal tick on the left of the line and the closing value is the small horizontal tick on the right side.

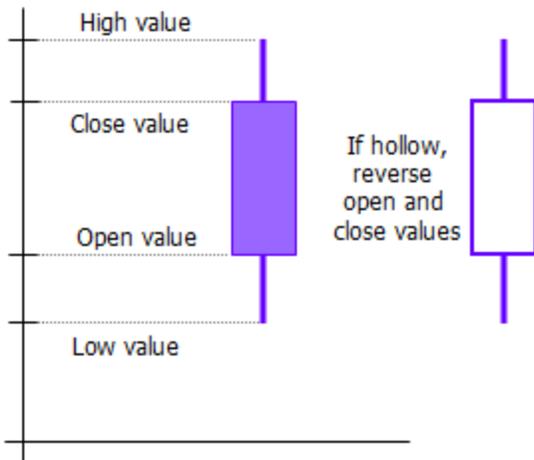
A high-low bar series can have a border, fill effect, width, and depth for the bars. Each point can be assigned a border and a fill effect for the bar.

An open-high-low-close series can be assigned a line for up or down points, and a width. The width is measured relative to the floor grid cell (0 = no depth, 1 = width to edge of grid cell).

Each point in an open-high-low-close series contains four data values: open, high, low, close. Each point is visualized as a line extending from the low value to the high value with smaller markers at the open and close values.



The candlestick high-low series are displayed as bars with the high value being the vertically highest point on the bar and the low value being the lowest point on the bar. If it is solid, then the opening value was lower; if it is hollow, the opening value was higher.



A candlestick series can be assigned a border and a fill effect for up or down points. You can also set the width and depth for the bars.

Each point in a candlestick series contains four values: open, high, low, and close. Each point is visualized as a line extending from the low value to the high value and a bar extending from the open value to the close value.

Each point can be assigned a border for up and down points. You can also set a fill effect for up and down points.

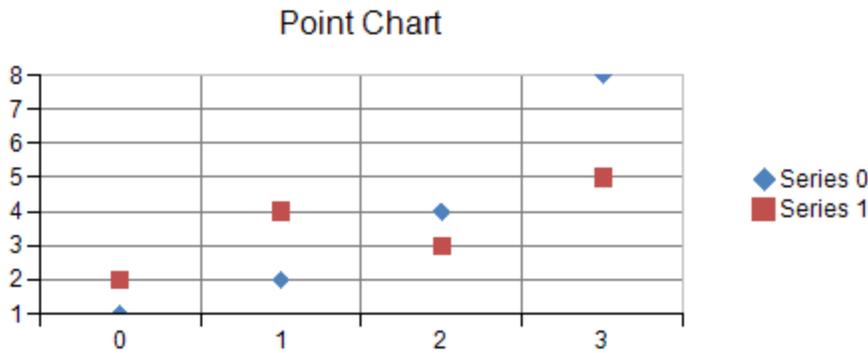
For more information on the objects in the API, refer to these:

- **HighLowAreaSeries ('HighLowAreaSeries Class' in the on-line documentation)**
- **HighLowBarSeries ('HighLowBarSeries Class' in the on-line documentation)**
- **HighLowCloseSeries ('HighLowCloseSeries Class' in the on-line documentation)**



Point Charts

The point chart can be a basic one-dimensional Cartesian plot such as the one shown in this figure:



A point marker is used to visualize each data value. Each point in a point series contains a single data value.

A point series can be assigned a border, fill effect, shape, size, and depth for the point markers. Assigning a null for the border or fill effect indicates that the property is unset. Size is measured in model units. Depth is measured relative to the floor of the grid cell (0 = no width, 1 = width of floor grid cell).

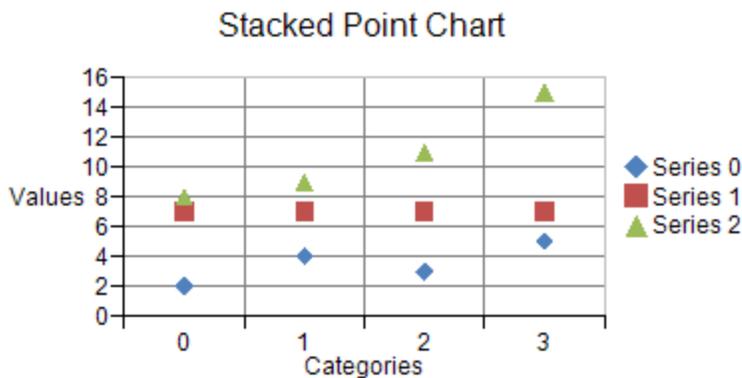
Each point in a point series can also be assigned a border and a fill effect for the point marker.

For more information on the point series object in the API, refer to the **PointSeries ('PointSeries Class' in the online documentation)** class.

You can have any of these types of point charts, which represent different ways of displaying the series data.

- (standard) Point
- Stacked Point
- Stacked 100% (normalized) Point

The stacked point chart shows the points vertically stacked.



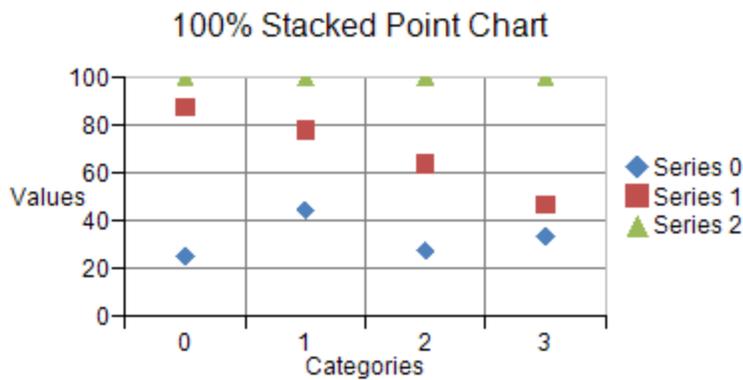
A stacked point series is a composite series that groups together two or more point series.

A stacked point series can have a border, fill effect, size, or depth for the point markers. You can also specify whether the group should be displayed as 100% stacked. Assigning null for a border or fill effect indicates that the property is unset. Size is measured in model units. Depth is measured relative to the depth of floor grid cell (0 = no depth, 1 = depth of floor grid cell).

Each point series in a stacked point series can be assigned a border and fill effect for point makers. Each point in a point series has a single data value. The data value is visualized as a point marker.

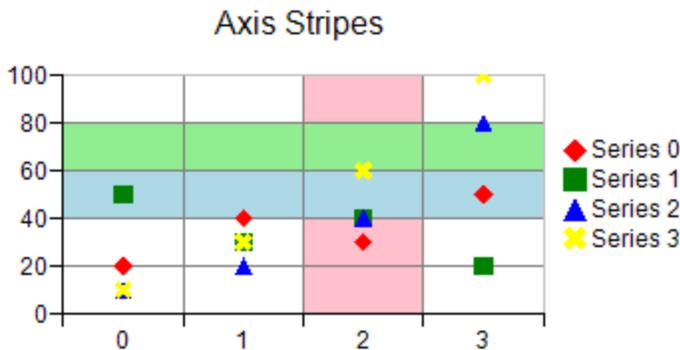
Each point in a point series in a stacked point series can be assigned a border and a fill effect for the point marker.

The stacked 100% point chart shows the points vertically stacked and spread all the way to the top (100%) but spread proportionately; otherwise, it is similar to a stacked point chart.



Stripe Charts

Stripes can be used in a basic one-dimensional Cartesian plot such as the one shown in this figure:



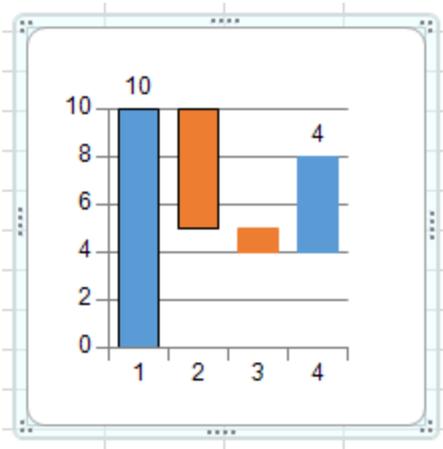
You can specify a start and end value for the stripe on the chart axis. You can also specify a fill type and color for the stripe. A stripe can be added to a chart with the axis properties in the appropriate plot area class.

For more information on the stripe object in the API, refer to the **Stripe ('Stripe Class' in the on-line documentation)** class.

For more information on the value axis series object in the API, refer to the **ValueAxis ('ValueAxis Class' in the on-line documentation)** class.

Waterfall Charts

A waterfall chart can be used to show how an initial value is affected by negative or positive values. Columns are color coded so you can quickly see differences between positive and negative values. Some values start on the horizontal axis, while intermediate values are shown as floating columns.



You can use the **WaterfallSeries** ('[WaterfallSeries Class](#)' in the on-line documentation) class and the **YPlotArea** ('[YPlotArea Class](#)' in the on-line documentation) class to create a waterfall chart.

For information about creating charts in the Spread Designer or the Chart Designer, refer to **Using the Chart Control on sheet** or **Adding Charts** (on-line documentation).

Using Code

1. Create the series with the **WaterfallSeries** ('[WaterfallSeries Class](#)' in the on-line documentation) class.
2. Create the plot type.
3. Set properties for the class.
4. Add the chart.

Example

This example creates a waterfall chart.

C#

```
FarPoint.Win.Chart.WaterfallSeries wseries = new FarPoint.Win.Chart.WaterfallSeries();
wseries.SeriesName = "Series0";
wseries.Values.Add(10);
wseries.Values.Add(-5);
wseries.Values.Add(-1);
wseries.Values.Add(4);
wseries.Border = new FarPoint.Win.Chart.SolidLine(Color.Black);

FarPoint.Win.Chart.YPlotArea plotArea = new FarPoint.Win.Chart.YPlotArea();
plotArea.Location = new System.Drawing.PointF(0.2f, 0.2f);
plotArea.Size = new System.Drawing.SizeF(0.6f, 0.6f);
plotArea.Series.Add(wseries);
FarPoint.Win.Chart.ChartModel model = new FarPoint.Win.Chart.ChartModel();
model.PlotAreas.Add(plotArea);

FarPoint.Win.Spread.Chart.SpreadChart chart = new
FarPoint.Win.Spread.Chart.SpreadChart();
chart.Model = model;
chart.Size = new Size(200, 200);
chart.Location = new Point(100, 100);
fpSpread1.Sheets[0].Charts.Add(chart);
```

VB

```

Dim wseries = New FarPoint.Win.Chart.WaterfallSeries()
wseries.SeriesName = "Series0"
wseries.Values.Add(10)
wseries.Values.Add(-5)
wseries.Values.Add(-1)
wseries.Values.Add(4)
wseries.Border = New FarPoint.Win.Chart.SolidLine(Color.Black)
Dim plotArea As New FarPoint.Win.Chart.YPlotArea()
plotArea.Location = New System.Drawing.PointF(0.2F, 0.2F)
plotArea.Size = New System.Drawing.SizeF(0.6F, 0.6F)
plotArea.Series.Add(wseries)
Dim model As New FarPoint.Win.Chart.ChartModel()
model.PlotAreas.Add(plotArea)

Dim chart As New FarPoint.Win.Spread.Chart.SpreadChart()
chart.Model = model
chart.Size = New Size(200, 200)
chart.Location = New Point(100, 100)
fpSpread1.Sheets(0).Charts.Add(chart)

```

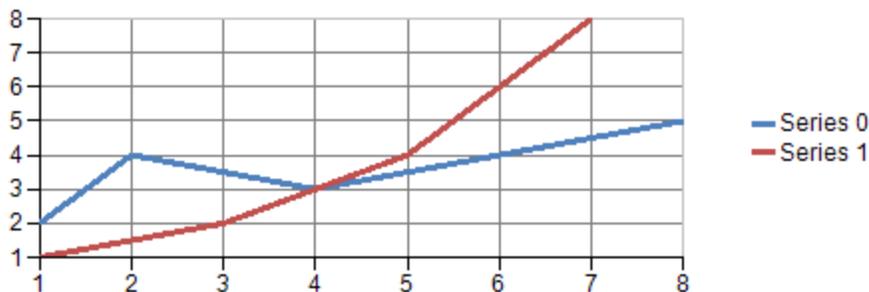
XY Plot Types

The XY plot area contains series that have values in two dimensions. When visualized in 2D, an XY plot area takes the form of a rectangle with a horizontal x-axis representing values and a vertical y-axis representing values. When visualized in 3D, the XY plot area takes the form of a cube with a horizontal x-axis representing values, a vertical y-axis representing values, and a depth z-axis representing series.

When a plot area has multiple x-axes or multiple y-axes, a series can be assigned to a specific axis using the axis's ID.

XY Line Chart, example of two-dimensional plot:

XY Line Chart

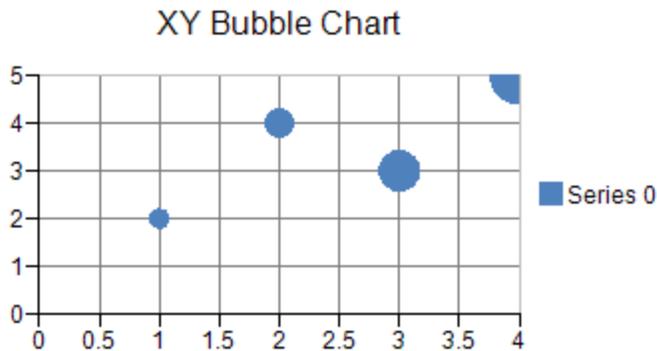


You can have any of these types of XY plots.

- **XY Bubble Charts**
- **XY Point Charts**
- **XY Line Charts**
- **XY Stripe Charts**

XY Bubble Charts

The bubble chart can be an XY plot such as the one shown in this figure.



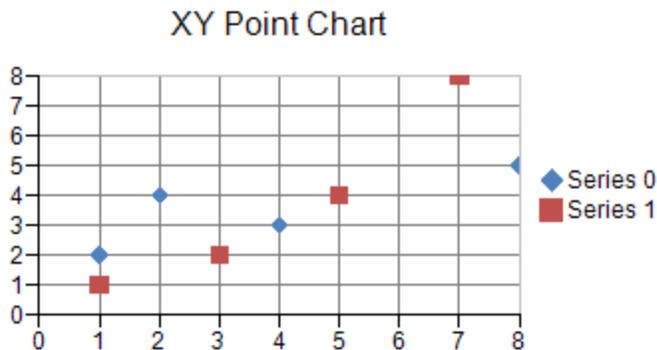
Each point contains two values: value and size.

Bubble borders and fill effects can be assigned for the series or for a point in the series. Null (Nothing in VB) indicates that the property is not set. Bubble size is measured relative to the plot area width (with a range of 0 to 1). Bubble depth is measured relative to the floor grid (with a range of 0 to 1).

For more information on the bubble series object in the API, refer to the **XYBubbleSeries ('XYBubbleSeries Class' in the on-line documentation)** class.

XY Point Charts

The point chart can be an XY plot such as the one shown in this figure.



A point marker is used to visualize each data value. Each point in a point series contains a single data value.

An XY point series can have a border, fill effect, shape, size, and depth for the point markers. Assigning null for a border or fill effect indicates that the property is unset. Size is measured in model units. Depth is measured relative to the floor grid cell (0 = no width, 1 = width of floor grid cell).

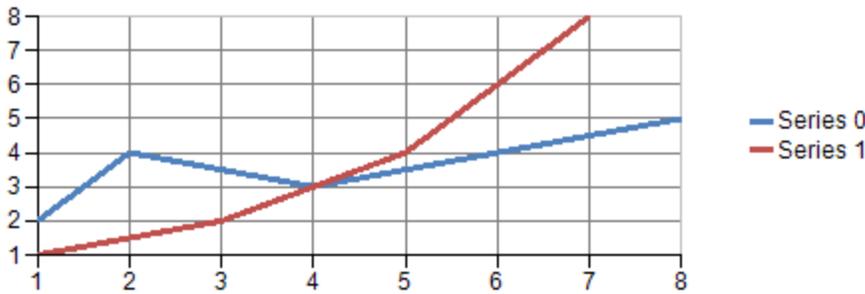
Each point in an XY point series contains two data values: x and y. Each point is visualized as a point marker. Each point can be assigned a border and a fill effect for the point marker.

For more information on the point series object in the API, refer to the **XYPointSeries ('XYPointSeries Class' in the on-line documentation)** class.

XY Line Charts

The line chart can be an XY plot such as the one shown in this figure.

XY Line Chart



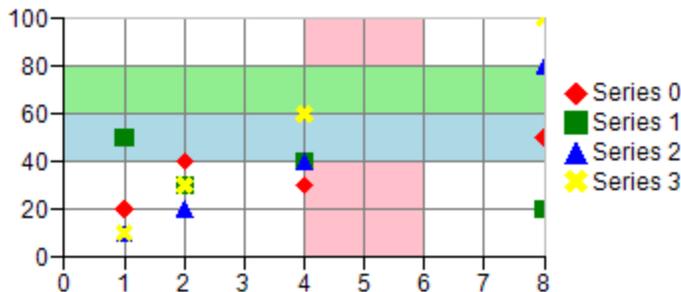
An XZ line series can have a border, fill effect, and depth for the line. You can also specify whether the line is jagged or smooth and whether drop lines are displayed. Assigning null for a border or fill effect indicates that the property is unset. Depth is measured relative to the floor grid cell (0 = no depth, 1 = depth of floor grid cell).

For more information on the line series object in the API, refer to the **XYLineSeries ('XYLineSeries Class' in the on-line documentation)** class.

XY Stripe Charts

The stripe chart can be an XY plot such as the one shown in this figure.

Axis Stripes



You can specify a start and end value for the stripe on the chart axis. You can also specify a fill type and color for the stripe. A stripe can be added to a chart with the axis properties in the appropriate plot area class.

For more information on the stripe object in the API, refer to the **Stripe ('Stripe Class' in the on-line documentation)** class.

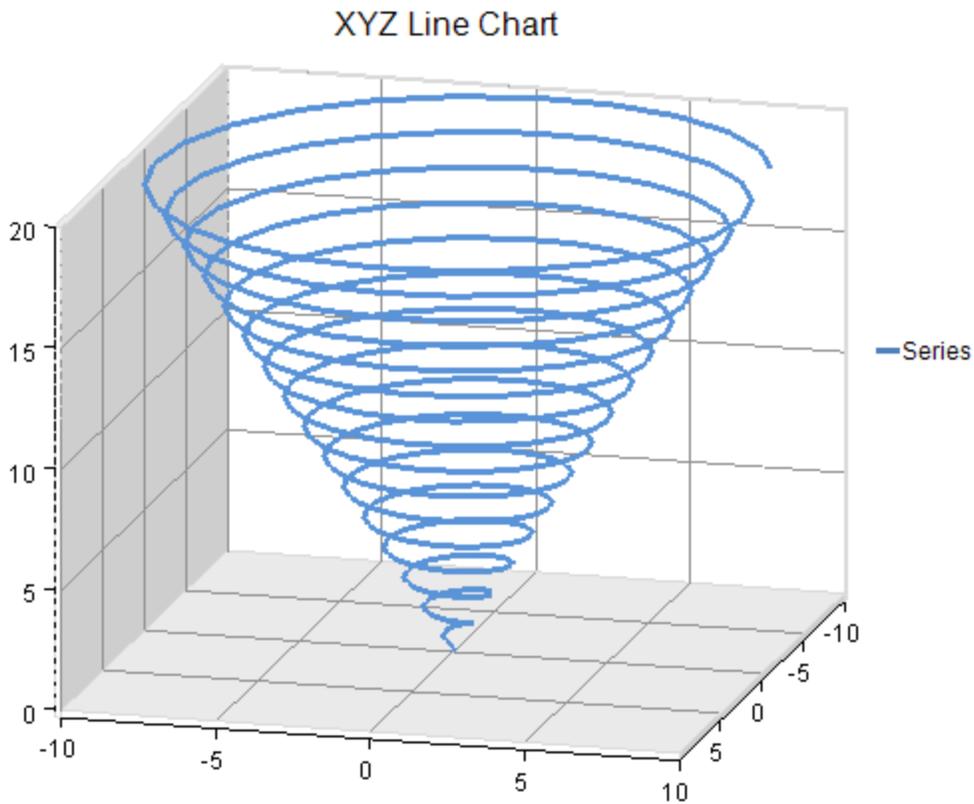
XYZ Plot Types

The XYZ plot area contains series that have values in three dimensions. When visualized in 2D, the XYZ plot area takes the form of a rectangle with a horizontal x-axis representing values and a vertical y-axis representing values. When visualized in 3D, the XYZ plot area takes the form of a cube with a horizontal x-axis representing values, a vertical y-axis representing values, and a depth z-axis representing values.

The Elevation and Rotation properties in the plot area class can be used to make the z-axis visible.

If an XYZ plot area has multiple x, y, or z-axes then the series can be assigned to a specific axis using the axis's ID. There are three subtypes of XYZ series: XYZ point, XYZ line, and XYZ surface.

The figure below is an example of an XYZ Line Chart.



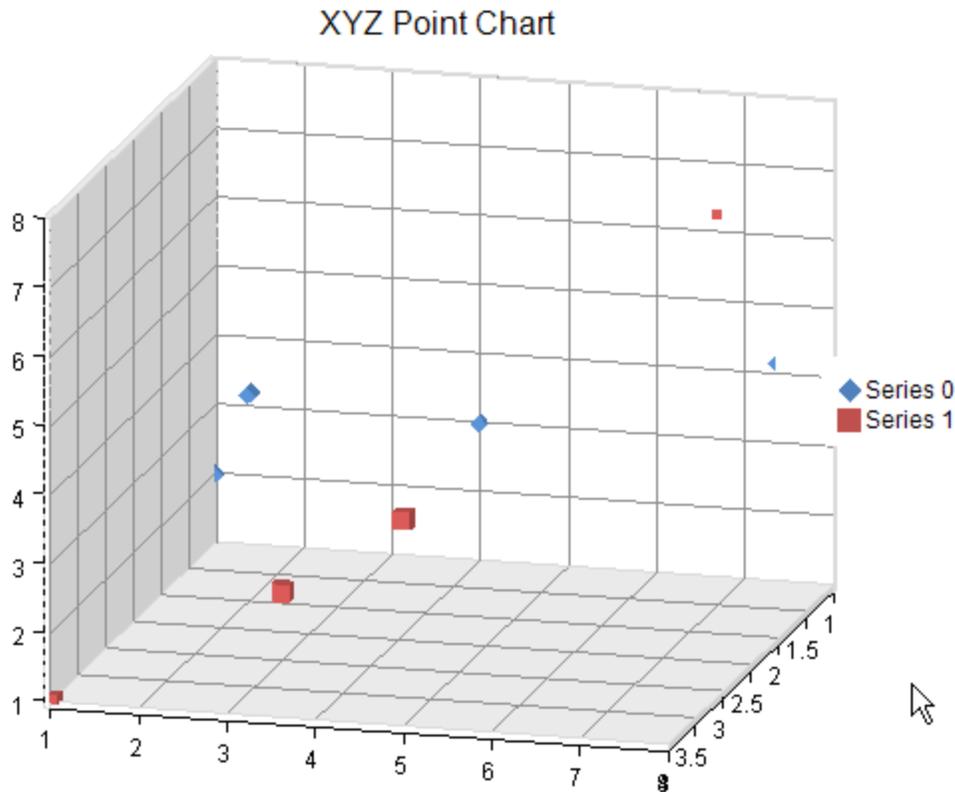
You can have any of these types of XYZ plots.

- **XYZ Point Charts**
- **XYZ Line Charts**
- **XYZ Surface Charts**
- **XYZ Stripe Charts**

For details on the API, see the [XYZPlotArea](#) class.

XYZ Point Charts

The point chart can be an XYZ plot such as the one shown in this figure.



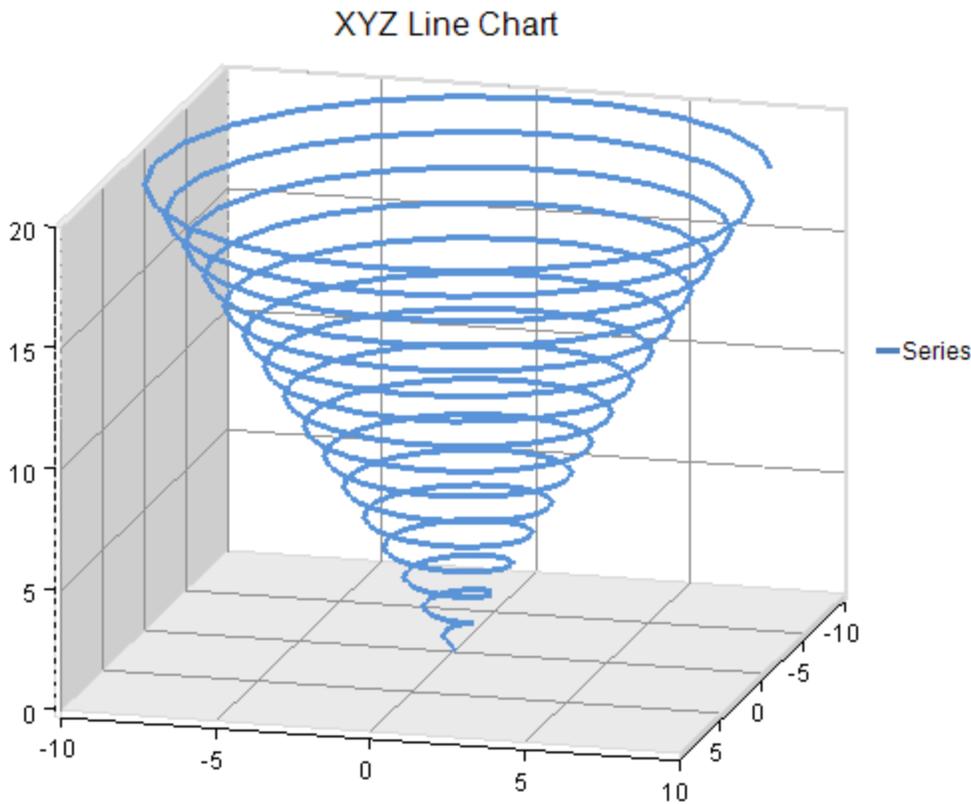
The point markers in an XYZ series or the series can be assigned a border, fill effect, shape, and a size. Settings at the point level have precedence. Assigning null for a border or fill effect indicates that the property is unset. Size is measured in model units.

Each point in an XYZ point series has three data values: x, y, and z. Each point is visualized as point marker.

For more information on the point series object in the API, refer to the **XYZPointSeries ('XYZPointSeries Class' in the on-line documentation)** class.

XYZ Line Charts

The point chart can be an XYZ plot such as the one shown in this figure.



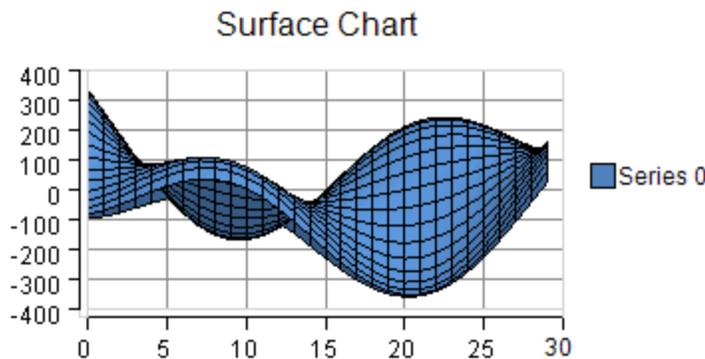
The point markers in an XYZ series or the series can be assigned a border, fill effect, shape, and a size. Settings at the point level have precedence.

Each point in an XYZ point series has three data values: x, y, and z. Each point is visualized as point marker. Assigning null for a border or fill effect indicates that the property is unset. Size is measured in model units.

For more information on the point series object in the API, refer to the **XYZLineSeries ('XYZLineSeries Class' in the on-line documentation)** class.

XYZ Surface Charts

The surface chart can be an XYZ plot such as the one shown in this figure.



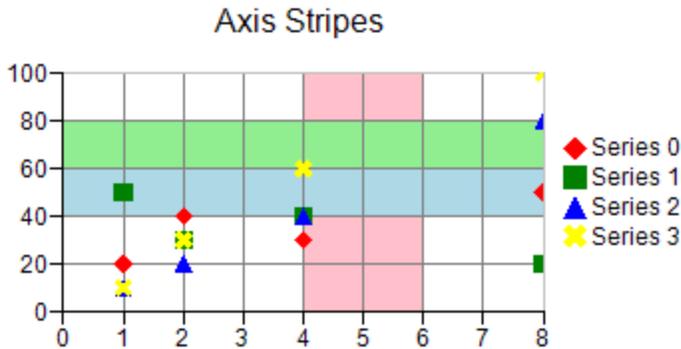
An XYZ surface series can be assigned a fill effect for the surface. Assigning null for the fill effect indicates that the property is unset.

Each point in a XYZ series has three data values: x, y, and z. Each point is visualized as a point on a surface.

For more information on the surface series object in the API, refer to the **XYZSurfaceSeries ('XYZSurfaceSeries Class' in the on-line documentation)** class.

XYZ Stripe Charts

The stripe chart can be an XYZ plot such as the one shown in this figure.



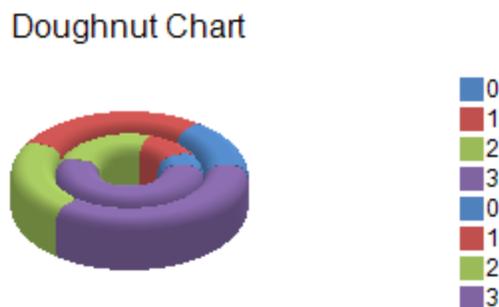
You can specify a start and end value for the stripe on the chart axis. You can also specify a fill type and color for the stripe. A stripe can be added to a chart with the axis properties in the appropriate plot area class.

For more information on the stripe object in the API, refer to the **Stripe ('Stripe Class' in the on-line documentation)** class.

Pie Plot Types

A pie plot area contains series that have values in one dimension. When visualized in two dimensions, a pie plot area takes the form of a circle (or partial circle). When visualized in three dimensions, a pie plot area takes the form of a disk (or partial disk).

The following image displays a three dimensional chart:



You can have any of these types of Pie plots.

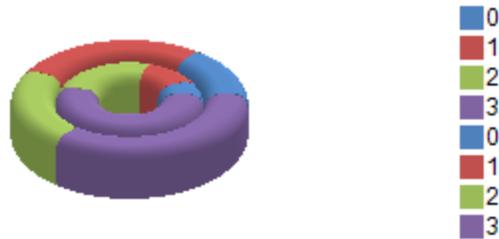
- Doughnut Charts
- Pie Charts

For details on the API, see the **PiePlotArea ('PiePlotArea Class' in the on-line documentation)** class.

Doughnut Charts

The doughnut chart can be a pie plot such as the one shown in this figure.

Doughnut Chart



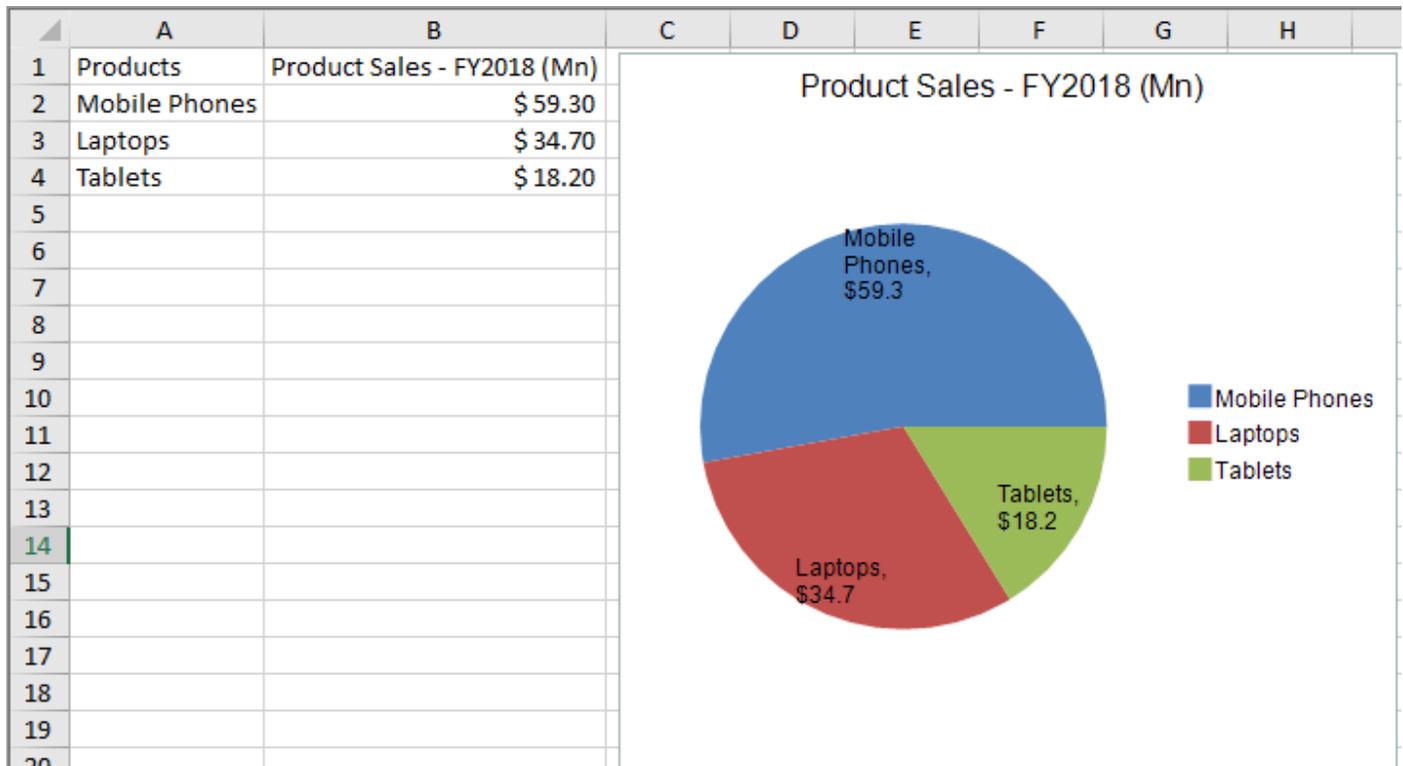
Each point can be assigned a border, fill effect, and a detachment distance for the pie slice. Assigning null for a border or fill effect indicates that the property is null.

Each point can be assigned a border, fill effect, and a detachment distance for the pie slice. Detachment distance is measured relative to pie radius (0 = no detachment, 1 = detachment is length of pie radius). The detachment distance (PieDetachments property) is used to create an exploded doughnut chart.

For more information on the pie series object in the API, refer to the **PieSeries ('PieSeries Class' in the on-line documentation)** class.

Pie Charts

The pie chart can be a pie plot such as the one shown in this figure.



A pie series can be assigned a border and fill effect for the pie slices. Assigning null for a border or fill effect indicates that the property is null.

Each point can be assigned a border, fill effect, and a detachment distance for the pie slice. Detachment distance is measured relative to pie radius (0 = no detachment, 1 = detachment is length of pie radius). The detachment distance (`PieDetachments` property) is used to create an exploded pie chart.

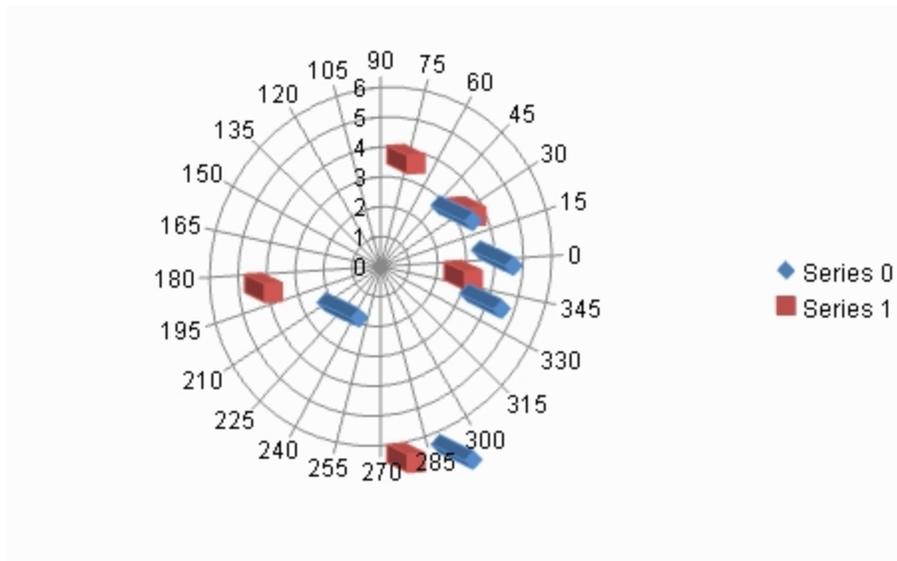
For more information on the pie series object in the API, refer to the **PieSeries ('PieSeries Class' in the on-line documentation)** class.

Polar Plot Types

A polar plot area contains series that have values in two dimensions (angle and radius). When visualized in two dimensions, a polar plot area takes the form of a circle with a circular x-axis representing angle values and a radial y-axis representing radius values. When visualized in three dimensions, a polar plot area takes the form of a disk with a circular x-axis representing an angle value and a radial y-axis representing a radius value.

A polar series is displayed in a polar plot area. Points have value(s) in two dimensions: x (angle) and y (radius). If a polar plot area has multiple y-axes then a series can be assigned to a specific axis using the axis's ID. There are several subtypes of polar series: polar point, polar line, polar area, and polar stripe.

The following image shows a three dimensional polar point chart that was created by using the `Elevation`, `Rotation`, and `ViewType` properties.



You can have any of these types of Polar plots.

- **Polar Point Charts**
- **Polar Line Charts**
- **Polar Area Charts**
- **Polar Stripe Charts**
- **Pie Charts**

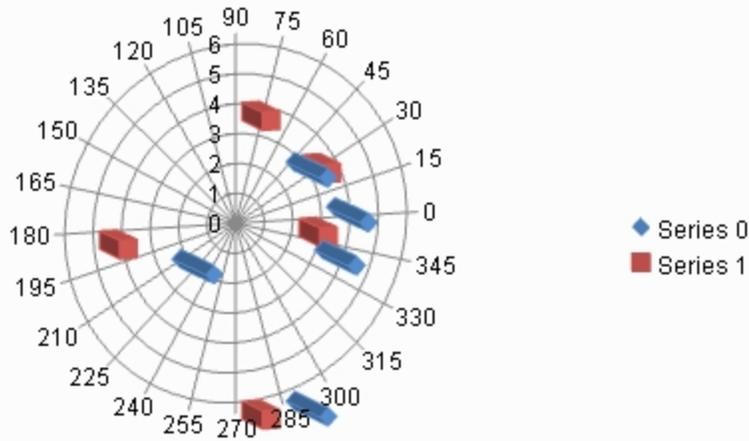
For details on the API, see the **PolarPlotArea ('PolarPlotArea Class' in the on-line documentation)** class.

Polar Point Charts

The point markers in the polar point series or the series can be assigned a border, fill effect, shape, size, and a depth. Assigning null for a border or fill effect indicates that the property is unset. The size of the point marker is measured in model units. The depth of the point marker is measured relative to plot area depth (0 = no depth, 1 = full depth of plot area).

Each point has two data values: x (angle) and y (radius). Each point is visualized as a point marker. You can also specify individual point marker borders and fill effects for each data point.

The point chart can be a polar plot such as the one shown in the figure:



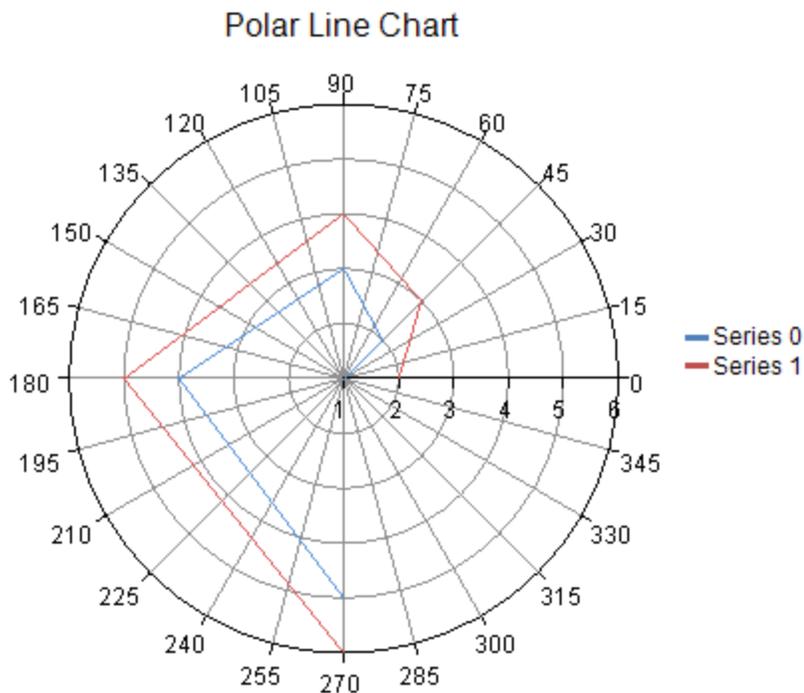
For more information on the point series object in the API, refer to the **PolarPointSeries ('PolarPointSeries Class' in the on-line documentation)** class.

Polar Line Charts

A polar line series or each point in the series can be assigned a border, fill effect, and a depth for the line. You can also specify whether the line is closed. Assigning null for a border or fill effect indicates that the property is unset. Depth is measured relative to plot area depth (0 = no depth, 1 = full depth of plot area).

Each point has two data values: x (angle) and y (radius). Each point is visualized as a point on a line. You can also specify individual point marker borders and fill effects for each data point.

The line chart can be a polar plot such as the one shown in the following figure:



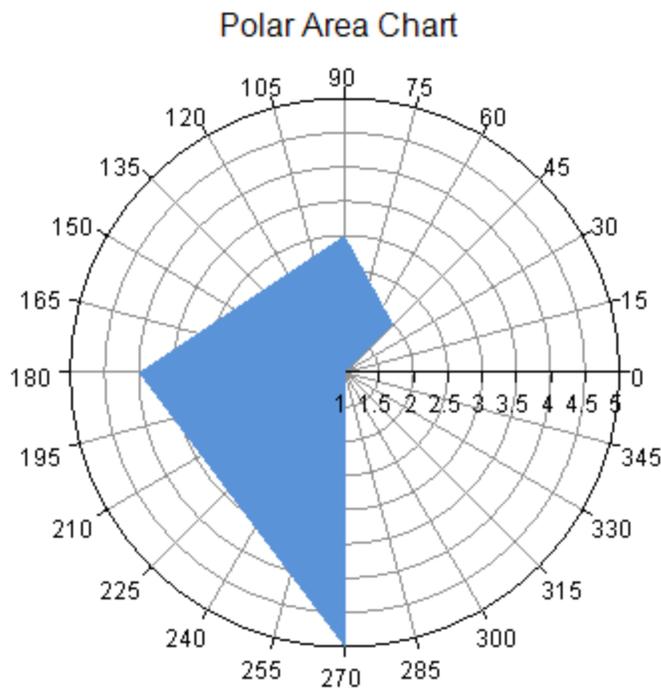
For more information on the line series object in the API, refer to the **PolarLineSeries ('PolarLineSeries Class' in the on-line documentation)** class.

Polar Area Charts

A polar area series can be assigned a border, fill effect, and a depth for the area. Each point can be assigned a border and a fill effect for the area. You can also specify whether the area is closed. Assigning null for a border or fill effect indicates that the property is unset. Depth is measured relative to the plot area depth (0 = no depth, 1 = full depth of plot area).

Each point has two data values: x (angle) and y (radius). Each point is visualized as a point on an area. You can also specify individual point marker borders and fill effects for each data point.

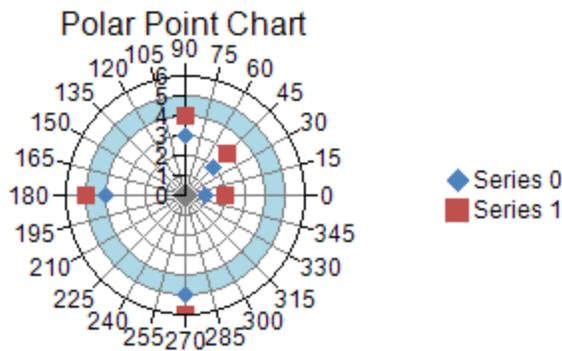
The area chart can be a polar plot such as the one shown in the following figure:



For more information on the area series object in the API, refer to the **PolarAreaSeries ('PolarAreaSeries Class' in the on-line documentation)** class.

Polar Stripe Charts

The stripe chart can be a polar plot such as the one shown in this figure.



You can specify a start and end value for the stripe on the chart axis. You can also specify a fill type and color for the stripe. A stripe can be added to a chart with the axis properties in the appropriate plot area class.

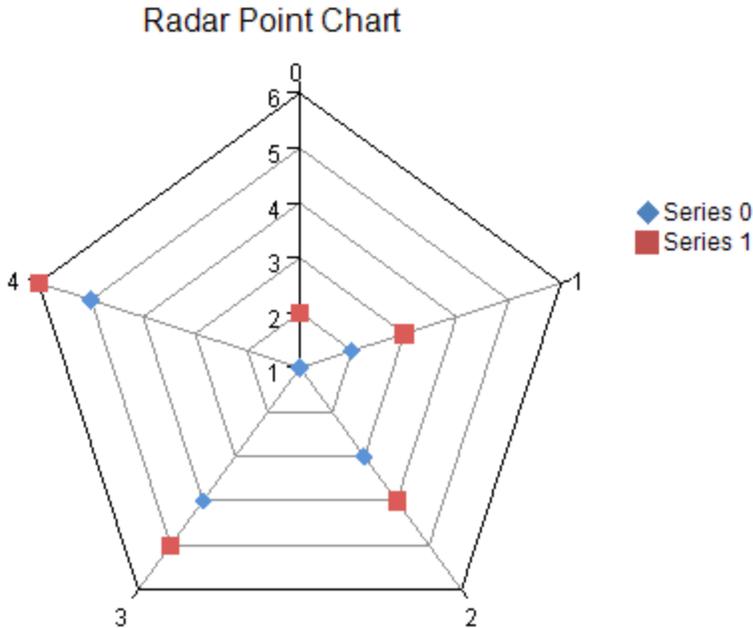
For more information on the stripe object in the API, refer to the **Stripe ('Stripe Class' in the on-line documentation)** class.

Radar Plot Types

A radar plot area contains series that have values in one dimension. When visualized in two dimensions, a radar plot area takes the form of an n-sided polygon with a circular x-axis representing categories and a radial y-axis representing values. When visualized in three dimensions, a radar plot area takes the form of an n-sided disk with a circular x-axis representing categories and a radial y-axis representing values.

A radar series is displayed in a radar plot area. Each point has value(s) in one dimension: y (radius). If a plot area has multiple y-axes then a series can be assigned to a specific axis using the axis's ID. There are several subtypes of radar series: radar point, radar line, radar area, and radar stripe.

The figure below is an example of a radar chart.



You can have any of these types of Radar plots.

- **Radar Point Charts**
- **Radar Line Charts**
- **Radar Area Charts**
- **Radar Stripe Charts**

For details on the API, see the **RadarPlotArea ('RadarPlotArea Class' in the on-line documentation)** class.

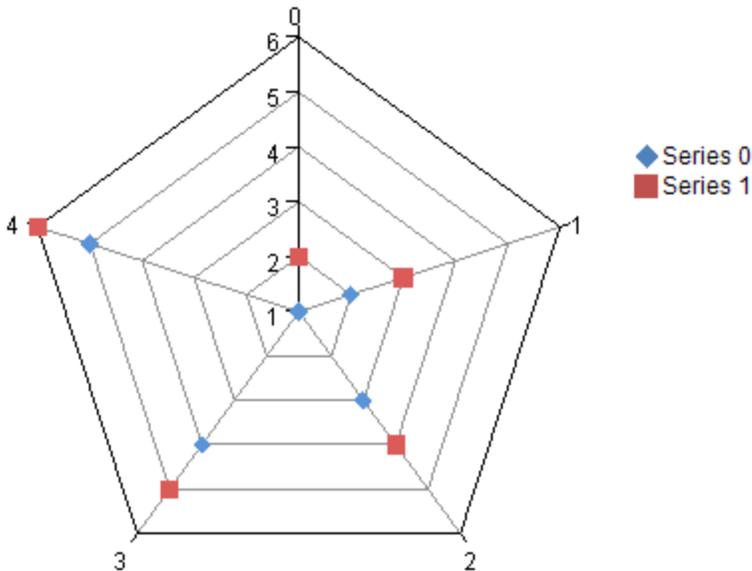
Radar Point Charts

The point markers in a radar point series or the series can be assigned a border, fill effect, shape, size, and a depth. Assigning null for a border or fill effect indicates that the property is unset. Size is measured in model units. Depth is measured relative to the plot area depth (0 = no depth, 1 = depth of plot area).

Each point has a single data value: y. Each point is visualized as a point marker. You can also specify individual point marker borders and fill effects for each data point.

The point chart can be a radar plot such as the one shown in this figure:

Radar Point Chart



For more information on the point series object in the API, refer to the **RadarPointSeries ('RadarPointSeries Class' in the on-line documentation)** class.

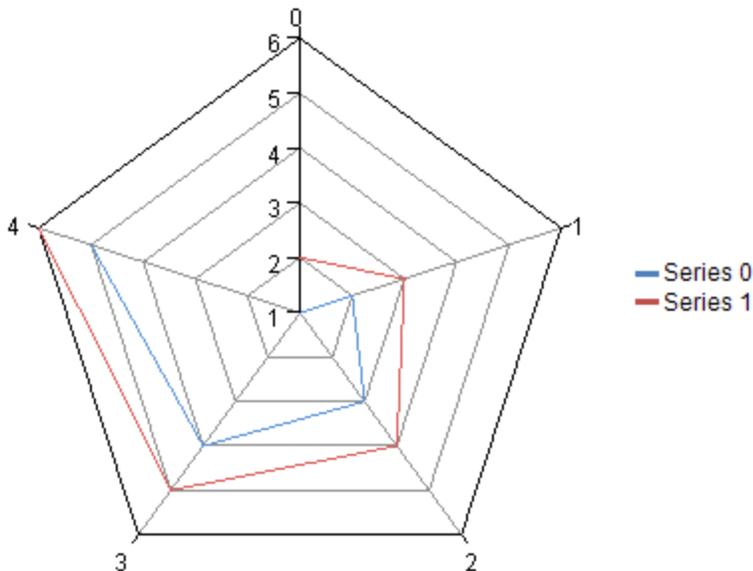
Radar Line Charts

A radar line series can be assigned a border, fill effect, and a depth for the line. Each point can be assigned a border and a fill effect for the line. Assigning null for a border or fill effect indicates that the property is unset. Depth is measured relative to the plot area depth (0 = no depth, 1 = depth of plot area).

Each point has a single data value: y. Each point is visualized as point on a line.

The line chart can be a radar plot such as the one shown in this figure:

Radar Line Chart



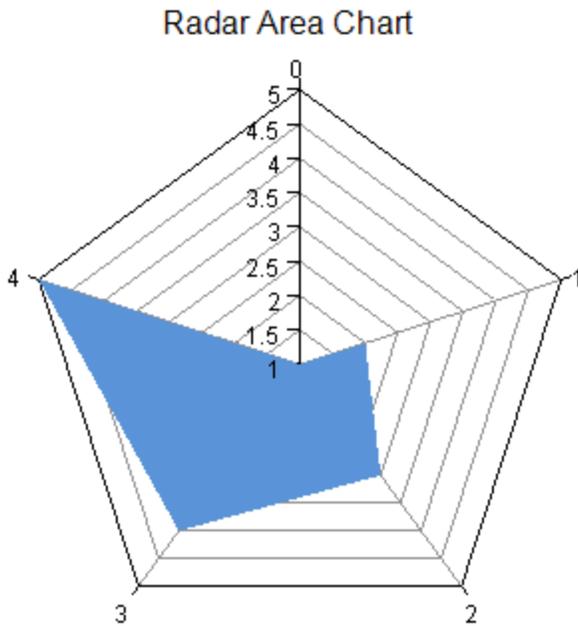
For more information on the line series object in the API, refer to the **RadarLineSeries ('RadarLineSeries Class' in the on-line documentation)** class.

Radar Area Charts

A radar area series can be assigned a border, fill effect, and a depth for the area. Each point can be assigned a border and a fill effect for the area. Assigning null for the border or fill effect indicates that the property is unset. Depth is measured relative to the plot area depth (0 = no depth, 1 = depth of plot area).

Each point has a single data value: y. Each point is visualized as a point on an area. You can specify area borders and fill effects for each data point.

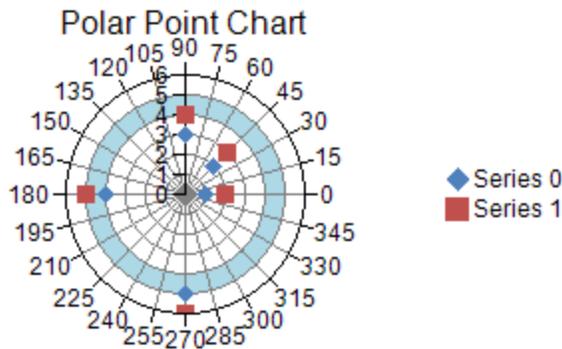
The area chart can be a radar plot such as the one shown in this figure:



For more information on the line series object in the API, refer to the **RadarAreaSeries ('RadarAreaSeries Class' in the on-line documentation)** class.

Radar Stripe Charts

The stripe chart can be a radar plot such as the one shown in this figure.



You can specify a start and end value for the stripe on the chart axis. You can also specify a fill type and color for the

stripe. A stripe can be added to a chart with the axis properties in the appropriate plot area class.

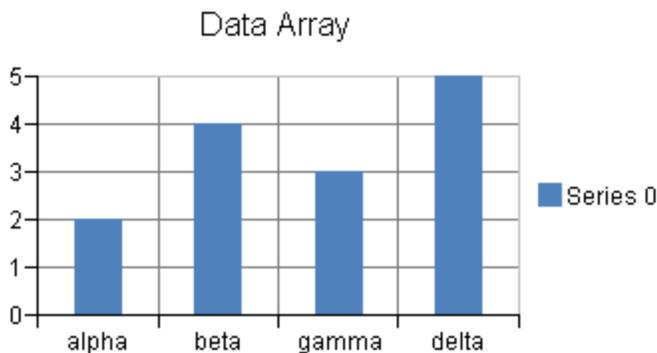
For more information on the stripe object in the API, refer to the **Stripe ('Stripe Class' in the on-line documentation)** class.

Data Plot Types

Data charts are charts that can be bound. Any series in any of the plot types can be bound to a data source using the data source property in the series class. You can use any of these types of data sources.

- Array
- List
- Table

The array data chart can be a one-dimensional plot such as the one shown in this figure. This array chart shows the bars alongside each other vertically.



See **Using a Bound Data Source** for more information.

Plot area

A plot area is the area in which data points (bars, points, lines, etc) are drawn. A plot area will contain a collection of series. Each series is a collection of data points. A plot area may also contain an axis(s) and wall(s). The axis(s) and wall(s) enhance the visual appearance of the plot area and are usually painted just outside the plot area.

A plot area can be assigned an anchor view location and a view size. The anchor view location is specified in normalized units ((0,0) = left upper corner of chart view, (1,1) = right lower corner of chart view). The view size of the plot area is specified in normalized units ((0,0) = zero size, (1,1) = full size of chart view). The left top corner of the plot area is aligned with the anchor location.

The elements that can be drawn in the plot area are explained below.

- **Series**
- **Walls**
- **Axis**
- **Chart Line Style**
- **Elevation and Rotation**
- **Lighting, Shapes, and Borders**

Example

The following example adds a series of bar charts to plot area and sets the display position and size of the plot area.

C#

```
//Create a series of bar charts
FarPoint.Win.Chart.BarSeries series = new FarPoint.Win.Chart.BarSeries();
series.SeriesName = "Series 0";
series.Values.Add(2.0);
series.Values.Add(4.0);
series.Values.Add(3.0);
series.Values.Add(5.0);
//Create plot area
FarPoint.Win.Chart.YPlotArea plotArea = new FarPoint.Win.Chart.YPlotArea();
plotArea.Location = new PointF(0.2f, 0.2f);
plotArea.Size = new SizeF(0.6f, 0.6f);
plotArea.Series.Add(series);
```

Visual Basic

```
'Create a series of bar charts
Dim series As New FarPoint.Win.Chart.BarSeries()
series.SeriesName = "Series 0"
series.Values.Add(2.0)
series.Values.Add(4.0)
series.Values.Add(3.0)
series.Values.Add(5.0)
'Create plot area
Dim plotArea As New FarPoint.Win.Chart.YPlotArea()
plotArea.Location = New PointF(0.2F, 0.2F)
plotArea.Size = New SizeF(0.6F, 0.6F)
plotArea.Series.Add(series)
```

Using the Spread Designer

1. Start the Chart Designer.
2. Select the [Plot Area] from the tree menu on the left side.
3. Set the required properties in the Property list on the right side.
4. Click <OK> to exit Chart Designer.

 For information on starting **Chart Designer ('Using the Chart Designer' in the on-line documentation)**, refer to Chart Designer in the **Spread Designer Guide (on-line documentation)**.

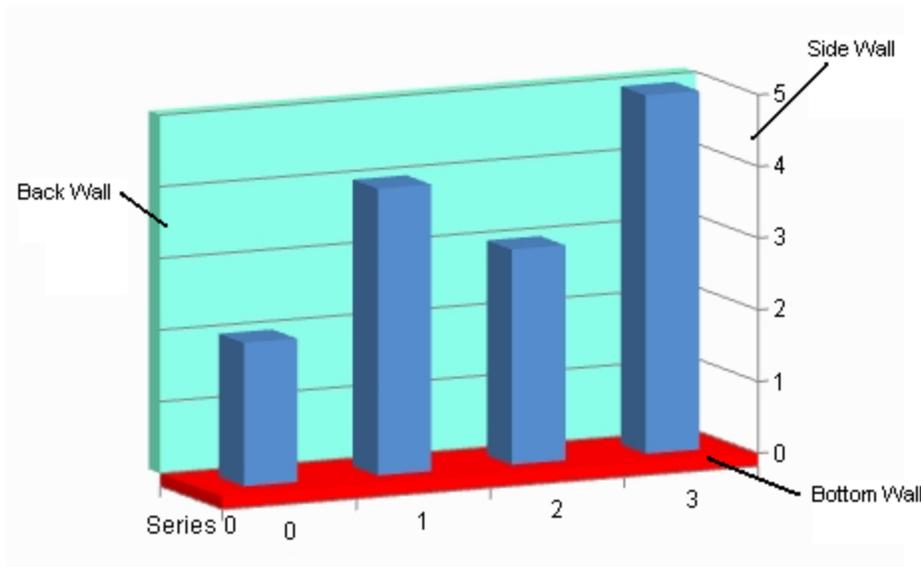
Series

A series is a collection of data points that are displayed in the plot area. There are several types of series, depending on the type of plot.

- 
- Area, Line, and Point series can have drop lines. Drop lines extend from the data point down to the series origin or category axis. This can be used to highlight the x value of the point. There are several subtypes of series such as: Y, XY, XYZ, Pie, Polar, and Radar. These subtypes of series correspond to the subtypes of plot areas.
 - When a chart has multiple legends, a series can be assigned to a specific legend using the legend's ID.
 - Many of the series have properties (e.g. border or fill effect) that can be assigned to both a series and a point.
 - If the property is set for both the series and the point then the point setting overrides the series setting. If the property is unset for both the series and the point then the chart view will provide a default setting.

Walls

A wall is the area (or plane) behind, below, or to the side of a chart.



A wall can have a border, fill effect, or width (measured in model units). The wall can be visible or hidden. The axis grids (major and minor) and stripes are painted on the walls. The axis grid lines (major and minor) are painted over the axis stripes.

See the following for more information on how to set properties for walls:

- **Wall ('Wall Class' in the on-line documentation)**
- **YPlotArea ('YPlotArea Class' in the on-line documentation)**

Example

The following example shows the back of the plot area and fills it with a solid color.

C#

```
FarPoint.Win.Chart.YPlotArea plotArea = new FarPoint.Win.Chart.YPlotArea();  
plotArea.BackWall.Visible = true;  
plotArea.BackWall.Fill = new FarPoint.Win.Chart.SolidFill(Color.Aquamarine);
```

Visual Basic

```
Dim plotArea As New FarPoint.Win.Chart.YPlotArea()  
plotArea.BackWall.Visible = True  
plotArea.BackWall.Fill = New FarPoint.Win.Chart.SolidFill(Color.Aquamarine)
```

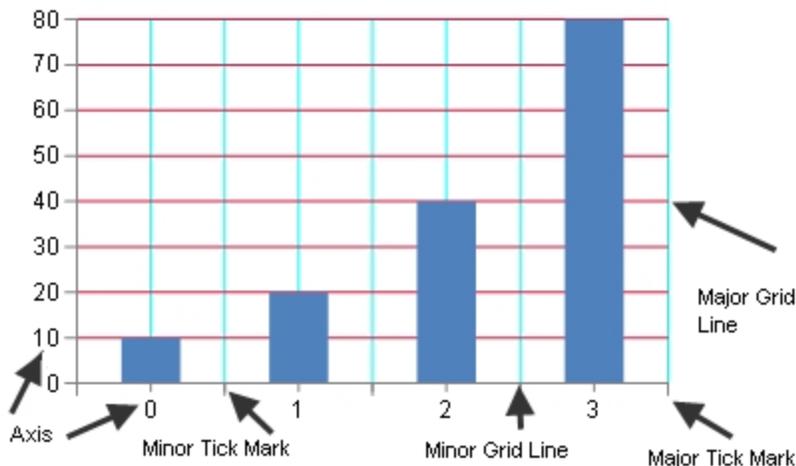
Using a Chart designer

1. Run the **Chart Designer**.
2. Select the target **Plot Area** from the tree menu on the left.
3. From the **Other** section of the property list on the right, expand the **BackWall** tree and set its properties.
4. Click OK and close the **Chart Designer**.

For information on starting **Chart Designer ('Spread Designer Guide' in the on-line documentation)**, refer to Chart Designer in the **SPREAD Designer Guide ('Using the Chart Designer' in the on-line documentation)**.

Axis

An axis is used to display the scale for a single dimension of a plot area. An axis can have a title, a ruler line, major and minor tick marks, tick mark labels, major and minor grid lines, and stripes. The direction of the axis can be reversed. The minimum, maximum, major or minor tick, and label units can be automatically generated or manually assigned. The scale can be linear or logarithmic.



Tick marks and grids are used to mark individual values on the ruler. Tick marks are painted on the ruler while corresponding grids are painted on the wall(s). Stripes are used to highlight ranges of values. Stripes are painted on the wall(s).

- The title, ruler, tick marks (major and minor), tick mark labels, and grids (major and minor) can be hidden.
- A font can be set for the title and tick marks and the title can be customized. The title and tick mark labels can have fill effects.
- Length (measured in model units), can also be set for major and minor tick marks.
- The axis can have lines for the ruler, major, and minor grids as well as a minimum and maximum value. The minimum and maximum values can be automatically generated by the chart view.
- An axis can be assigned a collection of stripes.
- The direction of the axis can be reversed.

For more information, see the following classes:

- **IndexAxis**
An index axis is used to display integer values such as a category or series index. Tick marks, tick mark labels, and grid lines can be displayed on the integer values or between the integer values. Index axis is set using the **IndexAxis ('IndexAxis Class' in the on-line documentation)** class.
- **ValueAxis**
A value axis is used to display double values (data values). The value axis can have a linear or logarithmic scale (when using a logarithmic scale, the value axis can use a logarithmic base). Value axis is set using the **ValueAxis ('ValueAxis Class' in the on-line documentation)** class.

Example

The following example sets a title for the axis.

C#

```
FarPoint.Win.Chart.YPlotArea plotArea = new FarPoint.Win.Chart.YPlotArea();
plotArea.Location = new PointF(0.2f, 0.2f);
plotArea.Size = new SizeF(0.6f, 0.6f);
plotArea.XAxis.Title = "Categories";
plotArea.YAxes[0].Title = "Values";
```

Visual Basic

```
Dim plotArea As New FarPoint.Win.Chart.YPlotArea()
plotArea.Location = New PointF(0.2F, 0.2F)
plotArea.Size = New SizeF(0.6F, 0.6F)
plotArea.XAxis.Title = "Categories"
plotArea.YAxes(0).Title = "Values"
```

Using SPREAD Designer

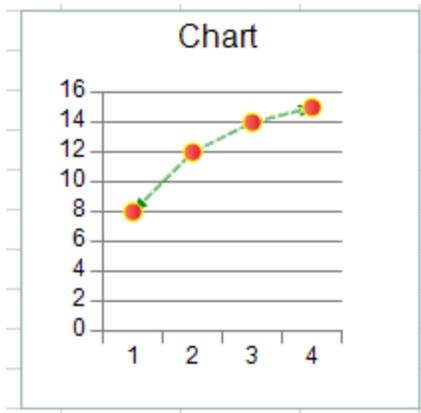
1. Run the **Chart Designer**.
2. Expand the target **Plot Area** from the tree menu on the left.
3. Select **X Index Axis** and set the required properties in the property list on the right.
4. Expand **Y Value Axis**, select **Axis0** and set the required properties in the property list on the right.
5. Click **OK** to exit **Chart Designer**.



For information on starting **Chart Designer** ('**Spread Designer Guide**' in the on-line documentation), refer to **Chart Designer** in the **SPREAD Designer Guide** ('**Using the Chart Designer**' in the on-line documentation).

Chart Line Style

You can create a line style with special options such as open and end arrows for the line chart with the **LineBorder** ('**LineBorder Property**' in the on-line documentation) property and the **EnhancedSolidLine** ('**EnhancedSolidLine Class**' in the on-line documentation) class. You can also specify line style options such as dash, cap type, and so on.



Using Code

1. Create a line chart.
2. Create an **EnhancedSolidLine** ('**EnhancedSolidLine Class**' in the on-line documentation) object.
3. Set the **LineBorder** ('**LineBorder Property**' in the on-line documentation) property.

Example

This example code creates a line chart with a line style that contains arrows.

C#

```
FarPoint.Win.Chart.EnhancedSolidLine eh = new
FarPoint.Win.Chart.EnhancedSolidLine(System.Drawing.Color.Green, 1,
FarPoint.Win.Chart.CompoundType.Double, FarPoint.Win.Chart.DashType.Dash,
FarPoint.Win.Chart.CapType.Flat, FarPoint.Win.Chart.JoinType.Round,
FarPoint.Win.Chart.ArrowType.Arrow, FarPoint.Win.Chart.ArrowType.OpenArrow, 1, 2);
FarPoint.Win.Chart.LineSeries series1 = new FarPoint.Win.Chart.LineSeries();
series1.PointMarker = new
FarPoint.Win.Chart.BuiltinMarker(FarPoint.Win.Chart.MarkerShape.Circle, 7.0f);
series1.PointFill = new FarPoint.Win.Chart.GradientFill(System.Drawing.Color.Coral,
System.Drawing.Color.Crimson);
series1.LineBorder = new FarPoint.Win.Chart.SolidLine(System.Drawing.Color.Yellow);
series1.LineBorder = eh;
series1.Values.Add(8.0);
series1.Values.Add(12.0);
series1.Values.Add(14.0);
series1.Values.Add(15.0);
FarPoint.Win.Chart.YPlotArea plotArea = new FarPoint.Win.Chart.YPlotArea();
plotArea.Location = new System.Drawing.PointF(0.2f, 0.2f);
plotArea.Size = new System.Drawing.SizeF(0.6f, 0.6f);
plotArea.Series.Add(series1);
FarPoint.Win.Chart.LabelArea labelArea = new FarPoint.Win.Chart.LabelArea();
labelArea.Location = new System.Drawing.PointF(0.5f, 0.02f);
labelArea.AlignmentX = 0.5f;
labelArea.AlignmentY = 0.0f;
labelArea.Text = "Chart";
FarPoint.Win.Chart.ChartModel model = new FarPoint.Win.Chart.ChartModel();
model.LabelAreas.Add(labelArea);
model.PlotAreas.Add(plotArea);
FarPoint.Win.Spread.Chart.SpreadChart chart = new
FarPoint.Win.Spread.Chart.SpreadChart();
chart.Size = new Size(200, 200);
chart.Location = new Point(100, 100);
chart.Model = model;
fpSpread1.Sheets[0].Charts.Add(chart);
```

VB

```
Dim eh As New FarPoint.Win.Chart.EnhancedSolidLine(System.Drawing.Color.Green, 1,
FarPoint.Win.Chart.CompoundType.Double, FarPoint.Win.Chart.DashType.Dash,
FarPoint.Win.Chart.CapType.Flat, FarPoint.Win.Chart.JoinType.Round,
FarPoint.Win.Chart.ArrowType.Arrow, FarPoint.Win.Chart.ArrowType.OpenArrow, 1, 2)
Dim series1 As New FarPoint.Win.Chart.LineSeries()
series1.PointMarker = New
FarPoint.Win.Chart.BuiltinMarker(FarPoint.Win.Chart.MarkerShape.Circle, 7.0F)
series1.PointFill = New FarPoint.Win.Chart.GradientFill(System.Drawing.Color.Coral,
System.Drawing.Color.Crimson)
```

```
series1.PointBorder = New FarPoint.Win.Chart.SolidLine(System.Drawing.Color.Yellow)
series1.LineBorder = eh
series1.Values.Add(8.0)
series1.Values.Add(12.0)
series1.Values.Add(14.0)
series1.Values.Add(15.0)
Dim plotArea As New FarPoint.Win.Chart.YPlotArea()
plotArea.Location = New System.Drawing.PointF(0.2F, 0.2F)
plotArea.Size = New System.Drawing.SizeF(0.6F, 0.6F)
plotArea.Series.Add(series1)
Dim labelArea As New FarPoint.Win.Chart.LabelArea()
labelArea.Location = New System.Drawing.PointF(0.5F, 0.02F)
labelArea.AlignmentX = 0.5F
labelArea.AlignmentY = 0.0F
labelArea.Text = "Chart"
Dim model As New FarPoint.Win.Chart.ChartModel()
model.LabelAreas.Add(labelArea)
model.PlotAreas.Add(plotArea)
Dim chart As New FarPoint.Win.Spread.Chart.SpreadChart()
chart.Size = New Size(200, 200)
chart.Location = New Point(100, 100)
chart.Model = model
fpSpread1.Sheets(0).Charts.Add(chart)
```

Using the Chart Designer

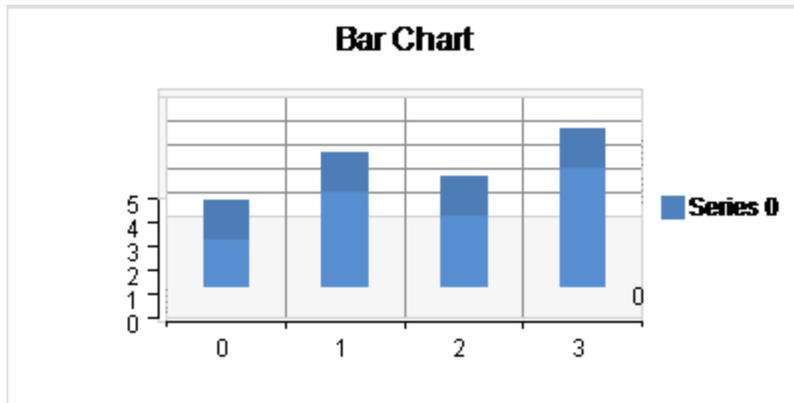
1. Right-click on the line in the Line chart.
2. Select the Format Series menu.
3. Select the Line Border option.
4. Select Solid Line.
5. Set properties and close the dialog.

Elevation and Rotation

You can specify the elevation or rotation for a chart.

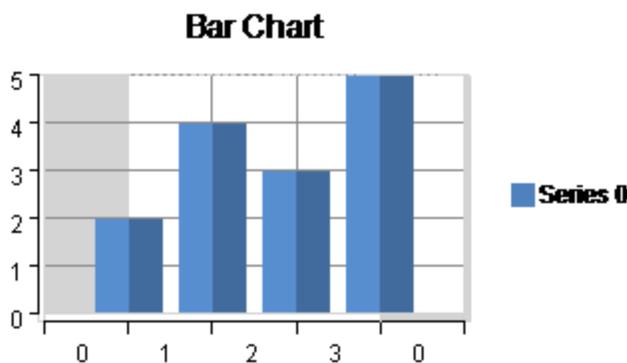
Elevation (X rotation)

The elevation rotates the graph counterclockwise around the horizontal axis. Set the **Elevation ('Elevation Property' in the on-line documentation)** property of the **PlotArea ('PlotArea Class' in the on-line documentation)** class to set the X rotation angle. The following image displays a graph with a changed elevation.



Rotation (Y rotation)

The rotation rotates the graph counterclockwise around the vertical axis. Set the **Rotation ('Rotation Property' in the on-line documentation)** property of the PlotArea class to set the Y rotation angle . The following image displays a graph with a changed rotation.



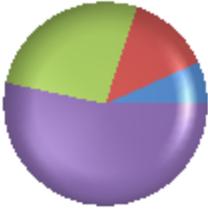
Lighting, Shapes, and Borders

You can set borders for most areas of the chart. See the **LineBorder** and **PointBorder** properties in the **LineSeries ('LineSeries Class' in the on-line documentation)** class for more information. The **XYBubbleSeries ('XYBubbleSeries Class' in the on-line documentation)** class has additional border settings such as **NegativeBorder** and **PositiveBorder**.

You can set shapes such as the bar shape. See the **BarShape ('BarShape Property' in the on-line documentation)** property for more information.

You can apply additional effects to the 3D chart control such as color, directional lighting, and positional lighting. Directional lighting mimics a distant light source such as rays from the sun (parallel paths). Positional lighting mimics a close light source such as a lamp where the light radiates out from a single point.

The following image displays a graph that uses light colors, direction, and position:



The following color effects are available:

- **AmbientColor** ('AmbientColor Property' in the on-line documentation)
- **DiffuseColor** ('DiffuseColor Property' in the on-line documentation)
- **SpecularColor** ('SpecularColor Property' in the on-line documentation)

You can specify the position and the direction of the light with the following properties:

- **PositionX** ('PositionX Property' in the on-line documentation)
- **PositionY** ('PositionY Property' in the on-line documentation)
- **PositionZ** ('PositionZ Property' in the on-line documentation)
- **DirectionX** ('DirectionX Property' in the on-line documentation)
- **DirectionY** ('DirectionY Property' in the on-line documentation)
- **DirectionZ** ('DirectionZ Property' in the on-line documentation)

Using Code

1. Create a series.
2. Add values to the series.
3. Create a plot area.
4. Set the **AmbientColor**, **DiffuseColor**, and **SpecularColor** in the **PositionalLight** ('PositionalLight Class' in the on-line documentation) class.
5. Set the **PositionX**, **PositionY**, and **PositionZ** properties in the **PositionalLight** ('PositionalLight Class' in the on-line documentation) class.
6. Set the **AmbientColor**, **DiffuseColor**, and **SpecularColor** in the **DirectionalLight** ('DirectionalLight Class' in the on-line documentation) class.
7. Set the **PositionX**, **PositionY**, and **PositionZ** properties in the **DirectionalLight** ('DirectionalLight Class' in the on-line documentation) class.
8. Add the light settings to the plot area.
9. Create a chart model and assign the plot area settings to it.
10. Create a chart and assign the chart model to it.

Example

The following example demonstrates using light colors, direction, and position.

C#

```
FarPoint.Win.Chart.PieSeries series = new FarPoint.Win.Chart.PieSeries();
series.SeriesName = "Series 1";
series.TopBevel = new FarPoint.Win.Chart.CircleBevel(12.0f, 12.0f);
series.BottomBevel = new FarPoint.Win.Chart.CircleBevel(12.0f, 12.0f);
series.Values.Add(1.0);
series.Values.Add(2.0);
```

```
series.Values.Add(4.0);
series.Values.Add(8.0);
FarPoint.Win.Chart.PiePlotArea plotArea = new FarPoint.Win.Chart.PiePlotArea();
plotArea.Location = new PointF(0.2f, 0.2f);
plotArea.Size = new SizeF(0.6f, 0.6f);
plotArea.Series.Add(series);
FarPoint.Win.Chart.PositionalLight light0 = new FarPoint.Win.Chart.PositionalLight();
light0.AmbientColor = Color.FromArgb(64, 64, 64);
light0.DiffuseColor = Color.FromArgb(64, 64, 64);
light0.SpecularColor = Color.FromArgb(128, 128, 128);
light0.PositionX = 0.0f;
light0.PositionY = 0.0f;
light0.PositionZ = 100.0f;
FarPoint.Win.Chart.DirectionallLight light1 = new FarPoint.Win.Chart.DirectionallLight();
light1.AmbientColor = Color.FromArgb(64, 64, 64);
light1.DiffuseColor = Color.FromArgb(64, 64, 64);
light1.SpecularColor = Color.FromArgb(128, 128, 128);
light1.DirectionX = 1.0f;
light1.DirectionY = 0.0f;
light1.DirectionZ = 1.0f;
FarPoint.Win.Chart.ChartModel model = new FarPoint.Win.Chart.ChartModel();
model.PlotAreas.Add(plotArea);
model.PlotAreas[0].Lights.Clear();
model.PlotAreas[0].Lights.Add(light0);
model.PlotAreas[0].Lights.Add(light1);
fpChart1.Model = model;
```

VB

```
Dim series As New FarPoint.Win.Chart.PieSeries()
series.SeriesName = "Series 1"
series.TopBevel = New FarPoint.Win.Chart.CircleBevel(12.0F, 12.0F)
series.BottomBevel = New FarPoint.Win.Chart.CircleBevel(12.0F, 12.0F)
series.Values.Add(1.0)
series.Values.Add(2.0)
series.Values.Add(4.0)
series.Values.Add(8.0)
Dim plotArea As New FarPoint.Win.Chart.PiePlotArea()
plotArea.Location = New PointF(0.2F, 0.2F)
plotArea.Size = New SizeF(0.6F, 0.6F)
plotArea.Series.Add(series)
Dim light0 As New FarPoint.Win.Chart.PositionalLight()
light0.AmbientColor = Color.FromArgb(64, 64, 64)
light0.DiffuseColor = Color.FromArgb(64, 64, 64)
light0.SpecularColor = Color.FromArgb(128, 128, 128)
light0.PositionX = 0.0F
light0.PositionY = 0.0F
light0.PositionZ = 100.0F
Dim light1 As New FarPoint.Win.Chart.DirectionallLight()
light1.AmbientColor = Color.FromArgb(64, 64, 64)
light1.DiffuseColor = Color.FromArgb(64, 64, 64)
light1.SpecularColor = Color.FromArgb(128, 128, 128)
light1.DirectionX = 1.0F
light1.DirectionY = 0.0F
light1.DirectionZ = 1.0F
Dim model As New FarPoint.Win.Chart.ChartModel()
model.PlotAreas.Add(plotArea)
```

```
model.PlotAreas(0).Lights.Clear()
model.PlotAreas(0).Lights.Add(light0)
model.PlotAreas(0).Lights.Add(light1)
fpChart1.Model = model
```

Using the Chart Designer

1. Select the **Plot Areas Collection**.
2. Select the **Light Collection** editor.
3. Set properties as needed.

Labels

The labels contain the plot title and the axis labels.

Plot Title

You can set the main title for the chart using the Text property in the **LabelArea ('LabelArea Class' in the on-line documentation)** class.

Axis Label

You can set the text, alignment, and other formatting properties for the axis labels. See the following for more information:

- **IndexAxis ('IndexAxis Class' in the on-line documentation)**
- **ValueAxis ('ValueAxis Class' in the on-line documentation)**

The label text can be bound to a data source with the **TitleDataSource ('TitleDataSource Property' in the on-line documentation)** and **TitleDataField ('TitleDataField Property' in the on-line documentation)** properties.

The following example sets the chart and axis title.

C#

```
FarPoint.Win.Chart.YPlotArea plotArea = new FarPoint.Win.Chart.YPlotArea();
plotArea.Location = new System.Drawing.PointF(0.2f, 0.2f);
plotArea.Size = new System.Drawing.SizeF(0.6f, 0.6f);
//Sets the Axis label
plotArea.XAxis.Title = "Categories";
plotArea.YAxes[0].Title = "Values";
plotArea.YAxes[0].TitleTextDirection =
FarPoint.Win.Chart.TextDirection.Rotate270Degree;
plotArea.Series.Add(series1);
//Sets chart title
FarPoint.Win.Chart.LabelArea labelArea = new FarPoint.Win.Chart.LabelArea();
labelArea.Location = new System.Drawing.PointF(0.5f, 0.02f);
labelArea.AlignmentX = 0.5f;
labelArea.AlignmentY = 0.0f;
labelArea.Text = "The Chart";
```

Visual Basic

```
Dim plotArea As New FarPoint.Win.Chart.YPlotArea()
```

```

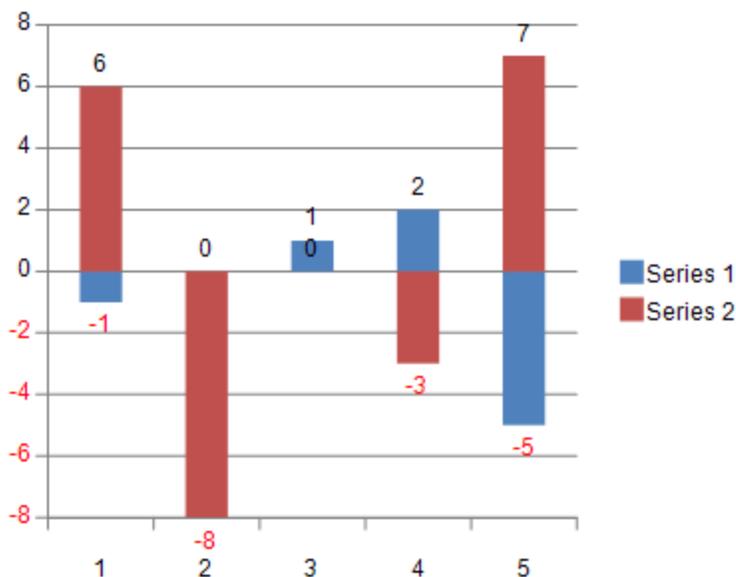
plotArea.Location = New System.Drawing.PointF(0.2F, 0.2F)
plotArea.Size = New System.Drawing.SizeF(0.6F, 0.6F)
'Sets the Axis label
plotArea.XAxis.Title = "Categories"
plotArea.YAxes(0).Title = "Values"
plotArea.YAxes(0).TitleTextDirection = FarPoint.Win.Chart.TextDirection.Rotate270Degree
plotArea.Series.Add(series1)
'Sets chart title
Dim labelArea As New FarPoint.Win.Chart.LabelArea()
labelArea.Location = New System.Drawing.PointF(0.5F, 0.02F)
labelArea.AlignmentX = 0.5F
labelArea.AlignmentY = 0F
labelArea.Text = "The Chart"

```

Display Negative Numbers in Red

The negative data label values and value axis labels of a chart can be displayed in red color.

The **Series.LabelNegativeRed** ('LabelNegativeRed Property' in the on-line documentation) and **ValueAxis.LabelNegativeRed** ('LabelNegativeRed Property' in the on-line documentation) properties can be set to True to display the negative values in red.



The properties show the following behavior:

- When the data label contains parts such as Series Name, along with negative values, the red font color will be applied to them as well.
- When the data label does not show value, the property loses its effect.
- The X Index Axis does not allow negative numbers.
- This feature is not applicable to whisker and waterfall charts.

The following code shows how to apply **LabelNegativeRed** property to display negative label values in red.

C#

```

var worksheet = fpSpread1_Sheet1.AsWorksheet();

worksheet.Cells[0, 0].Value = -1;
worksheet.Cells[1, 0].Value = 0;

```

```

worksheet.Cells[2, 0].Value = 1;
worksheet.Cells[3, 0].Value = 2;
worksheet.Cells[4, 0].Value = -5;
worksheet.Cells[0, 1].Value = 6;
worksheet.Cells[1, 1].Value = -8;
worksheet.Cells[2, 1].Value = 0;
worksheet.Cells[3, 1].Value = -3;
worksheet.Cells[4, 1].Value = 7;

FarPoint.Win.Spread.Model.CellRange range = new FarPoint.Win.Spread.Model.CellRange(0,
0, 5, 2);

FarPoint.Win.Spread.Chart.SpreadChart chart = fpSpread1.Sheets[0].AddChart(range,
typeof(FarPoint.Win.Chart.BarSeries), 400, 300, 200, 50);

FarPoint.Win.Chart.BarSeries series =
(FarPoint.Win.Chart.BarSeries)chart.Model.PlotAreas[0].Series[0];
FarPoint.Win.Chart.BarSeries series1 =
(FarPoint.Win.Chart.BarSeries)chart.Model.PlotAreas[0].Series[1];

// Setting label visible for data label in series 0
series.LabelVisible = true;
series.LabelNegativeRed = true; // Setting LabelNegativeRed as true

// Setting label visible for data label in series 1
series1.LabelVisible = true;
series1.LabelNegativeRed = true;

FarPoint.Win.Chart.YPlotArea plotArea =
(FarPoint.Win.Chart.YPlotArea)chart.Model.PlotAreas[0];

// Setting label visible for value axis
plotArea.YAxes[0].LabelVisible = true;
plotArea.YAxes[0].LabelNegativeRed = true; // Setting LabelNegativeRed as true

```

Visual Basic

```

Dim worksheet = FpSpread1_Sheet1.AsWorksheet()

worksheet.Cells(0, 0).Value = -1
worksheet.Cells(1, 0).Value = 0
worksheet.Cells(2, 0).Value = 1
worksheet.Cells(3, 0).Value = 2
worksheet.Cells(4, 0).Value = -5
worksheet.Cells(0, 1).Value = 6
worksheet.Cells(1, 1).Value = -8
worksheet.Cells(2, 1).Value = 0
worksheet.Cells(3, 1).Value = -3
worksheet.Cells(4, 1).Value = 7

Dim range As FarPoint.Win.Spread.Model.CellRange = New
FarPoint.Win.Spread.Model.CellRange(0, 0, 5, 2)

Dim chart As FarPoint.Win.Spread.Chart.SpreadChart =
FpSpread1.Sheets(0).AddChart(range, GetType(FarPoint.Win.Chart.BarSeries), 400, 300,
200, 50)

```

```

Dim series As FarPoint.Win.Chart.BarSeries = CType(chart.Model.PlotAreas(0).Series(0),
FarPoint.Win.Chart.BarSeries)
Dim series1 As FarPoint.Win.Chart.BarSeries = CType(chart.Model.PlotAreas(0).Series(1),
FarPoint.Win.Chart.BarSeries)

'Setting label visible for data label in series 0
series.LabelVisible = True
series.LabelNegativeRed = True 'Setting LabelNegativeRed as true

'Setting label visible for data label in series 1
series1.LabelVisible = True
series1.LabelNegativeRed = True 'Setting LabelNegativeRed as true

Dim plotArea As FarPoint.Win.Chart.YPlotArea = CType(chart.Model.PlotAreas(0),
FarPoint.Win.Chart.YPlotArea)

'Setting label visible for value axis
plotArea.YAxes(0).LabelVisible = True
plotArea.YAxes(0).LabelNegativeRed = True 'Setting LabelNegativeRed as true

```

ExcelIO supports negative red numbers by using the general formatter property. For example, you can use the following format code "#,0;[Red]-#,0" in Spread before exporting to Excel, where the positive and negative formats are separated by a semi-colon and the negative values are set to display in red.

The following code shows how to export a chart and display negative label values in red.

C#

```

// To export negative values in red
FarPoint.Win.Spread.Model.GeneralFormatter formatter = new
FarPoint.Win.Spread.Model.GeneralFormatter();
formatter.SetFormatString("#,0;[Red]-#,0", false);

series.LabelFormatter = formatter;

fpSpread1.SaveExcel("label-chart.xlsx", FarPoint.Excel.ExcelSaveFlags.UseOOXMLFormat |
FarPoint.Excel.ExcelSaveFlags.Exchangeable);

```

Visual Basic

```

'To export negative values in red
Dim formatter As FarPoint.Win.Spread.Model.GeneralFormatter = New
FarPoint.Win.Spread.Model.GeneralFormatter()
formatter.SetFormatString("#,0;[Red]-#,0", False)

series.LabelFormatter = formatter

FpSpread1.SaveExcel("label-chart.xlsx", FarPoint.Excel.ExcelSaveFlags.UseOOXMLFormat Or
FarPoint.Excel.ExcelSaveFlags.Exchangeable)

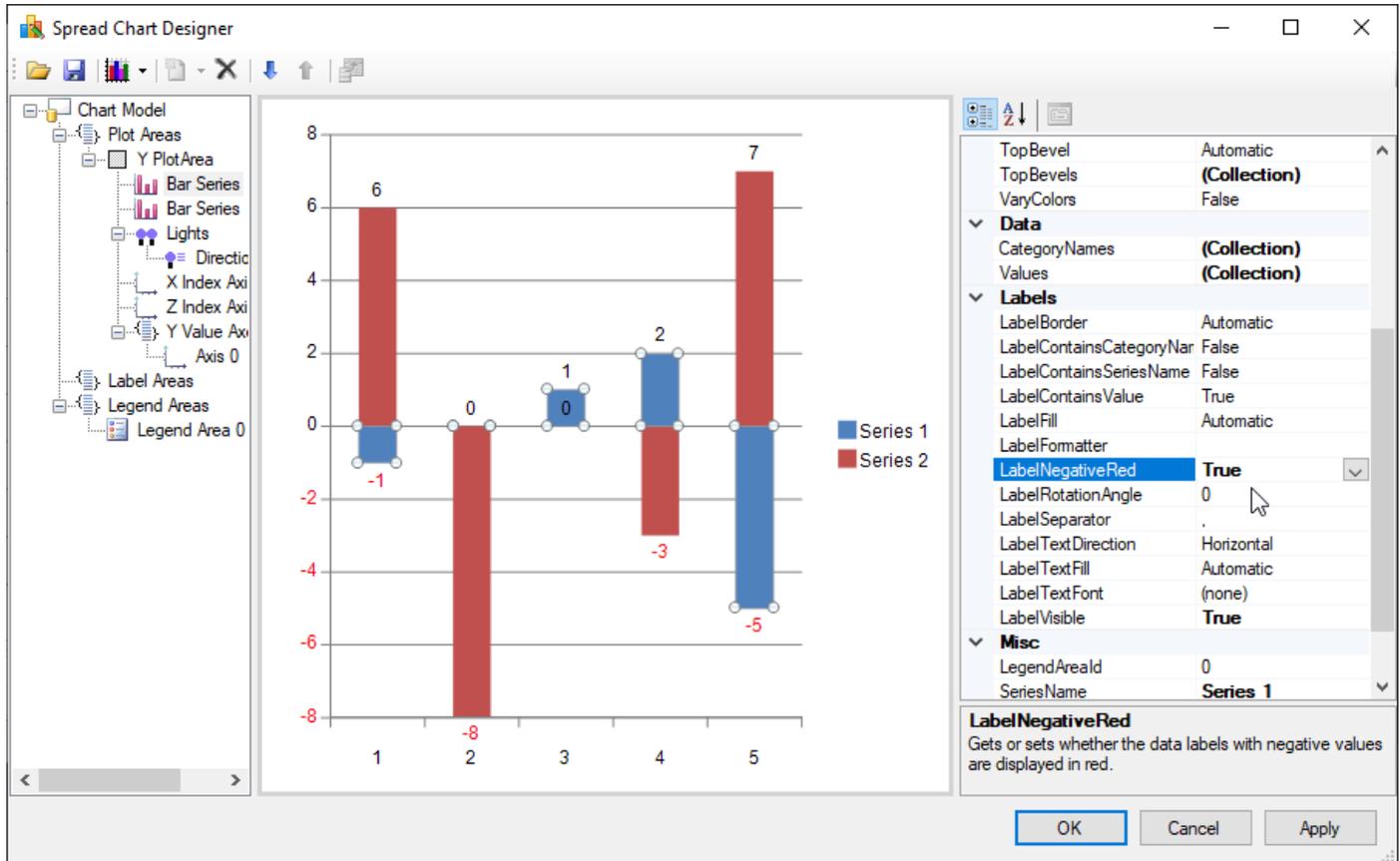
```

Using the Chart Designer

1. Run the **Chart Designer**.
2. Expand the target **Plot Area** from the tree menu on the left.
3. Select **X Index Axis** and set the required properties in the property list on the right.
4. Expand **Y Value Axis**, select **Axis0**, and set the required properties in the **Title** section of the property list on the right.

5. Click OK and exit **Chart Designer**.

You can also enable showing negative labels in red in the Chart Designer by selecting the Bar Series target from the tree menu on the left and enable the **LabelNegativeRed** option to True on the property list.



For information on starting **Chart Designer** ('**Spread Designer Guide**' in the on-line documentation), refer to Chart Designer in the **SPREAD Designer Guide** ('**Using the Chart Designer**' in the on-line documentation).

Legends

The legend contains identifiers for each of the series of the data. The legend area can contain legend items, a background, and borders. The legend area is positioned using a relative location (where (0,0) = the left upper corner of the chart and (1,1) = the right lower corner of the chart) and a relative alignment (where (0,0) = the left upper corner of the label area and (1,1) = the right lower corner of the label area).

See the following for more information on how to set properties for the legend:

- **Legend** ('**LegendArea Class**' in the on-line documentation)
- **LegendAreaCollection** ('**LegendAreaCollection Class**' in the on-line documentation)

Example

The following example sets properties for the legend.

C#

```
FarPoint.Win.Chart.LegendArea legend = new FarPoint.Win.Chart.LegendArea();
legend.Location = new PointF(0.98f, 0.5f);
legend.AlignmentX = 1.0f;
legend.AlignmentY = 0.5f;
```

Visual Basic

```
Dim legend As New FarPoint.Win.Chart.LegendArea()
legend.Location = New PointF(0.98F, 0.5F)
legend.AlignmentX = 1.0F
legend.AlignmentY = 0.5F
```

Using the Chart Designer

1. Run the **Chart Designer**.
2. Select the target **Legend Area** from the tree menu on the left.
3. Set the required properties in the property list on the right.
4. Click OK and exit **Chart Designer**.



For information on starting **Chart Designer** ('**Spread Designer Guide**' in the on-line documentation), refer to Chart Designer in the **SPREAD Designer Guide** ('**Using the Chart Designer**' in the on-line documentation).

Creating Charts

You can add charts using code, the Spread designer, or the Chart designer. You can also bind charts and let the end user make changes to the chart at run time.

For more information, see the following topics:

- **Using the Chart Control on sheet**
 - **Allowing the User to Change the Chart**
 - **Saving or Loading a Chart**
- **Using the Chart Control**
 - **Save/Load Chart Control**
- **Creating Plot Types**
 - **Creating a Y Plot**
 - **Creating an XY Plot**
 - **Creating an XYZ Plot**
 - **Creating a Pie Plot**
 - **Creating a Polar Plot**
 - **Creating a Radar Plot**
 - **Creating a Sunburst Chart**
 - **Creating a Treemap Chart**
 - **Combining different types of plots**
- **Connecting to Data**
 - **Using a Bound Data Source**
 - **Using an UnBound Data Source**
 - **Binding with cell range**
- **Advanced chart settings**
 - **Fill Effects**

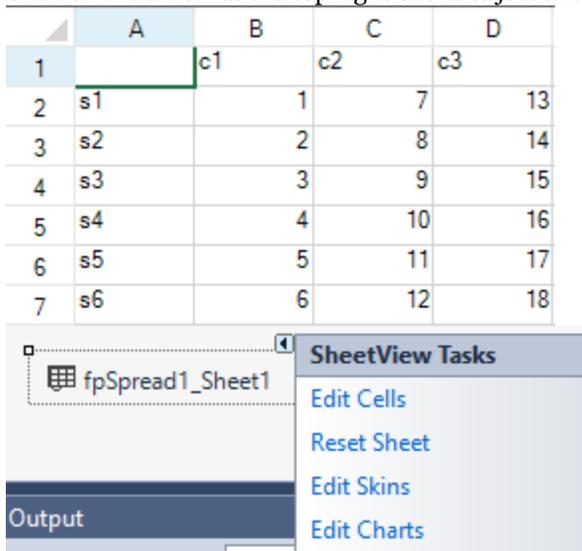
- View Type

Using the Chart Control on sheet

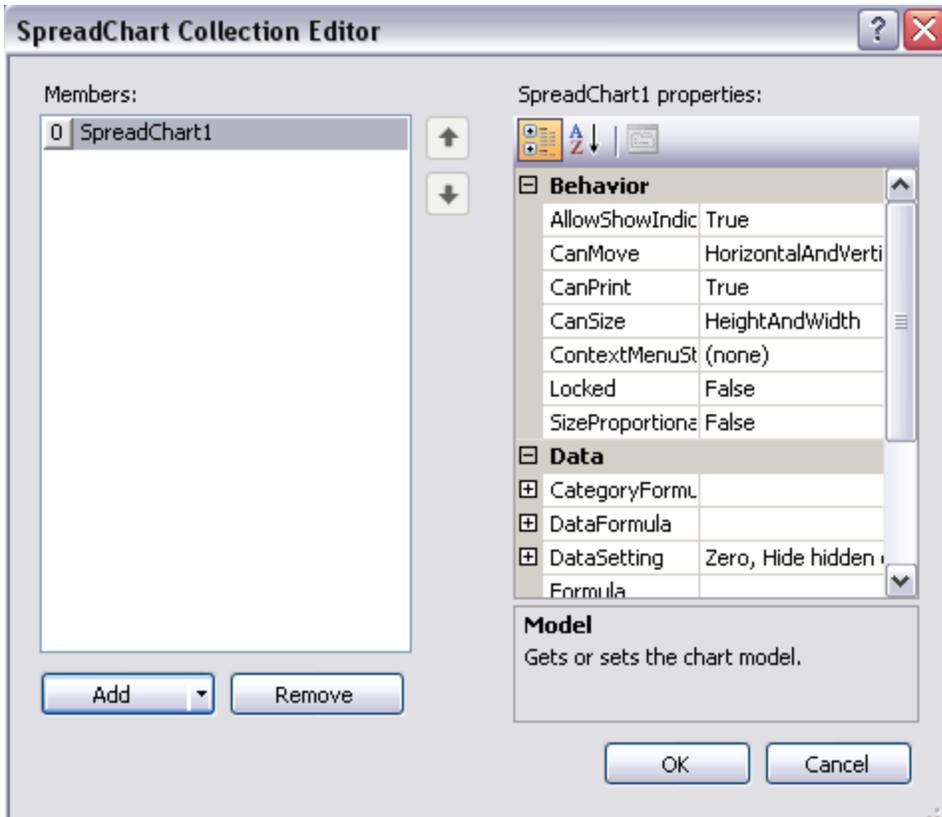
You can add a chart control to the sheet using code, Spread designer, and Visual Studio Design view. You can also allow the user to resize the chart and the range of data used in the chart control. For more details on adding chart control using Spread designer, refer to **Adding a Chart Control ('Adding Charts' in the on-line documentation)**.

Using the Design View

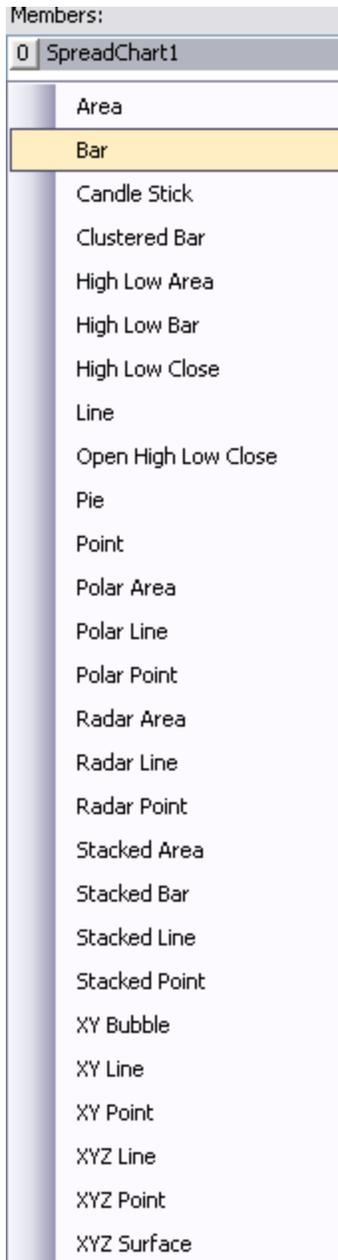
1. Click on the fpSpread1_Sheet1 object at the bottom of the page.
2. Click on the arrow at the top right of the object and select **Edit Charts**.



3. This brings up the **SpreadChart Collection Editor**. Click **Add** to add a chart and then set the chart properties.



4. The **Add** button has a drop-down menu with chart type options (bar, for example).



5. Select **Model** in the **SpreadChart Collection Editor** to bring up the chart designer or choose the **PlotAreas Collection** under Model. Use the **Add** button drop-down menu to select a plot area type (YPlotArea, for example).
6. Use the **Series Collection** to add a series type (BarSeries, for example). The **Add** button has a drop-down for the types of series you can add.
7. Use the **Values Collection** to bring up the **Double Collection Editor** that can be used to add data to the chart.
8. Select **OK** for each dialog.

Using Code

You can add a chart control to the Spread control using code. This example creates data in cells and then adds the chart control.

Example**C#**

```

fpSpread1.Sheets[0].Cells[0, 1].Value = "c1";
fpSpread1.Sheets[0].Cells[0, 2].Value = "c2";
fpSpread1.Sheets[0].Cells[0, 3].Value = "c3";
fpSpread1.Sheets[0].Cells[1, 0].Value = "s1";
fpSpread1.Sheets[0].Cells[2, 0].Value = "s2";
fpSpread1.Sheets[0].Cells[3, 0].Value = "s3";
fpSpread1.Sheets[0].Cells[4, 0].Value = "s4";
fpSpread1.Sheets[0].Cells[5, 0].Value = "s5";
fpSpread1.Sheets[0].Cells[6, 0].Value = "s6";
fpSpread1.Sheets[0].Cells[1, 1].Value = 1;
fpSpread1.Sheets[0].Cells[2, 1].Value = 2;
fpSpread1.Sheets[0].Cells[3, 1].Value = 3;
fpSpread1.Sheets[0].Cells[4, 1].Value = 4;
fpSpread1.Sheets[0].Cells[5, 1].Value = 5;
fpSpread1.Sheets[0].Cells[6, 1].Value = 6;
fpSpread1.Sheets[0].Cells[1, 2].Value = 7;
fpSpread1.Sheets[0].Cells[2, 2].Value = 8;
fpSpread1.Sheets[0].Cells[3, 2].Value = 9;
fpSpread1.Sheets[0].Cells[4, 2].Value = 10;
fpSpread1.Sheets[0].Cells[5, 2].Value = 11;
fpSpread1.Sheets[0].Cells[6, 2].Value = 12;
fpSpread1.Sheets[0].Cells[1, 3].Value = 13;
fpSpread1.Sheets[0].Cells[2, 3].Value = 14;
fpSpread1.Sheets[0].Cells[3, 3].Value = 15;
fpSpread1.Sheets[0].Cells[4, 3].Value = 16;
fpSpread1.Sheets[0].Cells[5, 3].Value = 17;
fpSpread1.Sheets[0].Cells[6, 3].Value = 18;
FarPoint.Win.Spread.Model.CellRange range = new
FarPoint.Win.Spread.Model.CellRange(0, 0, 7, 4);
fpSpread1.Sheets[0].AddChart(range, typeof(FarPoint.Win.Chart.BarSeries), 400, 300,
0, 0, FarPoint.Win.Chart.ChartViewType.View3D, false);

```

Visual Basic

```

FpSpread1.Sheets(0).Cells(0, 1).Value = "c1"
FpSpread1.Sheets(0).Cells(0, 2).Value = "c2"
FpSpread1.Sheets(0).Cells(0, 3).Value = "c3"
FpSpread1.Sheets(0).Cells(1, 0).Value = "s1"
FpSpread1.Sheets(0).Cells(2, 0).Value = "s2"
FpSpread1.Sheets(0).Cells(3, 0).Value = "s3"
FpSpread1.Sheets(0).Cells(4, 0).Value = "s4"
FpSpread1.Sheets(0).Cells(5, 0).Value = "s5"
FpSpread1.Sheets(0).Cells(6, 0).Value = "s6"
FpSpread1.Sheets(0).Cells(1, 1).Value = 1
FpSpread1.Sheets(0).Cells(2, 1).Value = 2
FpSpread1.Sheets(0).Cells(3, 1).Value = 3
FpSpread1.Sheets(0).Cells(4, 1).Value = 4
FpSpread1.Sheets(0).Cells(5, 1).Value = 5
FpSpread1.Sheets(0).Cells(6, 1).Value = 6
FpSpread1.Sheets(0).Cells(1, 2).Value = 7
FpSpread1.Sheets(0).Cells(2, 2).Value = 8
FpSpread1.Sheets(0).Cells(3, 2).Value = 9

```

```

FpSpread1.Sheets(0).Cells(4, 2).Value = 10
FpSpread1.Sheets(0).Cells(5, 2).Value = 11
FpSpread1.Sheets(0).Cells(6, 2).Value = 12
FpSpread1.Sheets(0).Cells(1, 3).Value = 13
FpSpread1.Sheets(0).Cells(2, 3).Value = 14
FpSpread1.Sheets(0).Cells(3, 3).Value = 15
FpSpread1.Sheets(0).Cells(4, 3).Value = 16
FpSpread1.Sheets(0).Cells(5, 3).Value = 17
FpSpread1.Sheets(0).Cells(6, 3).Value = 18
Dim range As New FarPoint.Win.Spread.Model.CellRange(0, 0, 7, 4)
FpSpread1.Sheets(0).AddChart(range, GetType(FarPoint.Win.Chart.BarSeries), 400, 300,
0, 0, FarPoint.Win.Chart.ChartViewType.View3D, False)

```

You can add a chart control to the Spread control using code. This example creates a chart control, adds data to the chart control, and then adds the chart control to Spread.

C#

```

FarPoint.Win.Chart.BarSeries series = new FarPoint.Win.Chart.BarSeries();
series.Add(2.0);
series.Add(4.0);
series.Add(3.0);
series.Add(5.0);
FarPoint.Win.Chart.YPlotArea plotArea = new FarPoint.Win.Chart.YPlotArea();
plotArea.Location = new PointF(0.2f, 0.2f);
plotArea.Size = new SizeF(0.6f, 0.6f);
plotArea.Series.Add(series);
FarPoint.Win.Chart.LabelArea label = new FarPoint.Win.Chart.LabelArea();
label.Text = "Bar Chart";
label.Location = new PointF(0.5f, 0.02f);
label.AlignmentX = 0.5f;
label.AlignmentY = 0.0f;
FarPoint.Win.Chart.LegendArea legend = new FarPoint.Win.Chart.LegendArea();
legend.Location = new PointF(0.98f, 0.5f);
legend.AlignmentX = 1.0f;
legend.AlignmentY = 0.5f;
FarPoint.Win.Chart.ChartModel model = new FarPoint.Win.Chart.ChartModel();
model.LabelAreas.Add(label);
model.LegendAreas.Add(legend);
model.PlotAreas.Add(plotArea);
FarPoint.Win.Spread.Chart.SpreadChart chart = new
FarPoint.Win.Spread.Chart.SpreadChart();
chart.Size = new Size(200, 200);
chart.Location = new Point(100, 100);
chart.Model = model;
fpSpread1.Sheets[0].Charts.Add(chart);

```

Visual Basic

```

Dim series As New FarPoint.Win.Chart.BarSeries()
series.Add(2.0)
series.Add(4.0)
series.Add(3.0)
series.Add(5.0)
Dim plotArea As New FarPoint.Win.Chart.YPlotArea()
plotArea.Location = New PointF(0.2F, 0.2F)
plotArea.Size = New SizeF(0.6F, 0.6F)

```

```

plotArea.Series.Add(series)
Dim label As New FarPoint.Win.Chart.LabelArea()
label.Text = "Bar Chart"
label.Location = New PointF(0.5F, 0.02F)
label.AlignmentX = 0.5F
label.AlignmentY = 0.0F
Dim legend As New FarPoint.Win.Chart.LegendArea()
legend.Location = New PointF(0.98F, 0.5F)
legend.AlignmentX = 1.0F
legend.AlignmentY = 0.5F
Dim model As New FarPoint.Win.Chart.ChartModel()
model.LabelAreas.Add(label)
model.LegendAreas.Add(legend)
model.PlotAreas.Add(plotArea)
Dim chart As New FarPoint.Win.Spread.Chart.SpreadChart()
chart.Size = New Size(200, 200)
chart.Location = New Point(100, 100)
chart.Model = model
FpSpread1.Sheets(0).Charts.Add(chart)

```

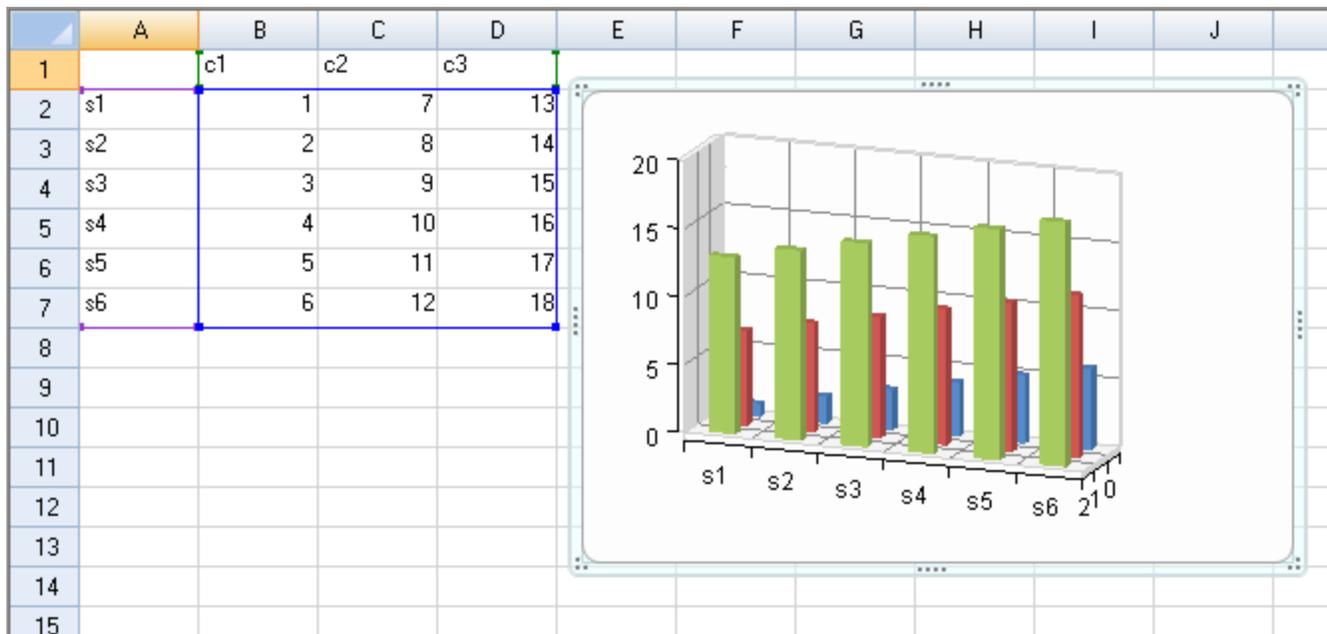
Allowing the User to Change the Chart

You can allow the users to resize, move, or change the range of elements displayed by the chart. The user can also select elements on the chart and the related cell range in the Spread control will be selected.

The user can make the following changes at run time.

- The user can select the chart and then move or resize the chart.
- The user can select the chart and then the range of data used by the chart.
- The user can resize the block of selected data to change the range of data in the chart.
- The user can edit the cells used by the chart to change the values.

The following image shows a selected range of data used by the chart. Put the mouse pointer over the blue square to get resize arrows.



You can use the **CanMove** ('**CanMove Property**' in the on-line documentation) property to prevent the user from moving and **CanSize** ('**CanSize Property**' in the on-line documentation) property to resize the chart. The **Locked** property prevents the user from moving and resizing the chart control.

Using Code

This example sets the **Locked**, **CanMove**, and **CanResize** properties.

Example

C#

```
FarPoint.Win.Spread.Chart.SpreadChart chart;
chart = fpSpread1.Sheets[0].AddChart(0, 0, typeof(FarPoint.Win.Chart.BarSeries), 400,
400, 200, 80, FarPoint.Win.Chart.ChartViewType.View2D, true);
chart.Locked = true;
//chart.CanSize = FarPoint.Win.Spread.DrawingSpace.Sizing.None;
//chart.CanMove = FarPoint.Win.Spread.DrawingSpace.Moving.Horizontal;
```

VB

```
Dim chart As FarPoint.Win.Spread.Chart.SpreadChart
Dim range As New FarPoint.Win.Spread.Model.CellRange(0, 0, 7, 4)
chart = fpSpread1.Sheets(0).AddChart(range, GetType(FarPoint.Win.Chart.BarSeries), 400,
300, 300, 80, FarPoint.Win.Chart.ChartViewType.View3D, False)
chart.Locked = True
'chart.CanSize = FarPoint.Win.Spread.DrawingSpace.Sizing.None
'chart.CanMove = FarPoint.Win.Spread.DrawingSpace.Moving.Horizontal
```

Saving or Loading a Chart

You can read or write to a file or stream using the **IXmlSerializable** interface.

Example

The following code writes to a file.

C#

```
FarPoint.Win.Chart.ChartModel model = chart.Model;
System.Xml.XmlTextWriter writer = new System.Xml.XmlTextWriter(@"c:\temp\test.xml",
null);
model.WriteXml(writer);
writer.Close();
```

Visual Basic

```
Dim model As FarPoint.Win.Chart.ChartModel = chart.Model
Dim writer As New System.Xml.XmlTextWriter("c:\temp\test.xml", Nothing)
model.WriteXml(writer)
writer.Close()
```

The following code reads from a file.

C#

```
FarPoint.Win.Chart.ChartModel model = chart.Model;
System.Xml.XmlTextReader reader = new System.Xml.XmlTextReader(@"c:\temp\test.xml");
model.ReadXml(reader);
reader.Close();
```

Visual Basic

```
Dim model As FarPoint.Win.Chart.ChartModel = Chart.Model
Dim reader As New System.Xml.XmlTextReader("c:\temp\test.xml")
model.ReadXml(reader)
reader.Close()
```

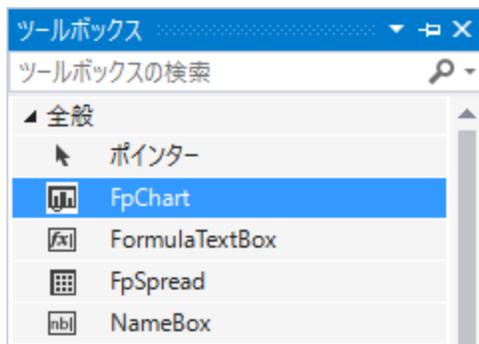
SPREAD デザイナの使用

1. Run **Chart Designer**.
2. Select the **Open Chart Model** or **Save Chart Model** icon at the left end of the icons lined up at the top.

 For information on starting **Chart Designer ('Spread Designer Guide' in the on-line documentation)**, refer to Chart Designer in the **SPREAD Designer Guide ('Using the Chart Designer' in the on-line documentation)**.

Using the Chart Control

The Chart (**FpChart ('FpChart Class' in the on-line documentation)**) control appears in the Visual Studio toolbox as the icon in the following image.

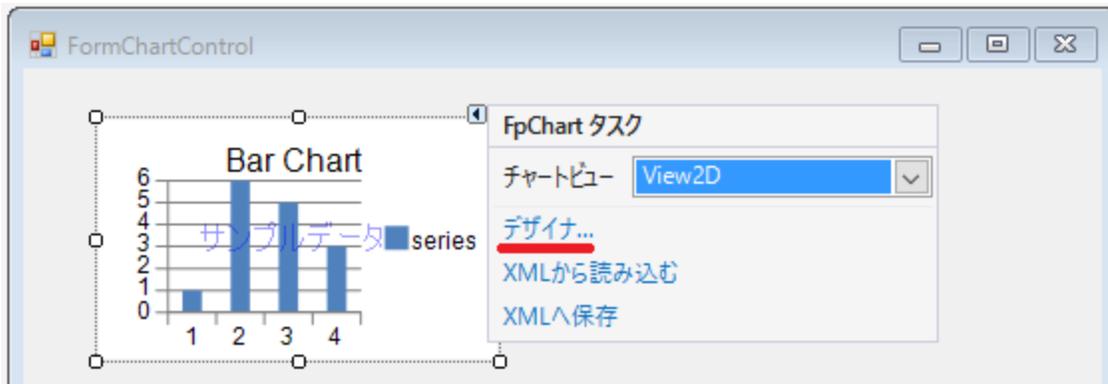


You can add chart controls to your form by dragging the FpChart control from the toolbox. Chart controls added to the form can be set using the chart designer or in code as follows:

 Chart controls can be saved and loaded into files or streams. See "**Save/Load Chart Control**" for more information.

Using the Chart Designer

1. Drag the FpChart control from the toolbox onto the form.
2. You can launch the Chart Designer by selecting **Designer** from the smart tags of the FpChart control.



 For information on creating graphs using the Chart Designer, see **Chart Designer ('Using the Chart Designer' in the on-line documentation)**.

You can also use code to set the FpChart control without using the chart designer. The code used is similar to the code used to add a chart on a SPREAD sheet.

Using the code

1. Create a series object and add data.
2. Create plot area using the **YPlotArea ('YPlotArea Class' in the on-line documentation)** class.
3. Set the position and size of the plot area.
4. Add the series object created in step 1 to the plot area.
5. Create a label and legend area to display the title.
6. Create a chart model and add plot areas, labels, and legend areas.
7. Set the **Model ('Model Property' in the on-line documentation)** property of the **FpChart ('FpChart Class' in the on-line documentation)** control to the chart model created in step 6.

Example

The following example demonstrate how to use the FpChart control to create a bar chart.

C#

```
FarPoint.Win.Chart.BarSeries series = new FarPoint.Win.Chart.BarSeries();
series.SeriesName = "Series 0";
series.Values.Add(2.0);
series.Values.Add(4.0);
series.Values.Add(3.0);
series.Values.Add(5.0);
FarPoint.Win.Chart.YPlotArea plotArea = new FarPoint.Win.Chart.YPlotArea();
plotArea.Location = new System.Drawing.PointF(0.2f, 0.2f);
plotArea.Size = new System.Drawing.SizeF(0.6f, 0.6f);
plotArea.Series.Add(series);
FarPoint.Win.Chart.LabelArea label = new FarPoint.Win.Chart.LabelArea();
label.Text = "Bar Chart";
label.Location = new System.Drawing.PointF(0.5f, 0.02f);
label.AlignmentX = 0.5f;
label.AlignmentY = 0.0f;
FarPoint.Win.Chart.LegendArea legend = new FarPoint.Win.Chart.LegendArea();
legend.Location = new System.Drawing.PointF(0.98f, 0.5f);
legend.AlignmentX = 1.0f;
legend.AlignmentY = 0.5f;
```

```
FarPoint.Win.Chart.ChartModel model = new FarPoint.Win.Chart.ChartModel();
model.LabelAreas.Add(label);
model.LegendAreas.Add(legend);
model.PlotAreas.Add(plotArea);
fpChart1.Model = model;
```

Visual Basic

```
Dim series As New FarPoint.Win.Chart.BarSeries()
series.SeriesName = "Series 0"
series.Values.Add(2.0)
series.Values.Add(4.0)
series.Values.Add(3.0)
series.Values.Add(5.0)
Dim plotArea As New FarPoint.Win.Chart.YPlotArea()
plotArea.Location = New System.Drawing.PointF(0.2F, 0.2F)
plotArea.Size = New System.Drawing.SizeF(0.6F, 0.6F)
plotArea.Series.Add(series)
Dim label As New FarPoint.Win.Chart.LabelArea()
label.Text = "Bar Chart"
label.Location = New System.Drawing.PointF(0.5F, 0.02F)
label.AlignmentX = 0.5F
label.AlignmentY = 0.0F
Dim legend As New FarPoint.Win.Chart.LegendArea()
legend.Location = New System.Drawing.PointF(0.98F, 0.5F)
legend.AlignmentX = 1.0F
legend.AlignmentY = 0.5F
Dim model As New FarPoint.Win.Chart.ChartModel()
model.LabelAreas.Add(label)
model.LegendAreas.Add(legend)
model.PlotAreas.Add(plotArea)
FpChart1.Model = model
```

Save/Load Chart Control

Chart controls can be saved to and loaded from an XML file. This feature is also available from the smart tag of the FpChart control placed on the form.

Using Code

You can load the XML file with the **LoadFromTemplate** ('**LoadFromTemplate Method**' in the **on-line documentation**) method of the **FpChart** ('**FpChart Class**' in the **on-line documentation**) class and save it to the XML file with the **SaveToTemplate** ('**SaveToTemplate Method**' in the **on-line documentation**) method.

Example

The following sample code saves or loads the chart control into an XML file.

C#

```
string f;
f = "c:\\temp\\chart.xml";
System.IO.FileStream s = new System.IO.FileStream(f, System.IO.FileMode.OpenOrCreate,
System.IO.FileAccess.ReadWrite);
fpChart1.SaveToTemplate(s);
```

```
//fpChart1.LoadFromTemplate(s);
```

Visual Basic

```
Dim f As String
f = "c:\chart.xml"
Dim s As New System.IO.FileStream(f, System.IO.FileMode.OpenOrCreate,
System.IO.FileAccess.ReadWrite)
FpChart1.SaveToTemplate(s)
'FpChart1.LoadFromTemplate(s)
```

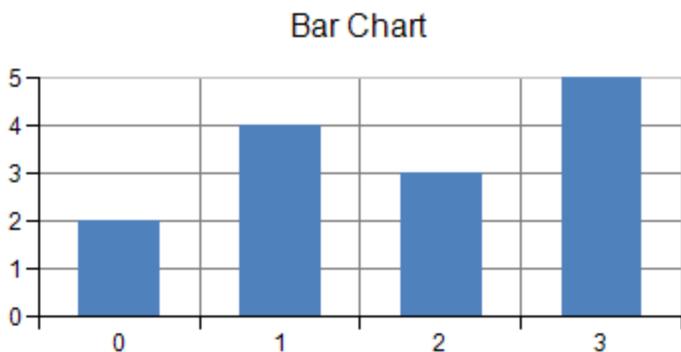
Creating Plot Types

The following topics explain how to create different plot types:

- **Creating a Y Plot**
- **Creating an XY Plot**
- **Creating an XYZ Plot**
- **Creating a Pie Plot**
- **Creating a Polar Plot**
- **Creating a Radar Plot**
- **Creating a Sunburst Chart**
- **Creating a Treemap Chart**
- **Combining different types of plots**

Creating a Y Plot

You can create a Y Plot chart using code or the designer. The following image shows a Y Plot bar type chart.



For details on the API, see the **YPlotArea** (**'YPlotArea Class' in the on-line documentation**) class.

The following classes are also available when creating Y plot type charts:

- Area chart
AreaSeries (**'AreaSeries Class' in the on-line documentation**)
- Bar chart
BarSeries (**'BarSeries Class' in the on-line documentation**)
- Box Whisker
BoxWhiskerSeries (**'BoxWhiskerSeries Class' in the on-line documentation**)
- Funnel chart
FunnelSeries (**'FunnelSeries Class' in the on-line documentation**)

- Histogram chart
HistogramSeries ('HistogramSeries Class' in the on-line documentation)
- Line chart
LineSeries ('LineSeries Class' in the on-line documentation)
- Market Data (High-Low) chart
CandlestickSeries ('CandlestickSeries Class' in the on-line documentation),
HighLowAreaSeries ('HighLowAreaSeries Class' in the on-line documentation),
HighLowBarSeries ('HighLowBarSeries Class' in the on-line documentation),
HighLowCloseSeries ('HighLowCloseSeries Class' in the on-line documentation),
OpenHighLowCloseSeries ('OpenHighLowCloseSeries Class' in the on-line documentation)
- Pareto chart
ParetoSeries ('ParetoSeries Class' in the on-line documentation)
- Point chart
PointSeries ('PointSeries Class' in the on-line documentation)
- Waterfall chart
WaterfallSeries ('WaterfallSeries Class' in the on-line documentation)

Also, you can add aggregate data series (Clustered Bar Chart, Stacked Bar Chart) with multiple series. The following classes refer the series collection from Series property respectively.

- Clustered bar Chart
ClusteredBarSeries ('ClusteredBarSeries Class' in the on-line documentation)
- Stacked bar chart
StackedBarSeries ('StackedBarSeries Class' in the on-line documentation)
- Stacked area chart
StackedAreaSeries ('StackedAreaSeries Class' in the on-line documentation)
- Stacked line chart
StackedLineSeries ('StackedLineSeries Class' in the on-line documentation)
- Stacked point chart (Scatter chart)
StackedPointSeries ('StackedPointSeries Class' in the on-line documentation)

Using Code

1. Create a **BarSeries** ('BarSeries Class' in the on-line documentation) object that represents the data series of bar chart and add the data.
2. Create a **YPlotArea** ('YPlotArea Class' in the on-line documentation) object that represents the plot area and set its position and size.
3. Add a data series to the plot area.
4. Create label and legend area.
5. Create a **ChartModel** ('ChartModel Class' in the on-line documentation) object and add plot area, label and legend area.
6. Assign a chart model to the chart.

Example

The following example creates a bar chart.

C#

```
FarPoint.Win.Chart.BarSeries series = new FarPoint.Win.Chart.BarSeries();
series.SeriesName = "Series 0";
series.Values.Add(2.0);
series.Values.Add(4.0);
series.Values.Add(3.0);
series.Values.Add(5.0);
```

```
FarPoint.Win.Chart.YPlotArea plotArea = new FarPoint.Win.Chart.YPlotArea();
plotArea.Location = new PointF(0.2f, 0.2f);
plotArea.Size = new SizeF(0.6f, 0.6f);
plotArea.Series.Add(series);
FarPoint.Win.Chart.LabelArea label = new FarPoint.Win.Chart.LabelArea();
label.Text = "Bar Chart";
label.Location = new PointF(0.5f, 0.02f);
label.AlignmentX = 0.5f;
label.AlignmentY = 0.0f;
FarPoint.Win.Chart.LegendArea legend = new FarPoint.Win.Chart.LegendArea();
legend.Location = new PointF(0.98f, 0.5f);
legend.AlignmentX = 1.0f;
legend.AlignmentY = 0.5f;
FarPoint.Win.Chart.ChartModel model = new FarPoint.Win.Chart.ChartModel();
model.LabelAreas.Add(label);
model.LegendAreas.Add(legend);
model.PlotAreas.Add(plotArea);
chart2DControl1.Model = model;
```

Visual Basic

```
Dim series As New FarPoint.Win.Chart.BarSeries()
series.SeriesName = "Series 0"
series.Values.Add(2.0)
series.Values.Add(4.0)
series.Values.Add(3.0)
series.Values.Add(5.0)
Dim plotArea As New FarPoint.Win.Chart.YPlotArea()
plotArea.Location = New PointF(0.2F, 0.2F)
plotArea.Size = New SizeF(0.6F, 0.6F)
plotArea.Series.Add(series)
Dim label As New FarPoint.Win.Chart.LabelArea()
label.Text = "Bar Chart"
label.Location = New PointF(0.5F, 0.02F)
label.AlignmentX = 0.5F
label.AlignmentY = 0.0F
Dim legend As New FarPoint.Win.Chart.LegendArea()
legend.Location = New PointF(0.98F, 0.5F)
legend.AlignmentX = 1.0F
legend.AlignmentY = 0.5F
Dim model As New FarPoint.Win.Chart.ChartModel()
model.LabelAreas.Add(label)
model.LegendAreas.Add(legend)
model.PlotAreas.Add(plotArea)
chart2DControl1.Model = model
```

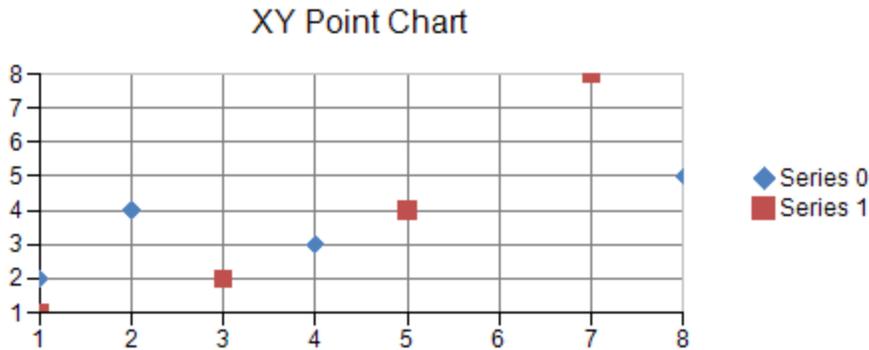
Using Chart designer

1. Run the **Chart Designer**.
 2. Select the target **Chart Model** from the tree menu on the left.
 3. Open the **Plot Area Collection Editor** from Plot Areas in the **Other** section of the property list on the right.
 4. Click the drop-down button to the right of the **Add** button.
 5. Select and add **YPlotArea** and set each property as needed.
-

 For more information regarding opening the [Chart Designer], kindly refer to "[Using the Chart Designer \(on-line documentation\)](#)" in the "[Spread Designer Guide \(on-line documentation\)](#)".

Creating an XY Plot

You can create an XY Plot chart using code or the designer. The following image shows an XY Plot point type chart.



For details on the API, see the **XYPlotArea** ('[XYPlotArea Class](#)' in the on-line documentation) class.

The following classes are also available when creating XY plot type charts:

- Bubble chart
XYBubbleSeries ('[XYBubbleSeries Class](#)' in the on-line documentation)
- Line chart
XYLineSeries ('[XYLineSeries Class](#)' in the on-line documentation)
- Point chart
XYPointSeries ('[XYPointSeries Class](#)' in the on-line documentation)

Using Code

1. Create an **XYPointSeries** ('[XYPointSeries Class](#)' in the on-line documentation) object that represents the scatter plot (marker only) data series and add the data.
2. Create an **XYPlotArea** ('[XYPlotArea Class](#)' in the on-line documentation) object that represents the plot area and set its position and size.
3. Add a data series to the plot area.
4. Create labels and legend areas.
5. Create a **ChartModel** ('[ChartModel Class](#)' in the on-line documentation) object and add plot areas, labels, and legend areas.
6. Assign a chart model to the chart.

Example

The following sample code creates a scatter plot (markers only).

C#

```
FarPoint.Win.Chart.XYPointSeries series0 = new FarPoint.Win.Chart.XYPointSeries();
series0.SeriesName = "Series 0";
series0.XValues.Add(1.0);
series0.XValues.Add(2.0);
series0.XValues.Add(4.0);
series0.XValues.Add(8.0);
series0.YValues.Add(2.0);
```

```
series0.YValues.Add(4.0);
series0.YValues.Add(3.0);
series0.YValues.Add(5.0);
FarPoint.Win.Chart.XYPointSeries series1 = new FarPoint.Win.Chart.XYPointSeries();
series1.SeriesName = "Series 1";
series1.XValues.Add(1.0);
series1.XValues.Add(3.0);
series1.XValues.Add(5.0);
series1.XValues.Add(7.0);
series1.YValues.Add(1.0);
series1.YValues.Add(2.0);
series1.YValues.Add(4.0);
series1.YValues.Add(8.0);
FarPoint.Win.Chart.XYPlotArea plotArea = new FarPoint.Win.Chart.XYPlotArea();
plotArea.Location = new PointF(0.2f, 0.2f);
plotArea.Size = new SizeF(0.6f, 0.6f);
plotArea.Series.Add(series0);
plotArea.Series.Add(series1);
FarPoint.Win.Chart.LabelArea label = new FarPoint.Win.Chart.LabelArea();
label.Text = "XY Point Chart";
label.Location = new PointF(0.5f, 0.02f);
label.AlignmentX = 0.5f;
label.AlignmentY = 0.0f;
FarPoint.Win.Chart.LegendArea legend = new FarPoint.Win.Chart.LegendArea();
legend.Location = new PointF(0.98f, 0.5f);
legend.AlignmentX = 1.0f;
legend.AlignmentY = 0.5f;
FarPoint.Win.Chart.ChartModel model = new FarPoint.Win.Chart.ChartModel();
model.LabelAreas.Add(label);
model.LegendAreas.Add(legend);
model.PlotAreas.Add(plotArea);
chart2DControl1.Model = model;
```

Visual Basic

```
Dim series0 As New FarPoint.Win.Chart.XYPointSeries()
series0.SeriesName = "Series 0"
series0.XValues.Add(1.0)
series0.XValues.Add(2.0)
series0.XValues.Add(4.0)
series0.XValues.Add(8.0)
series0.YValues.Add(2.0)
series0.YValues.Add(4.0)
series0.YValues.Add(3.0)
series0.YValues.Add(5.0)
Dim series1 As New FarPoint.Win.Chart.XYPointSeries()
series1.SeriesName = "Series 1"
series1.XValues.Add(1.0)
series1.XValues.Add(3.0)
series1.XValues.Add(5.0)
series1.XValues.Add(7.0)
series1.YValues.Add(1.0)
series1.YValues.Add(2.0)
series1.YValues.Add(4.0)
series1.YValues.Add(8.0)
Dim plotArea As New FarPoint.Win.Chart.XYPlotArea()
plotArea.Location = New PointF(0.2F, 0.2F)
```

```
plotArea.Size = New SizeF(0.6F, 0.6F)
plotArea.Series.Add(series0)
plotArea.Series.Add(series1)
Dim label As New FarPoint.Win.Chart.LabelArea()
label.Text = "XY Point Chart"
label.Location = New PointF(0.5F, 0.02F)
label.AlignmentX = 0.5F
label.AlignmentY = 0.0F
Dim legend As New FarPoint.Win.Chart.LegendArea()
legend.Location = New PointF(0.98F, 0.5F)
legend.AlignmentX = 1.0F
legend.AlignmentY = 0.5F
Dim model As New FarPoint.Win.Chart.ChartModel()
model.LabelAreas.Add(label)
model.LegendAreas.Add(legend)
model.PlotAreas.Add(plotArea)
Chart2DControl1.Model = model
```

Using the Chart designer

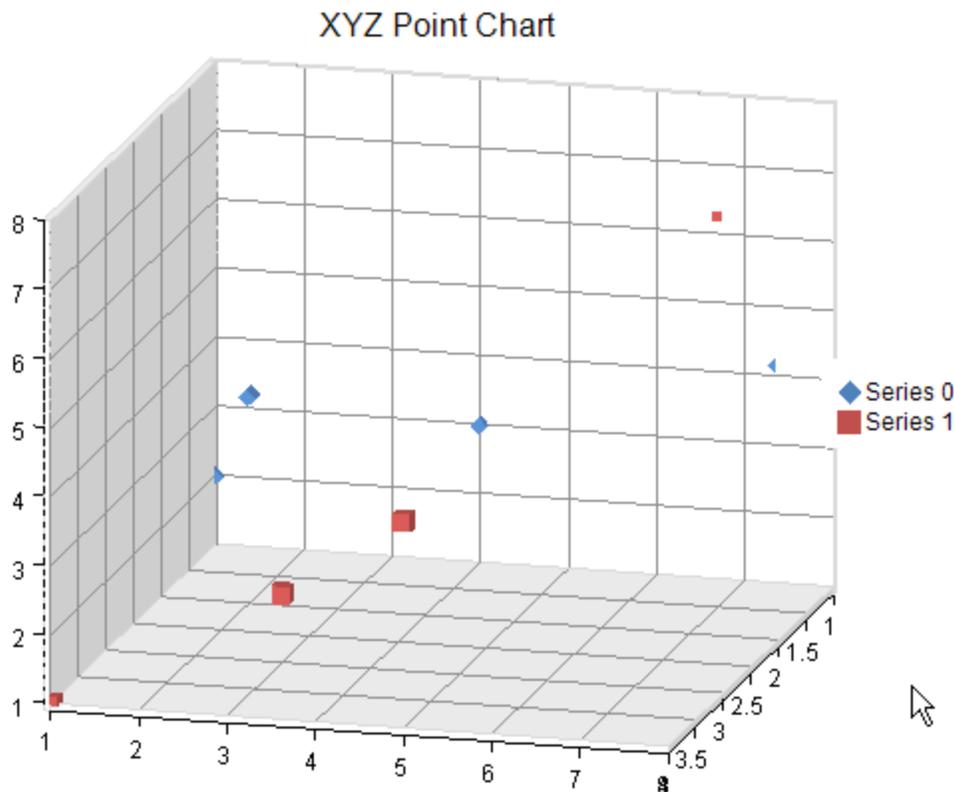
1. Run **Chart Designer**.
2. Select the target **Chart Model** from the tree menu on the left.
3. Open the **Plot Area Collection** Editor from **PlotAreas** in the **Other** section of the property list on the right.
4. Click the drop-down button to the right of the **Add** button.
5. Select and add the **XYPlotArea** and set each property as required.



For information on starting **Chart Designer ('Spread Designer Guide' in the on-line documentation)**, refer to Chart Designer in the **SPREAD Designer Guide ('Using the Chart Designer' in the on-line documentation)**.

Creating an XYZ Plot

You can create an XYZ Plot chart using code or the designer. The following image shows an XYZPlot point type chart.



For details on the API, see the **XYZPlotArea** (**'XYZPlotArea Class' in the on-line documentation**) class.

The following classes are also available when creating XYZ plot type charts:

- Pint Chart
XYZPointSeries (**'XYZPointSeries Class' in the on-line documentation**)
- Line Chart
XYZLineSeries (**'XYZLineSeries Class' in the on-line documentation**)
- XYZ Surface Series
XYZSurfaceSeries (**'XYZSurfaceSeries Class' in the on-line documentation**)

 To display a 3D chart, use the **ViewType** (**'ViewType Property' in the on-line documentation**) property of the **SpreadChart** (**'SpreadChart Class' in the on-line documentation**) class to set the chart display format.

Using Code

1. Create an **XYZPointSeries** (**'XYZPointSeries Class' in the on-line documentation**) object that represents the scatter plot (marker only) data series and add the data.
2. Create an **XYZPlotArea** (**'XYZPlotArea Class' in the on-line documentation**) object that represents the plot area and set its position and size.
3. Add a data series to the plot area.
4. Create labels and legend areas.
5. Create a **ChartModel** (**'ChartModel Class' in the on-line documentation**) object and add plot areas, labels, and legend areas.
6. Assign a chart model to the chart.

Example

The following example demonstrates creating an XYZ scatter plot (markers only).

C#

```
FarPoint.Win.Spread.Chart.SpreadChart chart3DControl1 = new
FarPoint.Win.Spread.Chart.SpreadChart()
{
    Size =new Size(300,300),
    ViewType =FarPoint.Win.Chart.ChartViewType.View3D
};
FarPoint.Win.Chart.XYZPointSeries series0 = new FarPoint.Win.Chart.XYZPointSeries();
series0.SeriesName = "Series 0";
series0.XValues.Add(1.0);
series0.XValues.Add(2.0);
series0.XValues.Add(4.0);
series0.XValues.Add(8.0);
series0.YValues.Add(2.0);
series0.YValues.Add(4.0);
series0.YValues.Add(3.0);
series0.YValues.Add(5.0);
series0.ZValues.Add(1.0);
series0.ZValues.Add(2.0);
series0.ZValues.Add(1.0);
series0.ZValues.Add(2.0);
FarPoint.Win.Chart.XYZPointSeries series1 = new FarPoint.Win.Chart.XYZPointSeries();
series1.SeriesName = "Series 1";
series1.XValues.Add(1.0);
series1.XValues.Add(3.0);
series1.XValues.Add(5.0);
series1.XValues.Add(8.0);
series1.YValues.Add(1.0);
series1.YValues.Add(2.0);
series1.YValues.Add(4.0);
series1.YValues.Add(8.0);
series1.ZValues.Add(4.0);
series1.ZValues.Add(3.0);
series1.ZValues.Add(4.0);
series1.ZValues.Add(3.0);
FarPoint.Win.Chart.XYZPlotArea plotArea = new FarPoint.Win.Chart.XYZPlotArea();
plotArea.Location = new PointF(0.2f, 0.2f);
plotArea.Size = new SizeF(0.6f, 0.6f);
plotArea.Series.Add(series0);
plotArea.Series.Add(series1);
plotArea.Elevation = 10;
plotArea.Rotation = -20;
plotArea.ZAxes.Clear();
plotArea.ZAxes.Add(new FarPoint.Win.Chart.ValueAxis() { AutoMinimum = false,
AutoMaximum = false, Minimum = 0, Maximum = 5 });
FarPoint.Win.Chart.LabelArea label = new FarPoint.Win.Chart.LabelArea();
label.Text = "XYZ Point Chart";
label.Location = new PointF(0.5f, 0.02f);
label.AlignmentX = 0.5f;
label.AlignmentY = 0.0f;
FarPoint.Win.Chart.LegendArea legend = new FarPoint.Win.Chart.LegendArea();
legend.Location = new PointF(0.98f, 0.5f);
legend.AlignmentX = 1.0f;
legend.AlignmentY = 0.5f;
```

```

FarPoint.Win.Chart.ChartModel model = new FarPoint.Win.Chart.ChartModel();
model.LabelAreas.Add(label);
model.LegendAreas.Add(legend);
model.PlotAreas.Add(plotArea);
chart3DControl1.Model = model;

```

Visual Basic

```

Dim chart3DControl1 As New FarPoint.Win.Spread.Chart.SpreadChart() With {
    .Size = New Size(300, 300),
    .ViewType = FarPoint.Win.Chart.ChartViewType.View3D
}
Dim series0 As New FarPoint.Win.Chart.XYZPointSeries()
series0.SeriesName = "Series 0"
series0.XValues.Add(1.0)
series0.XValues.Add(2.0)
series0.XValues.Add(4.0)
series0.XValues.Add(8.0)
series0.YValues.Add(2.0)
series0.YValues.Add(4.0)
series0.YValues.Add(3.0)
series0.YValues.Add(5.0)
series0.ZValues.Add(1.0)
series0.ZValues.Add(2.0)
series0.ZValues.Add(1.0)
series0.ZValues.Add(2.0)
Dim series1 As New FarPoint.Win.Chart.XYZPointSeries()
series1.SeriesName = "Series 1"
series1.XValues.Add(1.0)
series1.XValues.Add(3.0)
series1.XValues.Add(5.0)
series1.XValues.Add(8.0)
series1.YValues.Add(1.0)
series1.YValues.Add(2.0)
series1.YValues.Add(4.0)
series1.YValues.Add(8.0)
series1.ZValues.Add(4.0)
series1.ZValues.Add(3.0)
series1.ZValues.Add(4.0)
series1.ZValues.Add(3.0)
Dim plotArea As New FarPoint.Win.Chart.XYZPlotArea()
plotArea.Location = New PointF(0.2F, 0.2F)
plotArea.Size = New SizeF(0.6F, 0.6F)
plotArea.Series.Add(series0)
plotArea.Series.Add(series1)
plotArea.Elevation = 10
plotArea.Rotation = -20
plotArea.ZAxes.Clear()
plotArea.ZAxes.Add(New FarPoint.Win.Chart.ValueAxis() With {
    .AutoMinimum = False,
    .AutoMaximum = False,
    .Minimum = 0,
    .Maximum = 5
})
Dim label As New FarPoint.Win.Chart.LabelArea()
label.Text = "XYZ Point Chart"
label.Location = New PointF(0.5F, 0.02F)

```

```
label.AlignmentX = 0.5F
label.AlignmentY = 0F
Dim legend As New FarPoint.Win.Chart.LegendArea()
legend.Location = New PointF(0.98F, 0.5F)
legend.AlignmentX = 1.0F
legend.AlignmentY = 0.5F
Dim model As New FarPoint.Win.Chart.ChartModel()
model.LabelAreas.Add(label)
model.LegendAreas.Add(legend)
model.PlotAreas.Add(plotArea)
chart3DControl1.Model = model
```

Using the Chart designer

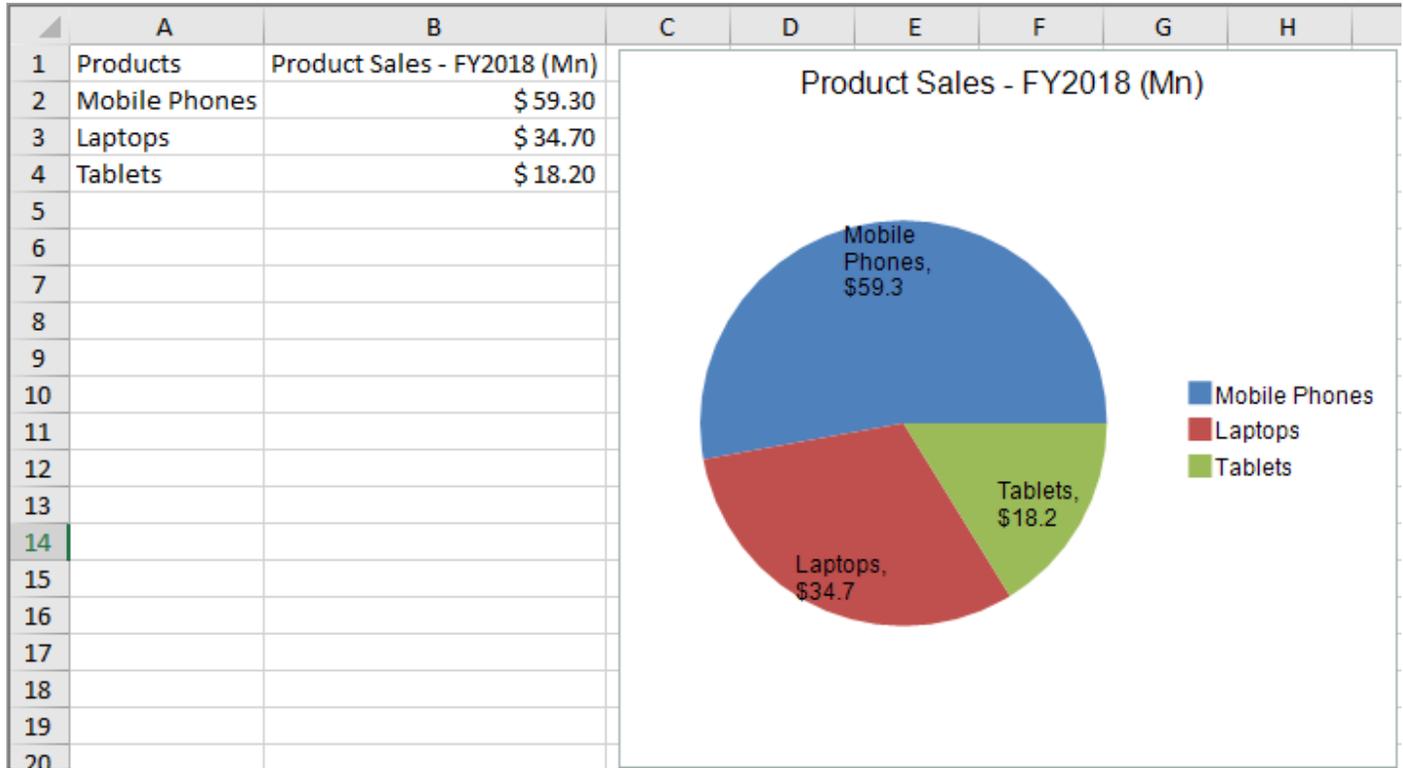
1. Run **Chart Designer**.
2. Select the target **Chart Model** from the tree menu on the left.
3. Open the **Plot Area Collection Editor** from PlotAreas in the **Other** section of the property list on the right.
4. Click the drop-down button to the right of the **Add** button.
5. Select and add the **XYZPlotArea** and set each property as required.



For information on starting **Chart Designer ('Spread Designer Guide' in the on-line documentation)**, refer to Chart Designer in the **SPREAD Designer Guide ('Using the Chart Designer' in the on-line documentation)**.

Creating a Pie Plot

You can create a pie plot chart using code or the designer. The following image shows a Pie Plot type chart for sales of electronic products.



For details on the API, see the **PiePlotArea ('PiePlotArea Class' in the on-line documentation)** class. Also, to create a pie chart, use the **PieSeries ('PieSeries Class' in the on-line documentation)** class that represents the data series of the circle chart.

- To set the pie detachments, set the **PieDetachments ('PieDetachments Property' in the on-line documentation)** property of the PieSeries class.
- To create a donut chart, set the size of the hole in the center of the circle chart with the **HoleSize ('HoleSize Property' in the on-line documentation)** property of the PiePlotArea class.

Using code

1. Create a **PieSeries ('PieSeries Class' in the on-line documentation)** object that represents the data series for the circle chart and add the data.
2. Create a **PiePlotArea ('PiePlotArea Class' in the on-line documentation)** object that represents the plot area and set its position and size.
3. Add a data series to the plot area.
4. Create labels and legend areas.
5. Create a **ChartModel ('ChartModel Class' in the on-line documentation)** object and add plot areas, labels, and legend areas.
6. Assign a chart model to the chart.

Example

The following example creates a pie chart to display the annual sales of electronic products (Mobile Phones, Laptops, Tablets) in a store.

C#

```
// Creating Pie Chart
fpSpread1.Sheets[0].Cells[0, 0].Text = "Products";
fpSpread1.Sheets[0].Cells[1, 0].Text = "Mobile Phones";
fpSpread1.Sheets[0].Cells[2, 0].Text = "Laptops";
fpSpread1.Sheets[0].Cells[3, 0].Text = "Tablets";
fpSpread1.Sheets[0].Cells[0, 1].Text = "Product Sales - FY2018 (Mn)";
fpSpread1.Sheets[0].Cells[1, 1].Value = 59.3;
fpSpread1.Sheets[0].Cells[2, 1].Value = 34.7;
fpSpread1.Sheets[0].Cells[3, 1].Value = 18.2;
CurrencyCellType currencycell = new CurrencyCellType();
currencycell.DecimalPlaces = 2;
currencycell.ShowCurrencySymbol = true;
currencycell.CurrencySymbol = "$";
fpSpread1.Sheets[0].Cells[1, 1, 3, 1].CellType = currencycell;

FarPoint.Win.Spread.Chart.SpreadChart chart = fpSpread1.Sheets[0].AddChart(0, 0,
typeof(FarPoint.Win.Chart.PieSeries), 400, 370, 240, 30);
FarPoint.Win.Chart.PieSeries series =
(FarPoint.Win.Chart.PieSeries)chart.Model.PlotAreas[0].Series[0];
series.LabelVisible = true;
series.LabelContainsCategoryName = true;
series.LabelContainsValue = true;
```

Visual Basic

```
// Creating Pie Chart
fpSpread1.Sheets(0).Cells(0, 0).Text = "Products"
fpSpread1.Sheets(0).Cells(1, 0).Text = "Mobile Phones"
fpSpread1.Sheets(0).Cells(2, 0).Text = "Laptops"
fpSpread1.Sheets(0).Cells(3, 0).Text = "Tablets"
fpSpread1.Sheets(0).Cells(0, 1).Text = "Product Sales - FY2018 (Mn)"
fpSpread1.Sheets(0).Cells(1, 1).Value = 59.3
fpSpread1.Sheets(0).Cells(2, 1).Value = 34.7
fpSpread1.Sheets(0).Cells(3, 1).Value = 18.2
Dim currencycell As CurrencyCellType = New CurrencyCellType()
currencycell.DecimalPlaces = 2
currencycell.ShowCurrencySymbol = True
currencycell.CurrencySymbol = "$"
fpSpread1.Sheets(0).Cells(1, 1, 3, 1).CellType = currencycell

Dim chart As FarPoint.Win.Spread.Chart.SpreadChart = fpSpread1.Sheets(0).AddChart(0, 0,
GetType(FarPoint.Win.Chart.PieSeries), 400, 370, 240, 30)
Dim series As FarPoint.Win.Chart.PieSeries = CType(chart.Model.PlotAreas(0).Series(0),
FarPoint.Win.Chart.PieSeries)
series.LabelVisible = True
series.LabelContainsCategoryName = True
series.LabelContainsValue = True
```

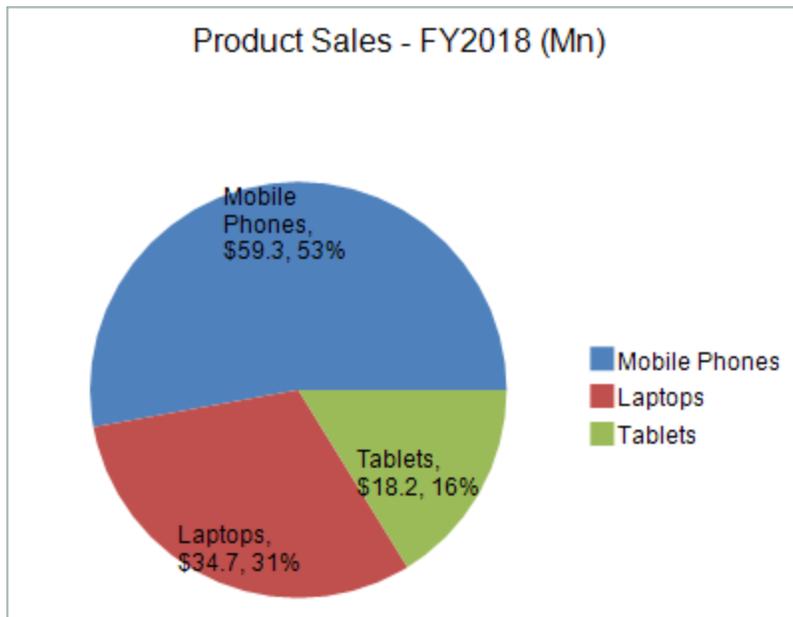
You can also add percentage labels in the pie chart by using **LabelContainsPercentage** property of the **PieSeries** class. It accepts boolean value and is false by default.

C#

```
// Enable Percentage label in pie chart
series.LabelContainsPercentage = true;
```

Visual Basic

```
// Enable Percentage label in pie chart  
series.LabelContainsPercentage = true;
```

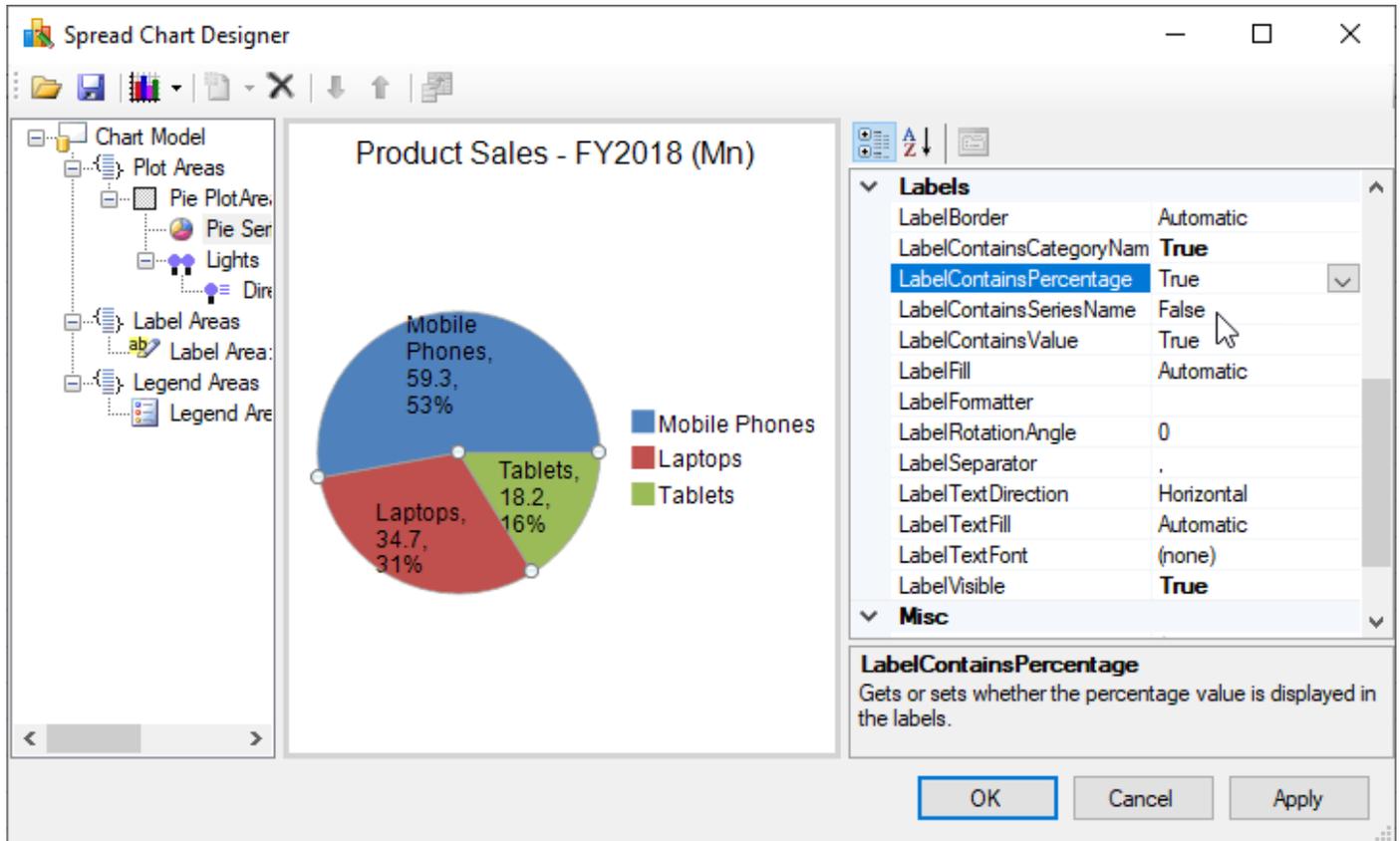


Using the Chart designer

Pie charts can be created in the Chart Designer using the following steps -

1. Run **Chart Designer**.
2. Select the target **Chart Model** from the tree menu on the left.
3. Open the **Plot Area Collection Editor** from PlotAreas in the **Other** section of the property list on the right.
4. Click the drop-down button to the right of the **Add** button.
5. Select and add the **PiePlotArea** and set each property as required.

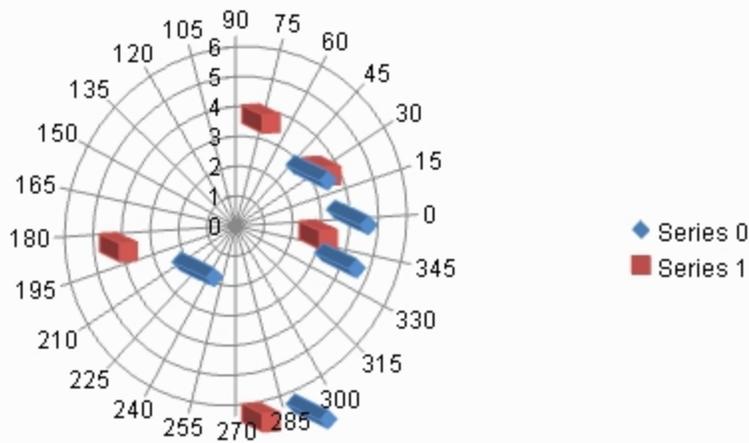
You can also enable percentage labels in the Chart Designer by selecting the Pie Series target from the tree menu on the left and enable the **LabelContainsPercentage** option to True on the property list.



For information on starting **Chart Designer** ('**Spread Designer Guide**' in the on-line documentation), refer to Chart Designer in the **SPREAD Designer Guide** ('**Using the Chart Designer**' in the on-line documentation).

Creating a Polar Plot

You can create a polar plot chart using code or the designer. The following image shows a Polar Plot type chart.



For details on the API, see the **PolarPlotArea ('PolarPlotArea Class' in the on-line documentation)** class. The following classes are also available when creating Polar plot type charts:

- Point chart (scatter plot)
PolarPointSeries ('PolarPointSeries Class' in the on-line documentation)
- Line Chart
PolarLineSeries ('PolarLineSeries Class' in the on-line documentation)
- Area Chart
PolarAreaSeries ('PolarAreaSeries Class' in the on-line documentation)

In addition, value axes such as ruler lines, memory, and scale labels are set using the following classes.

- **PolarAngleAxis ('PolarAngleAxis Class' in the on-line documentation)**
- **PolarRadiusAxis ('PolarRadiusAxis Class' in the on-line documentation)**

Using Code

1. Create a **PolarPointSeries ('PolarPointSeries Class' in the on-line documentation)** object that represents the data series for the circle chart and add the data.
2. Use the **PolarPlotArea ('PolarPlotArea Class' in the on-line documentation)** class to create the plot area.
3. Add the series to the plot area.
4. Create a label and legend for the chart.
5. Create a **ChartModel ('ChartModel Class' in the on-line documentation)** object and add the plot area, label, and legend to the model.
6. Create a chart and add the chart model to it.

Example

The following example demonstrates creating a polar point series chart.

C#

```
FarPoint.Win.Chart.PolarPointSeries series0 = new
FarPoint.Win.Chart.PolarPointSeries();
series0.SeriesName = "Series 0";
series0.XValues.Add(0.0);
series0.XValues.Add(45.0);
```

```
series0.XValues.Add(90.0);
series0.XValues.Add(180.0);
series0.XValues.Add(270.0);
series0.YValues.Add(1.0);
series0.YValues.Add(2.0);
series0.YValues.Add(3.0);
series0.YValues.Add(4.0);
series0.YValues.Add(5.0);
FarPoint.Win.Chart.PolarPointSeries series1 = new
FarPoint.Win.Chart.PolarPointSeries();
series1.SeriesName = "Series 1";
series1.XValues.Add(0.0);
series1.XValues.Add(45.0);
series1.XValues.Add(90.0);
series1.XValues.Add(180.0);
series1.XValues.Add(270.0);
series1.YValues.Add(2.0);
series1.YValues.Add(3.0);
series1.YValues.Add(4.0);
series1.YValues.Add(5.0);
series1.YValues.Add(6.0);
FarPoint.Win.Chart.PolarPlotArea plotArea = new FarPoint.Win.Chart.PolarPlotArea();
plotArea.Location = new PointF(0.2f, 0.2f);
plotArea.Size = new SizeF(0.6f, 0.6f);
plotArea.Series.Add(series0);
plotArea.Series.Add(series1);
FarPoint.Win.Chart.LabelArea label = new FarPoint.Win.Chart.LabelArea();
label.Text = "Polar Point Chart";
label.Location = new PointF(0.5f, 0.02f);
label.AlignmentX = 0.5f;
label.AlignmentY = 0.0f;
FarPoint.Win.Chart.LegendArea legend = new FarPoint.Win.Chart.LegendArea();
legend.Location = new PointF(0.98f, 0.5f);
legend.AlignmentX = 1.0f;
legend.AlignmentY = 0.5f;
FarPoint.Win.Chart.ChartModel model = new FarPoint.Win.Chart.ChartModel();
model.LabelAreas.Add(label);
model.LegendAreas.Add(legend);
model.PlotAreas.Add(plotArea);
chart2DControl1.Model = model;
```

Visual Basic

```
Dim series0 As New FarPoint.Win.Chart.PolarPointSeries()
series0.SeriesName = "Series 0"
series0.XValues.Add(0.0)
series0.XValues.Add(45.0)
series0.XValues.Add(90.0)
series0.XValues.Add(180.0)
series0.XValues.Add(270.0)
series0.YValues.Add(1.0)
series0.YValues.Add(2.0)
series0.YValues.Add(3.0)
series0.YValues.Add(4.0)
series0.YValues.Add(5.0)
Dim series1 As New FarPoint.Win.Chart.PolarPointSeries()
series1.SeriesName = "Series 1"
```

```
series1.XValues.Add(0.0)
series1.XValues.Add(45.0)
series1.XValues.Add(90.0)
series1.XValues.Add(180.0)
series1.XValues.Add(270.0)
series1.YValues.Add(2.0)
series1.YValues.Add(3.0)
series1.YValues.Add(4.0)
series1.YValues.Add(5.0)
series1.YValues.Add(6.0)
Dim plotArea As New FarPoint.Win.Chart.PolarPlotArea()
plotArea.Location = New PointF(0.2F, 0.2F)
plotArea.Size = New SizeF(0.6F, 0.6F)
plotArea.Series.Add(series0)
plotArea.Series.Add(series1)
Dim label As New FarPoint.Win.Chart.LabelArea()
label.Text = "Polar Point Chart"
label.Location = New PointF(0.5F, 0.02F)
label.AlignmentX = 0.5F
label.AlignmentY = 0F
Dim legend As New FarPoint.Win.Chart.LegendArea()
legend.Location = New PointF(0.98F, 0.5F)
legend.AlignmentX = 1.0F
legend.AlignmentY = 0.5F
Dim model As New FarPoint.Win.Chart.ChartModel()
model.LabelAreas.Add(label)
model.LegendAreas.Add(legend)
model.PlotAreas.Add(plotArea)
chart2DControl1.Model = model
```

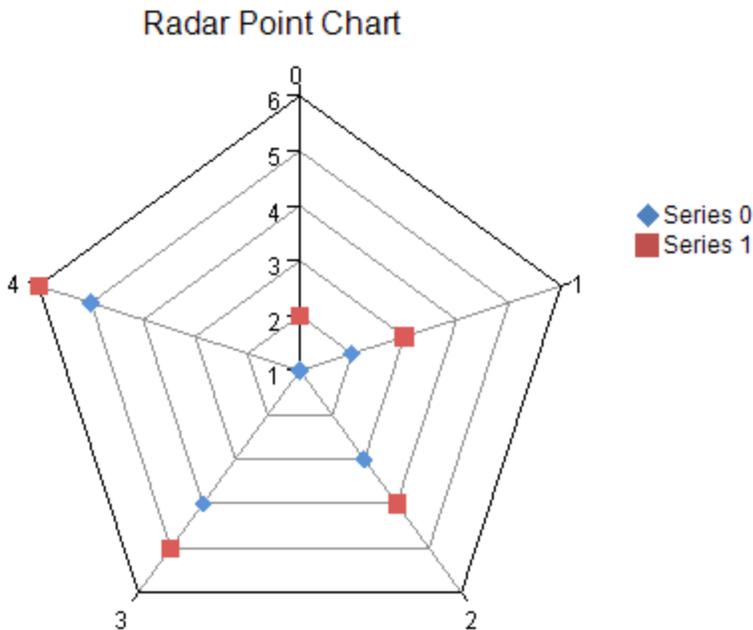
Using the Chart designer

1. Run **Chart Designer**.
2. Select the target **Chart Model** from the tree menu on the left.
3. Open the **Plot Area Collection Editor** from PlotAreas in the **Other** section of the property list on the right.
4. Click the drop-down button to the right of the **Add** button.
5. Select and add the **PolarPlotArea** and set each property as required.

 For information on starting **Chart Designer ('Spread Designer Guide' in the on-line documentation)**, refer to Chart Designer in the **SPREAD Designer Guide ('Using the Chart Designer' in the on-line documentation)**.

Creating a Radar Plot

You can create a radar plot chart using code or the designer. The following image shows a Radar point type chart.



For details on the API, see the **RadarPlotArea** ('**RadarPlotArea Class**' in the on-line documentation) class. The following classes are also available when creating Radar plot type charts:

- point Chart
RadarPointSeries ('**RadarPointSeries Class**' in the on-line documentation)
- Line Chart
RadarLineSeries ('**RadarLineSeries Class**' in the on-line documentation)
- Area Chart
RadarAreaSeries ('**RadarAreaSeries Class**' in the on-line documentation)

In addition, value axes such as ruler lines, memory, and scale labels are set using the following classes.

- **RadarIndexAxis** ('**RadarIndexAxis Class**' in the on-line documentation)
- **RadarValueAxis** ('**RadarValueAxis Class**' in the on-line documentation)

Using Code

1. Create a **RadarPointSeries** ('**RadarPointSeries Class**' in the on-line documentation) object that represents the data series of the radar (marker) chart and add the data.
2. Create a **RadarPlotArea** ('**RadarPlotArea Class**' in the on-line documentation) object that represents the plot area and set its position and size.
3. Add a data series to the plot area.
4. Create labels and legend areas.
5. Create a **ChartModel** ('**ChartModel Class**' in the on-line documentation) object and add plot areas, labels, and legend areas.
6. Assign a chart model to the chart.

Example

The following example demonstrates creating a Radar chart.

C#

```
FarPoint.Win.Chart.RadarPointSeries series0 = new
```

```
FarPoint.Win.Chart.RadarPointSeries();
series0.SeriesName = "Series 0";
series0.Values.Add(1.0);
series0.Values.Add(2.0);
series0.Values.Add(3.0);
series0.Values.Add(4.0);
series0.Values.Add(5.0);
FarPoint.Win.Chart.RadarPointSeries series1 = new
FarPoint.Win.Chart.RadarPointSeries();
series1.SeriesName = "Series 1";
series1.Values.Add(2.0);
series1.Values.Add(3.0);
series1.Values.Add(4.0);
series1.Values.Add(5.0);
series1.Values.Add(6.0);
FarPoint.Win.Chart.RadarPlotArea plotArea = new FarPoint.Win.Chart.RadarPlotArea();
plotArea.Location = new PointF(0.2f, 0.2f);
plotArea.Size = new SizeF(0.6f, 0.6f);
plotArea.Series.Add(series0);
plotArea.Series.Add(series1);
FarPoint.Win.Chart.LabelArea label = new FarPoint.Win.Chart.LabelArea();
label.Text = "Radar Point Chart";
label.Location = new PointF(0.5f, 0.02f);
label.AlignmentX = 0.5f;
label.AlignmentY = 0.0f;
FarPoint.Win.Chart.LegendArea legend = new FarPoint.Win.Chart.LegendArea();
legend.Location = new PointF(0.98f, 0.5f);
legend.AlignmentX = 1.0f;
legend.AlignmentY = 0.5f;
FarPoint.Win.Chart.ChartModel model = new FarPoint.Win.Chart.ChartModel();
model.LabelAreas.Add(label);
model.LegendAreas.Add(legend);
model.PlotAreas.Add(plotArea);
chart2DControl1.Model = model;
```

Visual Basic

```
Dim series0 As New FarPoint.Win.Chart.RadarPointSeries()
series0.SeriesName = "Series 0"
series0.Values.Add(1.0)
series0.Values.Add(2.0)
series0.Values.Add(3.0)
series0.Values.Add(4.0)
series0.Values.Add(5.0)
Dim series1 As New FarPoint.Win.Chart.RadarPointSeries()
series1.SeriesName = "Series 1"
series1.Values.Add(2.0)
series1.Values.Add(3.0)
series1.Values.Add(4.0)
series1.Values.Add(5.0)
series1.Values.Add(6.0)
Dim plotArea As New FarPoint.Win.Chart.RadarPlotArea()
plotArea.Location = New PointF(0.2F, 0.2F)
plotArea.Size = New SizeF(0.6F, 0.6F)
plotArea.Series.Add(series0)
plotArea.Series.Add(series1)
Dim label As New FarPoint.Win.Chart.LabelArea()
```

```
label.Text = "Radar Point Chart"  
label.Location = New PointF(0.5F, 0.02F)  
label.AlignmentX = 0.5F  
label.AlignmentY = 0.0F  
Dim legend As New FarPoint.Win.Chart.LegendArea()  
legend.Location = New PointF(0.98F, 0.5F)  
legend.AlignmentX = 1.0F  
legend.AlignmentY = 0.5F  
Dim model As New FarPoint.Win.Chart.ChartModel()  
model.LabelAreas.Add(label)  
model.LegendAreas.Add(legend)  
model.PlotAreas.Add(plotArea)  
Chart2DControl1.Model = model
```

Using the Chart designer

1. Run **Chart Designer**.
2. Select the target **Chart Model** from the tree menu on the left.
3. Open the **Plot Area Collection Editor** from PlotAreas in the **Other** section of the property list on the right.
4. Click the drop-down button to the right of the **Add** button.
5. Select and add the **RadarPlotArea** and set each property as required.

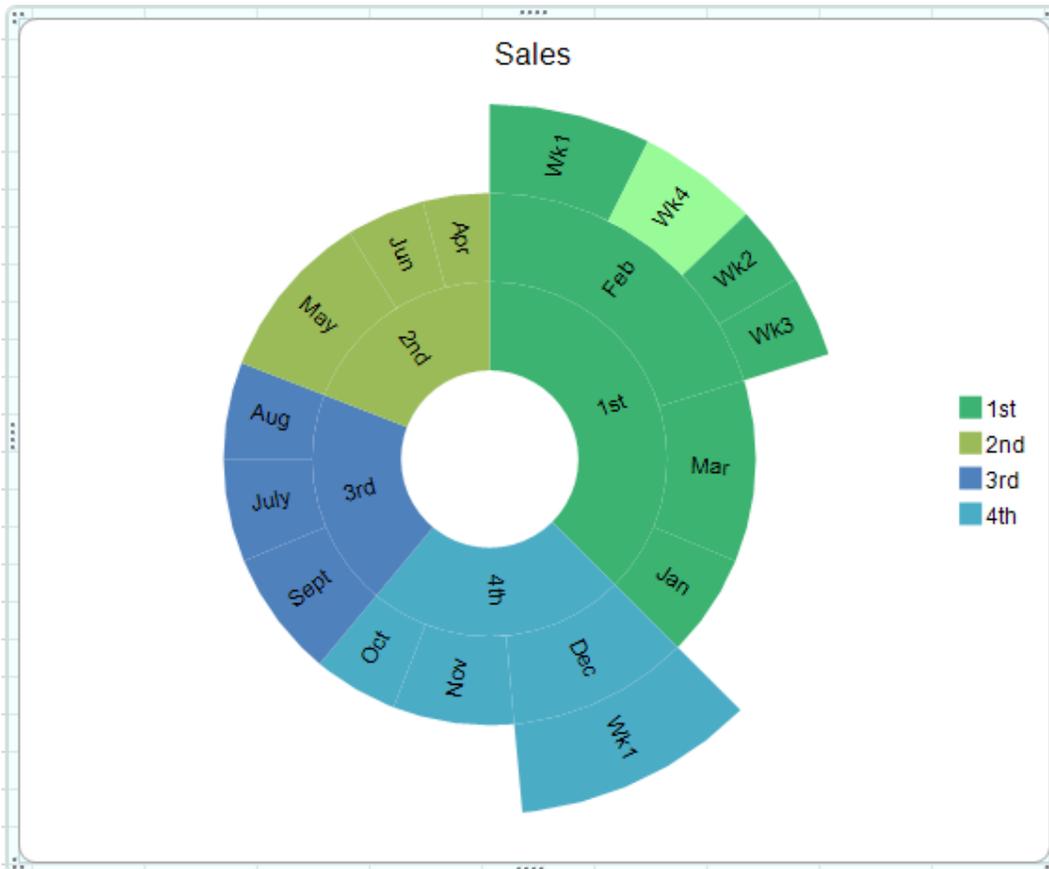


For information on starting **Chart Designer ('Spread Designer Guide' in the on-line documentation)**, refer to Chart Designer in the **SPREAD Designer Guide ('Using the Chart Designer' in the on-line documentation)**.

Creating a Sunburst Chart

You can use the sunburst chart to display hierarchical data. Each hierarchy level is represented by one ring or circle in the chart. The innermost circle is the top of the hierarchy.

The following image displays multiple rings of data.



You can use the **SunburstSeries** ('SunburstSeries Class' in the on-line documentation) and **SunburstPlotArea** ('SunburstPlotArea Class' in the on-line documentation) classes to add a sunburst chart.

For information about creating charts in the Spread Designer or the Chart Designer, refer to **Using the Chart Control on sheet** or **Adding Charts (on-line documentation)**.

Using Code

1. Add data for the chart.
2. Add the chart.
3. Set any additional properties such as the fill color.

Example

This example creates a sunburst chart.

C#

```
fpSpread1.ActiveSheet.Cells[0, 0].Text = "Quarter";
fpSpread1.ActiveSheet.Cells[0, 1].Text = "Month";
fpSpread1.ActiveSheet.Cells[0, 2].Text = "Week";
fpSpread1.ActiveSheet.Cells[0, 3].Text = "Sales";
fpSpread1.ActiveSheet.Cells[1, 0].Text = "1st";
fpSpread1.ActiveSheet.Cells[1, 1].Text = "Jan";
fpSpread1.ActiveSheet.Cells[1, 3].Value = 1.7;
fpSpread1.ActiveSheet.Cells[2, 1].Text = "Feb";
fpSpread1.ActiveSheet.Cells[2, 2].Text = "Wk1";
fpSpread1.ActiveSheet.Cells[2, 3].Value = 2.0;
fpSpread1.ActiveSheet.Cells[3, 2].Text = "Wk2";
```

```

fpSpread1.ActiveSheet.Cells[3, 3].Value = 1.0;
fpSpread1.ActiveSheet.Cells[4, 2].Text = "Wk3";
fpSpread1.ActiveSheet.Cells[4, 3].Value = 1.0;
fpSpread1.ActiveSheet.Cells[5, 2].Text = "Wk4";
fpSpread1.ActiveSheet.Cells[5, 3].Value = 1.5;
fpSpread1.ActiveSheet.Cells[6, 1].Text = "Mar";
fpSpread1.ActiveSheet.Cells[6, 3].Value = 3.0;
fpSpread1.ActiveSheet.Cells[7, 0].Text = "2nd";
fpSpread1.ActiveSheet.Cells[7, 1].Text = "Apr";
fpSpread1.ActiveSheet.Cells[7, 3].Value = 1.1;
fpSpread1.ActiveSheet.Cells[8, 1].Text = "May";
fpSpread1.ActiveSheet.Cells[8, 3].Value = 2.8;
fpSpread1.ActiveSheet.Cells[9, 1].Text = "Jun";
fpSpread1.ActiveSheet.Cells[9, 3].Value = 1.3;
fpSpread1.ActiveSheet.Cells[10, 0].Text = "3rd";
fpSpread1.ActiveSheet.Cells[10, 1].Text = "July";
fpSpread1.ActiveSheet.Cells[10, 3].Value = 1.7;
fpSpread1.ActiveSheet.Cells[11, 1].Text = "Aug";
fpSpread1.ActiveSheet.Cells[11, 3].Value = 1.6;
fpSpread1.ActiveSheet.Cells[12, 1].Text = "Sept";
fpSpread1.ActiveSheet.Cells[12, 3].Value = 2.1;
fpSpread1.ActiveSheet.Cells[13, 0].Text = "4th";
fpSpread1.ActiveSheet.Cells[13, 1].Text = "Oct";
fpSpread1.ActiveSheet.Cells[13, 3].Value = 1.4;
fpSpread1.ActiveSheet.Cells[14, 1].Text = "Nov";
fpSpread1.ActiveSheet.Cells[14, 3].Value = 2.0;
fpSpread1.ActiveSheet.Cells[15, 1].Text = "Dec";
fpSpread1.ActiveSheet.Cells[15, 2].Text = "Wk1";
fpSpread1.ActiveSheet.Cells[15, 3].Value = 3.0;
fpSpread1.ActiveSheet.AddChart(new FarPoint.Win.Spread.Model.CellRange(0, 0, 16, 4),
typeof(FarPoint.Win.Chart.SunburstSeries), 550, 450, 300, 0);
FarPoint.Win.Chart.SunburstSeries sunseries =
(FarPoint.Win.Chart.SunburstSeries) fpSpread1.Sheets[0].Charts[0].Model.PlotAreas[0].Series[0];
sunseries.Fills.AddRange(new FarPoint.Win.Chart.Fill[] { new
FarPoint.Win.Chart.SolidFill(Color.MediumSeaGreen), null, null, null, null, null, new
FarPoint.Win.Chart.SolidFill(Color.PaleGreen), null, null });

```

VB

```

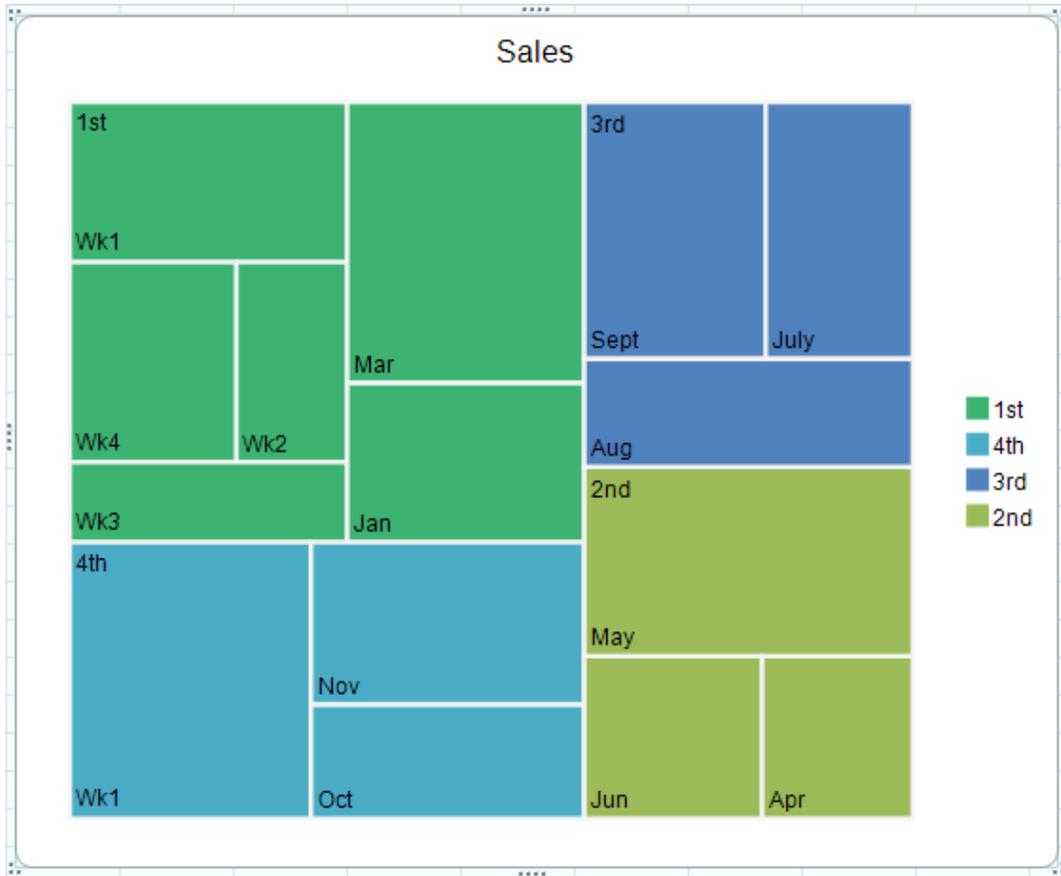
fpSpread1.ActiveSheet.Cells(0, 0).Text = "Quarter"
fpSpread1.ActiveSheet.Cells(0, 1).Text = "Month"
fpSpread1.ActiveSheet.Cells(0, 2).Text = "Week"
fpSpread1.ActiveSheet.Cells(0, 3).Text = "Sales"
fpSpread1.ActiveSheet.Cells(1, 0).Text = "1st"
fpSpread1.ActiveSheet.Cells(1, 1).Text = "Jan"
fpSpread1.ActiveSheet.Cells(1, 3).Value = 1.7
fpSpread1.ActiveSheet.Cells(2, 1).Text = "Feb"
fpSpread1.ActiveSheet.Cells(2, 2).Text = "Wk1"
fpSpread1.ActiveSheet.Cells(2, 3).Value = 2.0
fpSpread1.ActiveSheet.Cells(3, 2).Text = "Wk2"
fpSpread1.ActiveSheet.Cells(3, 3).Value = 1.0
fpSpread1.ActiveSheet.Cells(4, 2).Text = "Wk3"
fpSpread1.ActiveSheet.Cells(4, 3).Value = 1.0
fpSpread1.ActiveSheet.Cells(5, 2).Text = "Wk4"
fpSpread1.ActiveSheet.Cells(5, 3).Value = 1.5
fpSpread1.ActiveSheet.Cells(6, 1).Text = "Mar"
fpSpread1.ActiveSheet.Cells(6, 3).Value = 3.0
fpSpread1.ActiveSheet.Cells(7, 0).Text = "2nd"
fpSpread1.ActiveSheet.Cells(7, 1).Text = "Apr"
fpSpread1.ActiveSheet.Cells(7, 3).Value = 1.1

```

```
fpSpread1.ActiveSheet.Cells(8, 1).Text = "May"
fpSpread1.ActiveSheet.Cells(8, 3).Value = 2.8
fpSpread1.ActiveSheet.Cells(9, 1).Text = "Jun"
fpSpread1.ActiveSheet.Cells(9, 3).Value = 1.3
fpSpread1.ActiveSheet.Cells(10, 0).Text = "3rd"
fpSpread1.ActiveSheet.Cells(10, 1).Text = "July"
fpSpread1.ActiveSheet.Cells(10, 3).Value = 1.7
fpSpread1.ActiveSheet.Cells(11, 1).Text = "Aug"
fpSpread1.ActiveSheet.Cells(11, 3).Value = 1.6
fpSpread1.ActiveSheet.Cells(12, 1).Text = "Sept"
fpSpread1.ActiveSheet.Cells(12, 3).Value = 2.1
fpSpread1.ActiveSheet.Cells(13, 0).Text = "4th"
fpSpread1.ActiveSheet.Cells(13, 1).Text = "Oct"
fpSpread1.ActiveSheet.Cells(13, 3).Value = 1.4
fpSpread1.ActiveSheet.Cells(14, 1).Text = "Nov"
fpSpread1.ActiveSheet.Cells(14, 3).Value = 2.0
fpSpread1.ActiveSheet.Cells(15, 1).Text = "Dec"
fpSpread1.ActiveSheet.Cells(15, 2).Text = "Wk1"
fpSpread1.ActiveSheet.Cells(15, 3).Value = 3.0
fpSpread1.ActiveSheet.AddChart(New FarPoint.Win.Spread.Model.CellRange(0, 0, 16, 4),
GetType(FarPoint.Win.Chart.SunburstSeries), 550, 450, 300, 0)
Dim sunseries As FarPoint.Win.Chart.SunburstSeries =
DirectCast(fpSpread1.Sheets(0).Charts(0).Model.PlotAreas(0).Series(0),
FarPoint.Win.Chart.SunburstSeries)
sunseries.Fills.AddRange(New FarPoint.Win.Chart.Fill() {New
FarPoint.Win.Chart.SolidFill(Color.MediumSeaGreen), Nothing, Nothing, Nothing, Nothing,
Nothing, New FarPoint.Win.Chart.SolidFill(Color.PaleGreen), Nothing, Nothing})
```

Creating a Treemap Chart

A treemap chart displays hierarchical data as a set of nested rectangles. The tree branches are represented by rectangles and each sub-branch is shown as a smaller rectangle. The rectangles in the chart use color and size to make it easier to spot patterns. Treemap charts also make efficient use of space and are useful for displaying large amounts of data.



You can use the **TreemapSeries** ('**TreemapSeries Class**' in the on-line documentation) class and the **TreemapPlotArea** ('**TreemapPlotArea Class**' in the on-line documentation) class to create a treemap chart.

You can set a color for each item in the treemap chart using the **Fills** ('**Fills Property**' in the on-line documentation) property. Colors are mapped to the items based on the index order. The following image and code example show how the colors would be mapped to each data item.

1st	Jan		3.5	Index	Name	Color
	Fed	Week1	1.2	0	1st	Blue
		Week2	0.8	1	Jan	Aqua
		Week3	0.6	2	Fed	Coral
		Week4	0.5	3	Week1	Lavender
	Mar		1.7	4	Week2	Lavender
2nd	Apr		1.1	5	Week3	Lavender
	May		0.7	6	Week4	Lavender
	June		1.3	7	Mar	Olive
3rd	July		2	8	2nd	Orange
				9	Apr	Beige
				10	May	Firebrick
				11	June	Gray
				12	3rd	Magenta
				13	July	

Chart data

Data index

C#

```

FarPoint.Win.Chart.TreemapSeries series = new FarPoint.Win.Chart.TreemapSeries();
series.Values.AddRange(new double[] { 3.5, 1.2, 0.8, 0.6, 0.5, 1.7, 1.1, 0.7, 1.3, 2.0 });
series.Fills.AddRange(new FarPoint.Win.Chart.Fill[] { new
FarPoint.Win.Chart.SolidFill(Color.Blue), new FarPoint.Win.Chart.SolidFill(Color.Aqua), new
FarPoint.Win.Chart.SolidFill(Color.Coral), new
FarPoint.Win.Chart.SolidFill(Color.Lavender), new
FarPoint.Win.Chart.SolidFill(Color.Lavender), new
FarPoint.Win.Chart.SolidFill(Color.Lavender), new
FarPoint.Win.Chart.SolidFill(Color.Lavender), new
FarPoint.Win.Chart.SolidFill(Color.Olive), new FarPoint.Win.Chart.SolidFill(Color.Orange),
new FarPoint.Win.Chart.SolidFill(Color.Beige), new
FarPoint.Win.Chart.SolidFill(Color.Firebrick), new
FarPoint.Win.Chart.SolidFill(Color.Gray), new FarPoint.Win.Chart.SolidFill(Color.Magenta)
});

FarPoint.Win.Chart.StringCollectionItem collection1 = new
FarPoint.Win.Chart.StringCollectionItem();
collection1.AddRange(new String[] { "1st", "", "", "", "", "", "2nd", "", "", "3rd" });
FarPoint.Win.Chart.StringCollectionItem collection2 = new
FarPoint.Win.Chart.StringCollectionItem();
collection2.AddRange(new String[] { "Jan", "Feb", "", "", "", "Mar", "Apr", "May", "June",
"July" });
FarPoint.Win.Chart.StringCollectionItem collection3 = new
FarPoint.Win.Chart.StringCollectionItem();
collection3.AddRange(new String[] { "", "Week1", "Week2", "Week3", "Week4", "", "", "", "",
"" });
series.CategoryNames.AddRange(new FarPoint.Win.Chart.StringCollectionItem[] { collection1,
collection2, collection3 });
FarPoint.Win.Chart.TreemapPlotArea plotArea = new FarPoint.Win.Chart.TreemapPlotArea();
plotArea.Location = new PointF(0.2f, 0.2f);
plotArea.Size = new SizeF(0.6f, 0.6f);
plotArea.Series.Add(series);
FarPoint.Win.Chart.ChartModel model = new FarPoint.Win.Chart.ChartModel();
    
```

```
model.PlotAreas.Add(plotArea);
FarPoint.Win.Spread.Chart.SpreadChart chart = new FarPoint.Win.Spread.Chart.SpreadChart();
chart.Size = new Size(500, 500);
chart.Location = new Point(50, 50);
chart.Model = model;
fpSpread1.Sheets[0].Charts.Add(chart);
```

For information about creating charts in the Spread Designer or the Chart Designer, refer to **Using the Chart Control on sheet** or **Adding Charts (on-line documentation)**.

Using Code

1. Add data for the chart.
2. Add the chart.
3. Set any additional properties such as the fill color.

Example

This example creates a treemap chart.

C#

```
fpSpread1.ActiveSheet.Cells[0, 0].Text = "Quarter";
fpSpread1.ActiveSheet.Cells[0, 1].Text = "Month";
fpSpread1.ActiveSheet.Cells[0, 2].Text = "Week";
fpSpread1.ActiveSheet.Cells[0, 3].Text = "Sales";
fpSpread1.ActiveSheet.Cells[1, 0].Text = "1st";
fpSpread1.ActiveSheet.Cells[1, 1].Text = "Jan";
fpSpread1.ActiveSheet.Cells[1, 3].Value = 1.7;
fpSpread1.ActiveSheet.Cells[2, 1].Text = "Feb";
fpSpread1.ActiveSheet.Cells[2, 2].Text = "Wk1";
fpSpread1.ActiveSheet.Cells[2, 3].Value = 2.0;
fpSpread1.ActiveSheet.Cells[3, 2].Text = "Wk2";
fpSpread1.ActiveSheet.Cells[3, 3].Value = 1.0;
fpSpread1.ActiveSheet.Cells[4, 2].Text = "Wk3";
fpSpread1.ActiveSheet.Cells[4, 3].Value = 1.0;
fpSpread1.ActiveSheet.Cells[5, 2].Text = "Wk4";
fpSpread1.ActiveSheet.Cells[5, 3].Value = 1.5;
fpSpread1.ActiveSheet.Cells[6, 1].Text = "Mar";
fpSpread1.ActiveSheet.Cells[6, 3].Value = 3.0;
fpSpread1.ActiveSheet.Cells[7, 0].Text = "2nd";
fpSpread1.ActiveSheet.Cells[7, 1].Text = "Apr";
fpSpread1.ActiveSheet.Cells[7, 3].Value = 1.1;
fpSpread1.ActiveSheet.Cells[8, 1].Text = "May";
fpSpread1.ActiveSheet.Cells[8, 3].Value = 2.8;
fpSpread1.ActiveSheet.Cells[9, 1].Text = "Jun";
fpSpread1.ActiveSheet.Cells[9, 3].Value = 1.3;
fpSpread1.ActiveSheet.Cells[10, 0].Text = "3rd";
fpSpread1.ActiveSheet.Cells[10, 1].Text = "July";
fpSpread1.ActiveSheet.Cells[10, 3].Value = 1.7;
fpSpread1.ActiveSheet.Cells[11, 1].Text = "Aug";
fpSpread1.ActiveSheet.Cells[11, 3].Value = 1.6;
fpSpread1.ActiveSheet.Cells[12, 1].Text = "Sept";
fpSpread1.ActiveSheet.Cells[12, 3].Value = 2.1;
fpSpread1.ActiveSheet.Cells[13, 0].Text = "4th";
fpSpread1.ActiveSheet.Cells[13, 1].Text = "Oct";
fpSpread1.ActiveSheet.Cells[13, 3].Value = 1.4;
fpSpread1.ActiveSheet.Cells[14, 1].Text = "Nov";
fpSpread1.ActiveSheet.Cells[14, 3].Value = 2.0;
```

```

fpSpread1.ActiveSheet.Cells[15, 1].Text = "Dec";
fpSpread1.ActiveSheet.Cells[15, 2].Text = "Wk1";
fpSpread1.ActiveSheet.Cells[15, 3].Value = 3.0;
fpSpread1.ActiveSheet.AddChart(new FarPoint.Win.Spread.Model.CellRange(0, 0, 16, 4),
typeof(FarPoint.Win.Chart.TreemapSeries), 550, 450, 300, 0);
FarPoint.Win.Chart.TreemapSeries treeseries =
(FarPoint.Win.Chart.TreemapSeries)fpSpread1.Sheets[0].Charts[0].Model.PlotAreas[0].Series[0];
treeseries.Fills.AddRange(new FarPoint.Win.Chart.Fill[] { new
FarPoint.Win.Chart.SolidFill(Color.MediumSeaGreen), null, null, null, null });

```

VB

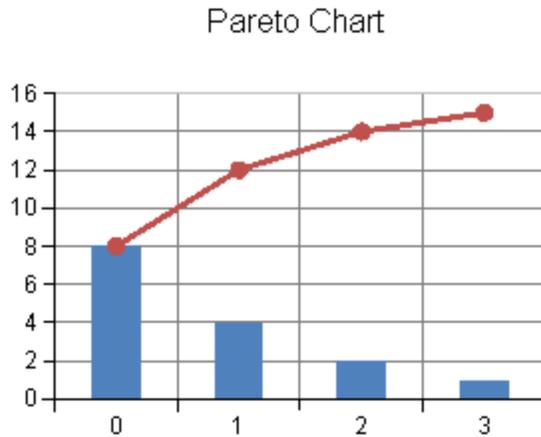
```

fpSpread1.ActiveSheet.Cells(0, 0).Text = "Quarter"
fpSpread1.ActiveSheet.Cells(0, 1).Text = "Month"
fpSpread1.ActiveSheet.Cells(0, 2).Text = "Week"
fpSpread1.ActiveSheet.Cells(0, 3).Text = "Sales"
fpSpread1.ActiveSheet.Cells(1, 0).Text = "1st"
fpSpread1.ActiveSheet.Cells(1, 1).Text = "Jan"
fpSpread1.ActiveSheet.Cells(1, 3).Value = 1.7
fpSpread1.ActiveSheet.Cells(2, 1).Text = "Feb"
fpSpread1.ActiveSheet.Cells(2, 2).Text = "Wk1"
fpSpread1.ActiveSheet.Cells(2, 3).Value = 2.0
fpSpread1.ActiveSheet.Cells(3, 2).Text = "Wk2"
fpSpread1.ActiveSheet.Cells(3, 3).Value = 1.0
fpSpread1.ActiveSheet.Cells(4, 2).Text = "Wk3"
fpSpread1.ActiveSheet.Cells(4, 3).Value = 1.0
fpSpread1.ActiveSheet.Cells(5, 2).Text = "Wk4"
fpSpread1.ActiveSheet.Cells(5, 3).Value = 1.5
fpSpread1.ActiveSheet.Cells(6, 1).Text = "Mar"
fpSpread1.ActiveSheet.Cells(6, 3).Value = 3.0
fpSpread1.ActiveSheet.Cells(7, 0).Text = "2nd"
fpSpread1.ActiveSheet.Cells(7, 1).Text = "Apr"
fpSpread1.ActiveSheet.Cells(7, 3).Value = 1.1
fpSpread1.ActiveSheet.Cells(8, 1).Text = "May"
fpSpread1.ActiveSheet.Cells(8, 3).Value = 2.8
fpSpread1.ActiveSheet.Cells(9, 1).Text = "Jun"
fpSpread1.ActiveSheet.Cells(9, 3).Value = 1.3
fpSpread1.ActiveSheet.Cells(10, 0).Text = "3rd"
fpSpread1.ActiveSheet.Cells(10, 1).Text = "July"
fpSpread1.ActiveSheet.Cells(10, 3).Value = 1.7
fpSpread1.ActiveSheet.Cells(11, 1).Text = "Aug"
fpSpread1.ActiveSheet.Cells(11, 3).Value = 1.6
fpSpread1.ActiveSheet.Cells(12, 1).Text = "Sept"
fpSpread1.ActiveSheet.Cells(12, 3).Value = 2.1
fpSpread1.ActiveSheet.Cells(13, 0).Text = "4th"
fpSpread1.ActiveSheet.Cells(13, 1).Text = "Oct"
fpSpread1.ActiveSheet.Cells(13, 3).Value = 1.4
fpSpread1.ActiveSheet.Cells(14, 1).Text = "Nov"
fpSpread1.ActiveSheet.Cells(14, 3).Value = 2.0
fpSpread1.ActiveSheet.Cells(15, 1).Text = "Dec"
fpSpread1.ActiveSheet.Cells(15, 2).Text = "Wk1"
fpSpread1.ActiveSheet.Cells(15, 3).Value = 3.0
fpSpread1.ActiveSheet.AddChart(New FarPoint.Win.Spread.Model.CellRange(0, 0, 16, 4),
GetType(FarPoint.Win.Chart.TreemapSeries), 550, 450, 300, 0)
Dim treeseries As FarPoint.Win.Chart.TreemapSeries =
DirectCast(fpSpread1.Sheets(0).Charts(0).Model.PlotAreas(0).Series(0),
FarPoint.Win.Chart.TreemapSeries)
treeseries.Fills.AddRange(New FarPoint.Win.Chart.Fill() {New
FarPoint.Win.Chart.SolidFill(Color.MediumSeaGreen), Nothing, Nothing, Nothing, Nothing})

```

Combining different types of plots

Different types of series are compatible with each other and can be displayed in a single plot area, as long as the main classifications are the same series. For example, you can display bar and line series in the same Y plot area, as shown in the following image.



For more information on the API, see the **YPlotArea ('YPlotArea Class' in the on-line documentation)** class. Also, in the above image, the following classes are used to create bar and line series.

- **BarSeries ('BarSeries Class' in the on-line documentation)**
Represents the data series of a bar chart.
- **LineSeries ('LineSeries Class' in the on-line documentation)**
Represents the data series of a line chart.

Using the code

1. Create **BarSeries ('BarSeries Class' in the on-line documentation)** and **LineSeries ('LineSeries Class' in the on-line documentation)** objects to add data.
2. Create a **YPlotArea ('YPlotArea Class' in the on-line documentation)** object that represents the plot area and set its position and size.
3. Add the two data series you created to the plot area.
4. Create a label area.
5. Create a **ChartModel ('ChartModel Class' in the on-line documentation)** object and add plot areas, labels, and legend areas.
6. Assign a chart model to the chart.

Example

The following example demonstrates how to display a bar series and a line series in the same plot area.

C#

```
FarPoint.Win.Chart.BarSeries series0 = new FarPoint.Win.Chart.BarSeries();
series0.Values.Add(8.0);
series0.Values.Add(4.0);
series0.Values.Add(2.0);
series0.Values.Add(1.0);
FarPoint.Win.Chart.LineSeries series1 = new FarPoint.Win.Chart.LineSeries();
series1.PointMarker = new
```

```
FarPoint.Win.Chart.BuiltinMarker(FarPoint.Win.Chart.MarkerShape.Circle, 7.0f);
series1.Values.Add(8.0);
series1.Values.Add(12.0);
series1.Values.Add(14.0);
series1.Values.Add(15.0);
FarPoint.Win.Chart.YPlotArea plotArea = new FarPoint.Win.Chart.YPlotArea();
plotArea.Location = new PointF(0.2f, 0.2f);
plotArea.Size = new SizeF(0.6f, 0.6f);
plotArea.Series.Add(series0);
plotArea.Series.Add(series1);
FarPoint.Win.Chart.LabelArea labelArea = new FarPoint.Win.Chart.LabelArea();
labelArea.Location = new PointF(0.5f, 0.02f);
labelArea.AlignmentX = 0.5f;
labelArea.AlignmentY = 0.0f;
labelArea.Text = "Pareto Chart";
FarPoint.Win.Chart.ChartModel model = new FarPoint.Win.Chart.ChartModel();
model.LabelAreas.Add(labelArea);
model.PlotAreas.Add(plotArea);
chart2DControl1.Model = model;
```

Visual Basic

```
Dim series0 As New FarPoint.Win.Chart.BarSeries()
series0.Values.Add(8.0)
series0.Values.Add(4.0)
series0.Values.Add(2.0)
series0.Values.Add(1.0)
Dim series1 As New FarPoint.Win.Chart.LineSeries()
series1.PointMarker = New
FarPoint.Win.Chart.BuiltinMarker(FarPoint.Win.Chart.MarkerShape.Circle, 7.0F)
series1.Values.Add(8.0)
series1.Values.Add(12.0)
series1.Values.Add(14.0)
series1.Values.Add(15.0)
Dim plotArea As New FarPoint.Win.Chart.YPlotArea()
plotArea.Location = New PointF(0.2F, 0.2F)
plotArea.Size = New SizeF(0.6F, 0.6F)
plotArea.Series.Add(series0)
plotArea.Series.Add(series1)
Dim labelArea As New FarPoint.Win.Chart.LabelArea()
labelArea.Location = New PointF(0.5F, 0.02F)
labelArea.AlignmentX = 0.5F
labelArea.AlignmentY = 0.0F
labelArea.Text = "Pareto Chart"
Dim model As New FarPoint.Win.Chart.ChartModel()
model.LabelAreas.Add(labelArea)
model.PlotAreas.Add(plotArea)
chart2DControl1.Model = model
```

Connecting to Data

The chart control can be bound or unbound. If the control is unbound, provide the values as double values.

When the chart is bound, the values can any data type that can be converted to a double value (including int, double, decimal, string, and so on).

 You can set the data display unit. For example, if your data values are in the millions, you can display the data in millions, with a smaller range of values (100,000,000 is represented by 100). The display unit is set by the **DisplayUnit** ('DisplayUnits Property' in the on-line documentation) property of the **ValueAxis** ('ValueAxis Class' in the on-line documentation) class that represents the value axis.

For more information on adding data to a Chart control, see the following topics:

- **Using a Bound Data Source**
- **Using an UnBound Data Source**
- **Binding with cell range**

Using a Bound Data Source

You can bind the chart to the following data sources:

- Array
- Array List (IList)
- List Collection
- Table

When the chart is bound to data, it dynamically plots the data when it paints. A single chart can support (and display) data from multiple data sources and multiple data fields within a data source.

For information on binding to a data source, see the members of the class that represents the data series of interest (for example, the **BarSeries** ('BarSeries Class' in the on-line documentation) class for bar charts).

Example

The following example demonstrates how to bind a bar chart to array data.

C#

```
// Create an array and bind.
object[] values = new object[] { 2, 4.0, 3.0m, "5.0" };
BarSeries series = new BarSeries();
series.Values.DataSource = values;
```

Visual Basic

```
' Create an array and bind.
Dim values() As Object = {2, 4.0, 3.0D, "5.0"}
Dim series As New BarSeries()
series.Values.DataSource = values
```

The following example demonstrates how to bind a bar chart to table.

C#

```
DataTable dt = new DataTable("Test");
DataRow dr = default(DataRow);
dt.Columns.Add("Series0");
dt.Columns.Add("Series1");
dr = dt.NewRow();
dr[0] = 2;
dr[1] = 1;
dt.Rows.Add(dr);
dr = dt.NewRow();
dr[0] = 4;
```

```
dr[1] = 2;
dt.Rows.Add(dr);
dr = dt.NewRow();
dr[0] = 3;
dr[1] = 4;
FarPoint.Win.Chart.BarSeries series = new FarPoint.Win.Chart.BarSeries();
series.Values.DataSource = dt;
series.Values.DataField = dt.Columns[0].ColumnName;
FarPoint.Win.Chart.YPlotArea plotArea = new FarPoint.Win.Chart.YPlotArea();
FarPoint.Win.Chart.ChartModel model = new FarPoint.Win.Chart.ChartModel();
plotArea.Location = new PointF(0.2F, 0.2F);
plotArea.Size = new SizeF(0.6F, 0.6F);
plotArea.Series.Add(series);
model.PlotAreas.Add(plotArea);
fpChart1.Model = model;
```

Visual Basic

```
Dim dt As New DataTable("Test")
Dim dr As DataRow
dt.Columns.Add("Series0")
dt.Columns.Add("Series1")
dr = dt.NewRow()
dr(0) = 2
dr(1) = 1
dt.Rows.Add(dr)
dr = dt.NewRow()
dr(0) = 4
dr(1) = 2
dt.Rows.Add(dr)
dr = dt.NewRow()
dr(0) = 3
dr(1) = 4
dt.Rows.Add(dr)
Dim series As New FarPoint.Win.Chart.BarSeries
series.Values.DataSource = dt
series.Values.DataField = dt.Columns(0).ColumnName
Dim model As New FarPoint.Win.Chart.ChartModel()
Dim plotArea As New FarPoint.Win.Chart.YPlotArea()
plotArea.Location = New PointF(0.2F, 0.2F)
plotArea.Size = New SizeF(0.6F, 0.6F)
plotArea.Series.Add(series)
model.PlotAreas.Add(plotArea)
FpChart1.Model = model
```

Using an UnBound Data Source

You can use the chart in an unbound state without binding it to the data source. In such case, add double values to the chart control without using a datasource.

Example

The following example demonstrates adding unbound data to the control.

C#

```
BarSeries series = new BarSeries();
series.Values.Add(2.0);
series.Values.Add(4.0);
series.Values.Add(3.0);
series.Values.Add(5.0);
```

Visual Basic

```
Dim series As New BarSeries()
series.Values.Add(2.0)
series.Values.Add(4.0)
series.Values.Add(3.0)
series.Values.Add(5.0)
```

Using Chart Designer

1. Run the **Chart Designer**.
2. Select the target **Chart Model** from the tree menu on the left.
3. Open the **Plot Area Collection Editor** from **Plot Area** in the **Other** section of the property list on the right.
4. Open the **Series Collection Editor** from Series in the **Data** section.
5. Open the **Value Collection Editor** from Values in the **Data** section.
6. Set the values as required.

 For information on starting **Chart Designer ('Spread Designer Guide' in the on-line documentation)**, refer to Chart Designer in the **SPREAD Designer Guide ('Using the Chart Designer' in the on-line documentation)**.

Binding with cell range

A series contains three parts (category, series name, and data). You can bind each part to an instance of the series data field. The entire chart control can not be bound; however, you can use a cell range or a formula to put data in the chart.

You can use an instance of the **SeriesDataField ('SeriesDataField Class' in the on-line documentation)** class to bind each part into a cell ranges.

 In addition to using the SeriesDataField class to bind each series to a range of cells, you can also use **AddChart ('AddChart Method' in the on-line documentation)** method of the **SheetView ('SheetView Class' in the on-line documentation)** class to add a chart bound to a range of cells to the sheet. In such a case, a series is automatically created from the target cell range.

Using the code

Set the SeriesDataField object to the following property of the target series (for example, the **LineSeries ('LineSeries Class' in the on-line documentation)** class for line charts). In the **Formula ('Formula Property' in the on-line documentation)** property of the SeriesDataField object, specify the binding cell ranges or the formula.

- **SeriesNameDataSource ('SeriesNameDataSource Property' in the on-line documentation)** property
Binds series names into a range of cells.
- **DataSource ('DataSource Property' in the on-line documentation)** property of the StringCollection object referenced in the **CategoryNames ('CategoryNames Property' in the on-line documentation)** property
Binds categories into a range of cells.
- **DataSource ('DataSource Property' in the on-line documentation)** property of the DoubleCollection

object referenced in the **Values ('Values Property' in the on-line documentation)** property
 Binds the data into a range of cells.

Example

The following example first uses the AddChart method of the SheetView class in the form's Load event to add a chart bound to a range of cells.

Then in the button's Click event, add data to the cell and bind the chart to the new cell range. Use the SeriesDataField object to bind existing series category names, data, new series name, and data into a range of cells.

C#

```
private void Form1_Load(object sender, EventArgs e)
{
    object[,] values = { { "", "Category-1", "Category-2" }, { "Series-A", 2.0, 5.0 },
    { "Series-B", 4.0, 5.0 } };
    fpSpread1.Sheets[0].SetArray(0, 0, values);
    FarPoint.Win.Spread.Model.CellRange cellRange = new
FarPoint.Win.Spread.Model.CellRange(0, 0, values.GetLength(0), values.GetLength(1));
    fpSpread1.Sheets[0].AddChart(cellRange, typeof(FarPoint.Win.Chart.LineSeries), 400,
400, 100, 100);
}
private void button1_Click(object sender, EventArgs e)
{
    fpSpread1.Sheets[0].SetArray(0, 3, new object[,] { { "Category-3" }, { 4.0 }, { 2.0
} });
    fpSpread1.Sheets[0].SetArray(3, 0, new object[,] { { "Series-C", 3.0, 2.0, 1.0 }
});
    FarPoint.Win.Chart.YPlotArea plotArea =
(FarPoint.Win.Chart.YPlotArea) fpSpread1.Sheets[0].Charts[0].Model.PlotAreas[0];
    FarPoint.Win.Chart.LineSeries series;
    FarPoint.Win.Spread.Chart.SeriesDataField data;
    series = (FarPoint.Win.Chart.LineSeries)plotArea.Series[0];
    series.CategoryNames.DataSource = new
FarPoint.Win.Spread.Chart.SeriesDataField(fpSpread1, "DataFieldCategoryName",
"Sheet1!$B$1:$D$1", FarPoint.Win.Spread.Chart.SegmentDataType.Text);
    data = (FarPoint.Win.Spread.Chart.SeriesDataField)series.Values.DataSource;
    data.Formula = "Sheet1!$B$2:$D$2";
    series = (FarPoint.Win.Chart.LineSeries)plotArea.Series[1];
    data = (FarPoint.Win.Spread.Chart.SeriesDataField)series.Values.DataSource;
    data.Formula = "Sheet1!$B$3:$D$3";
    series = new FarPoint.Win.Chart.LineSeries();
    series.SeriesNameDataSource = new
FarPoint.Win.Spread.Chart.SeriesDataField(fpSpread1, "DataFieldSeriesName",
"Sheet1!$A$4:$A$4", FarPoint.Win.Spread.Chart.SegmentDataType.Text);
    series.Values.DataSource = new FarPoint.Win.Spread.Chart.SeriesDataField(fpSpread1,
"DataFieldValue", "Sheet1!$B$4:$D$4");
    plotArea.Series.Add(series);
}
}
```

Visual Basic

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
MyBase.Load
    Dim values As Object(,) = {{"", "Category-1", "Category-2"}, {"Series-A", 2.0,
5.0}, {"Series-B", 4.0, 5.0}}
    FpSpread1.Sheets(0).SetArray(0, 0, values)
End Sub
```

```

    Dim cellRange As New FarPoint.Win.Spread.Model.CellRange(0, 0, values.GetLength(0),
values.GetLength(1))
    FpSpread1.Sheets(0).AddChart(cellRange, GetType(FarPoint.Win.Chart.LineSeries),
400, 400, 100, 100)
End Sub
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    FpSpread1.Sheets(0).SetArray(0, 3, New Object(,) {"Category-3"}, {4.0}, {2.0})
    FpSpread1.Sheets(0).SetArray(3, 0, New Object(,) {"Series-C", 3.0, 2.0, 1.0})
    Dim plotArea As FarPoint.Win.Chart.YPlotArea =
DirectCast(FpSpread1.Sheets(0).Charts(0).Model.PlotAreas(0),
FarPoint.Win.Chart.YPlotArea)
    Dim series As FarPoint.Win.Chart.LineSeries
    Dim data As FarPoint.Win.Spread.Chart.SeriesDataField
    series = DirectCast(plotArea.Series(0), FarPoint.Win.Chart.LineSeries)
    series.CategoryNames.DataSource = New
FarPoint.Win.Spread.Chart.SeriesDataField(FpSpread1, "DataFieldCategoryName",
"Sheet1!$B$1:$D$1", FarPoint.Win.Spread.Chart.SegmentDataType.Text)
    data = DirectCast(series.Values.DataSource,
FarPoint.Win.Spread.Chart.SeriesDataField)
    data.Formula = "Sheet1!$B$2:$D$2"
    series = DirectCast(plotArea.Series(1), FarPoint.Win.Chart.LineSeries)
    data = DirectCast(series.Values.DataSource,
FarPoint.Win.Spread.Chart.SeriesDataField)
    data.Formula = "Sheet1!$B$3:$D$3"
    series = New FarPoint.Win.Chart.LineSeries()
    series.SeriesNameDataSource = New
FarPoint.Win.Spread.Chart.SeriesDataField(FpSpread1, "DataFieldSeriesName",
"Sheet1!$A$4:$A$4", FarPoint.Win.Spread.Chart.SegmentDataType.Text)
    series.Values.DataSource = New FarPoint.Win.Spread.Chart.SeriesDataField(FpSpread1,
"DataFieldValue", "Sheet1!$B$4:$D$4")
    plotArea.Series.Add(series)
End Sub

```

The following sample code binds a control to a data table. The data in the data table is set in the cell. A chart using these data is added by the AddChart method of the SheetView class.

C#

```

DataTable dt = new DataTable("Test");
DataRow dr = default(DataRow);
dt.Columns.Add("Series1");
dt.Columns.Add("Series2");
dr = dt.NewRow();
dr[0] = 1;
dr[1] = 4;
dt.Rows.Add(dr);
dr = dt.NewRow();
dr[0] = 2;
dr[1] = 5;
dt.Rows.Add(dr);
dr = dt.NewRow();
dr[0] = 3;
dr[1] = 6;
dt.Rows.Add(dr);
fpSpread1.DataSource = dt;
FarPoint.Win.Spread.Model.CellRange cellRange = new
FarPoint.Win.Spread.Model.CellRange(0, 0, 3, 3);

```

```
fpSpread1.Sheets[0].AddChart(cellRange, typeof(FarPoint.Win.Chart.PointSeries), 400, 400, 0, 0);
```

Visual Basic

```
Dim dt As New DataTable("Test")
Dim dr As DataRow
dt.Columns.Add("Series1")
dt.Columns.Add("Series2")
dr = dt.NewRow()
dr(0) = 1
dr(1) = 4
dt.Rows.Add(dr)
dr = dt.NewRow()
dr(0) = 2
dr(1) = 5
dt.Rows.Add(dr)
dr = dt.NewRow()
dr(0) = 3
dr(1) = 6
dt.Rows.Add(dr)
FpSpread1.DataSource = dt
Dim cellRange As New FarPoint.Win.Spread.Model.CellRange(0, 0, 3, 3)
FpSpread1.Sheets(0).AddChart(cellRange, GetType(FarPoint.Win.Chart.PointSeries), 400, 400, 0, 0)
```

Advanced chart settings

You can set the chart in the following ways:

- **Fill Effects**
- **View Type**

Fill Effects

A fill effect is when the interior of an object is painted. Two types of fill effects are solid and gradient. A solid fill effect uses a single color and a gradient fill uses two colors and a direction.

The following fill effects are available in the **Fill ('Fill Class' in the on-line documentation)** class:

- No fill - Uses **NoFill ('NoFill Class' in the on-line documentation)** class.
- Solid Color Fill - Uses **SolidFill ('SolidFill Class' in the on-line documentation)** class.
- Gradient fill - Uses **GradientFill ('GradientFill Class' in the on-line documentation)** class.

You can fill elements using the Fill property in the following classes:

- Labels (**LabelArea ('LabelArea Class' in the on-line documentation)** object)
- Legends (**LegendArea ('LegendArea Class' in the on-line documentation)** object)
- Walls (**Wall ('Wall Class' in the on-line documentation)** object)
- Stripes (**Stripe ('Stripe Class' in the on-line documentation)** object)
- Chart itself (**ChartModel ('ChartModel Class' in the on-line documentation)** object)



To set the fill effect for the entire plot area, you can set the Fill property of the wall for the plot area. For example, for a y-plot, use the **BackWall ('BackWall Property' in the on-line documentation)** property of the

YPlotArea ('YPlotArea Class' in the on-line documentation) class to reference the **Wall ('Wall Class' in the on-line documentation)** object and set the **Fill ('Fill Property' in the on-line documentation)** property.

Example

The following example sets a fill effect for a bar chart. You can also set the fill effect before or after adding the data points if you set the fill effect for the entire series.

C#

```
FarPoint.Win.Chart.BarSeries series = new FarPoint.Win.Chart.BarSeries();
series.BarFill = new FarPoint.Win.Chart.SolidFill(Color.Red);
series.Values.Add(2.0);
series.Values.Add(4.0);
series.Values.Add(3.0);
series.Values.Add(5.0);
FarPoint.Win.Chart.YPlotArea plotArea = new FarPoint.Win.Chart.YPlotArea();
plotArea.Location = new PointF(0.2f, 0.2f);
plotArea.Size = new SizeF(0.6f, 0.6f);
plotArea.Series.Add(series);
FarPoint.Win.Chart.ChartModel model = new FarPoint.Win.Chart.ChartModel();
model.PlotAreas.Add(plotArea);
fpChart1.Model = model;
```

Visual Basic

```
Dim series As New FarPoint.Win.Chart.BarSeries()
series.BarFill = New FarPoint.Win.Chart.SolidFill(Color.Red)
series.Values.Add(2.0)
series.Values.Add(4.0)
series.Values.Add(3.0)
series.Values.Add(5.0)
Dim plotArea As New FarPoint.Win.Chart.YPlotArea()
plotArea.Location = New PointF(0.2F, 0.2F)
plotArea.Size = New SizeF(0.6F, 0.6F)
plotArea.Series.Add(series)
Dim model As New FarPoint.Win.Chart.ChartModel()
model.PlotAreas.Add(plotArea)
fpChart1.Model = model
```

If you set the fill effect for a single data point, then you need to assign the fill effect after you add the data point, so there is a data point to store the fill effect in. For example:

C#

```
FarPoint.Win.Chart.BarSeries series = new FarPoint.Win.Chart.BarSeries();
series.Values.Add(2.0);
series.Values.Add(4.0);
series.BarFills.Add(new SolidFill(Color.Green));
```

Visual Basic

```
Dim series As New FarPoint.Win.Chart.BarSeries()
series.Values.Add(2.0)
series.Values.Add(4.0)
series.BarFills.Add(New SolidFill(Color.Green))
```

You can assign fill effects for lines and markers as well. For example:

C#

```
FarPoint.Win.Chart.PointSeries series = new FarPoint.Win.Chart.PointSeries();
series.PointFill = new FarPoint.Win.Chart.SolidFill(Color.Lime);
series.PointBorder = new FarPoint.Win.Chart.SolidLine(Color.Red);
series.PointMarker = new
FarPoint.Win.Chart.BuiltinMarker(FarPoint.Win.Chart.MarkerShape.Triangle, 10.0f);
series.Values.Add(2.0);
series.Values.Add(4.0);
series.Values.Add(3.0);
series.Values.Add(5.0);
```

Visual Basic

```
Dim series As New FarPoint.Win.Chart.PointSeries()
series.PointFill = New FarPoint.Win.Chart.SolidFill(Color.Lime)
series.PointBorder = New FarPoint.Win.Chart.SolidLine(Color.Red)
series.PointMarker = New
FarPoint.Win.Chart.BuiltinMarker(FarPoint.Win.Chart.MarkerShape.Triangle, 10.0F)
series.Values.Add(2.0)
series.Values.Add(4.0)
series.Values.Add(3.0)
series.Values.Add(5.0)
```

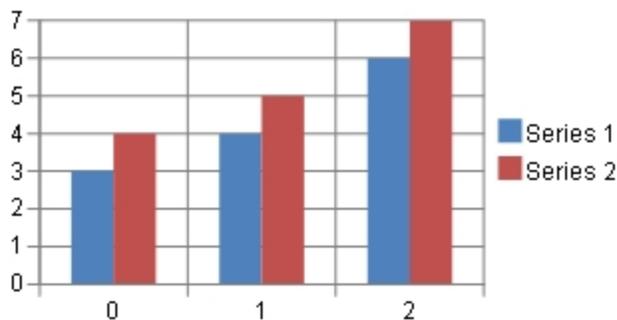
Using the Chart Designer

1. Run the **Chart Designer**.
2. Expand the target **Plot Area** from the tree menu on the left.
3. Select **Bar Chart Series** and set **BarFill** or **BarFills** in the **Appearance** section in the property list on the right.
4. Click OK and close the **Chart Designer**.

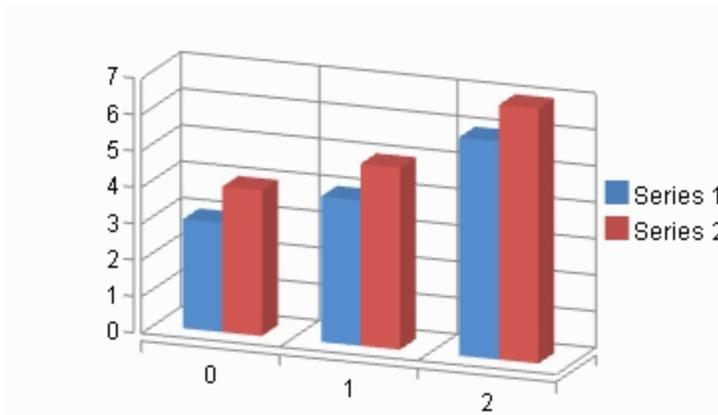
 For information on starting **Chart Designer ('Spread Designer Guide' in the on-line documentation)**, refer to Chart Designer in the **SPREAD Designer Guide ('Using the Chart Designer' in the on-line documentation)**.

Display format

The chart display format can be specified in 2D or 3D. The following image shows a 2D chart.



The following image shows an example of a 3D chart.



Using code

Set the **ViewType** (**'ViewType Property' in the on-line documentation**) property of the **SpreadChart** (**'SpreadChart Class' in the on-line documentation**) class that represents the chart. Also, you can rotate the plot area around the horizontal and vertical axes as required. You can set the rotation angle around the horizontal axis with the **Elevation** (**'Elevation Property' in the on-line documentation**) property of the **PlotArea** (**'PlotArea Class' in the on-line documentation**) class and the vertical axis with the **Rotation** (**'Rotation Property' in the on-line documentation**) property.

Example

The following example demonstrate how to set the chart display format to 3D.

C#

```
FarPoint.Win.Spread.Chart.SpreadChart chart = new  
FarPoint.Win.Spread.Chart.SpreadChart();  
chart.ViewType = FarPoint.Win.Chart.ChartViewType.View3D;
```

Visual Basic

```
Dim chart As New FarPoint.Win.Spread.Chart.SpreadChart()  
chart.ViewType = FarPoint.Win.Chart.ChartViewType.View3D
```

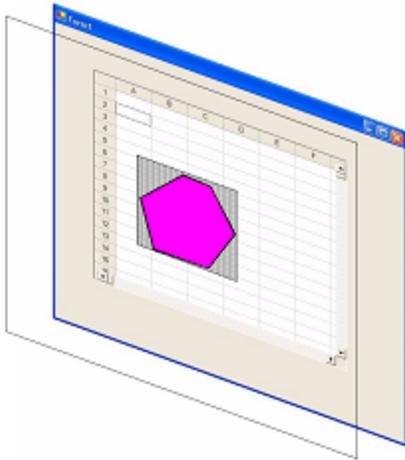
Using the Chart Designer

1. On the Chart Designer, right-click on the chart.
2. Select [3D Display] or [2D Display] (the displayed items differ depending on the current display format) from the context menu.

 You can also select a chart on the SPREAD designer and set the ViewType from the **View** section of the property list on the right side of the screen.

Customizing Drawing

Each sheet can have its own drawing layer that can contain built-in shapes, custom shapes, and annotations (free-hand drawings). Shapes and annotations are a form of graphics that are drawn on a separate layer from that of the spreadsheet. This drawing layer, or drawing space, is in front of the spreadsheet in the display. Shapes can be made all or partially transparent to reveal the spreadsheet behind. An example of a multiple-sided shape drawn in the space above a spreadsheet is shown in this figure to help you understand the concept of layers. Because the shapes appear on this separate layer from the sheet and can be thought to float above the spreadsheet, they are sometimes called floating objects.



These topics can help you customize the interaction with the drawing layer:

- **Working with Shapes in Code**
- **Creating Camera Shapes**

Note:

- The shapes are available in code (each shape being a separate class in the **DrawingSpace** namespace) or from the **Insert** menu in the Spread Designer.
- You could customize aspects of the shape from size and background color to rotation angle or gradient.
- You can use shapes and annotation to draw attention to parts of your spreadsheet or emphasize some information or process involving the use of the spreadsheet. For example, you can display a logo on your sheet, show a process with flowchart-like graphics, or use shapes to simply highlight a particular result.

For instance, in a spreadsheet you could create a shape like a "star" or other graphic that could highlight data or point the user to some aspect of working with the sheet and place this on a layer independent of the cells and their values. You could then proceed to customize aspects of that star or graphic from size and background color to rotation angle or gradient. The shapes are available in code (each shape being a separate class in the **DrawingSpace** namespace) or from the **Insert** menu in the Spread Designer. You can use shapes and annotation to draw attention to parts of your spreadsheet or emphasize some information or process involving the use of the spreadsheet. For example, you can display a logo on your sheet, show a process with flowchart-like graphics, or use shapes to simply highlight a particular result.

Working with Shapes in Code

There are several built-in shapes for you to use on a sheet. Each shape can be rotated and resized, and their ability to be rotated and resized by the end user can be constrained. When selected, the shape has resize handles with which you can adjust the size and a rotate handle with which you can rotate the shape.

Colors, shadows, and transparency can be adjusted. Most users find it easy to create and place the shapes using Spread Designer.

You may also create and place shapes using code. Besides working with shapes from Spread Designer, you can also add and remove shapes programmatically. You can perform the following work with shapes using the corresponding methods in the **SheetView ('SheetView Class' in the on-line documentation)** class:

- Add a shape in code using the **AddShape ('AddShape Method' in the on-line documentation)** method
- Remove a shape using the **RemoveShape ('RemoveShape Method' in the on-line documentation)** method
- Remove all the shapes using the **ClearShapes ('ClearShapes Method' in the on-line documentation)** method
- Get a shape using the **GetShape ('GetShape Method' in the on-line documentation)** method

To add a shape using code, refer to the **DrawingSpace** namespace and select the particular shape and define its properties using code. While there is much flexibility in setting up shapes, there are some limitations. These are listed in the **Assembly Reference (on-line documentation)** in the topics for the shape methods and classes. For example, for the **LineShape ('LineShape Class' in the on-line documentation)** class, the maximum thickness for a line is 64 pixels.

Example

This example constructs a basic rectangle shape with default properties and adds the shape the active sheet.

C#

```
fpSpread1.ActiveSheet.AddShape(new FarPoint.Win.Spread.DrawingSpace.RectangleShape());
```

VB

```
FpSpread1.ActiveSheet.AddShape(New FarPoint.Win.Spread.DrawingSpace.RectangleShape())
```

Example

The following example creates a shape with custom settings.

C#

```
// Create a new shape.
FarPoint.Win.Spread.DrawingSpace.RectangleShape rShape = new
FarPoint.Win.Spread.DrawingSpace.RectangleShape();
// Assign a name, overriding the unique default assigned name.
// All shape names within a sheet must be unique.
rShape.Name = "rShape1";
// Assign a location at which to start the display of the shape.
rShape.Top = 20;
rShape.Left = 60;
// Alternatively, you could set the Location property
// with a Point object as in:
// rShape.Location = new Point(20, 60);

// Assign a custom fill color to the shape.
rShape.BackColor = Color.Blue;
// Assign a size to the shape.
rShape.Width = 100;
rShape.Height = 100;
// Alternatively, you could set the Size property
// with a Size object as in:
```

```
// rShape.Size = new Size(100, 100);
// Add the shape to the sheet so that it appears on that sheet.
fpSpread1.ActiveSheet.AddShape(rShape);
```

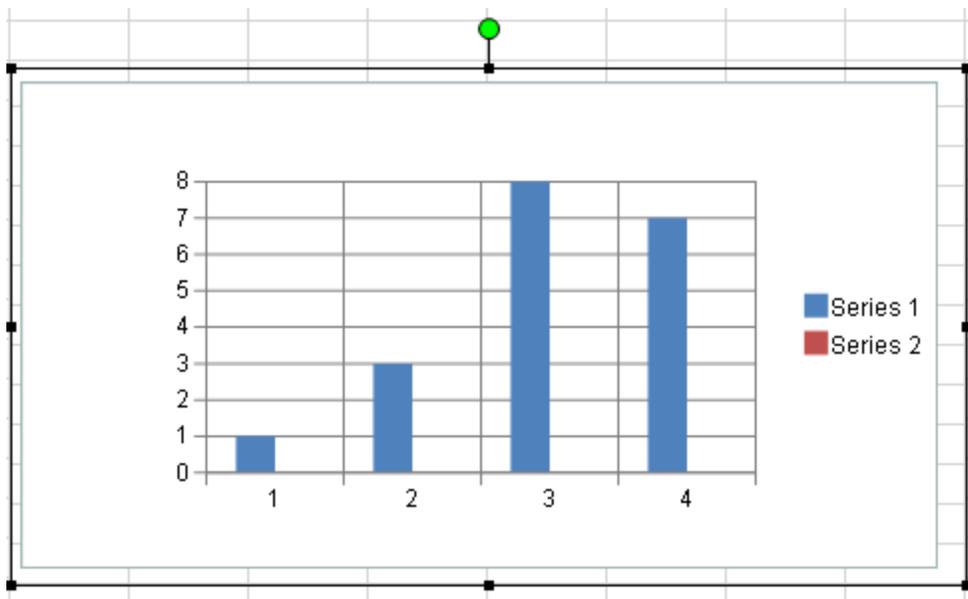
VB

```
' Create a shape.
Dim rShape As New FarPoint.Win.Spread.DrawingSpace.RectangleShape()
' Assign a name, overriding the unique default assigned name.
' All shape names within a sheet must be unique.
rShape.Name = "rShape1"
' Assign a location at which to start the display of the shape.
rShape.Top = 20
rShape.Left = 60
' Alternatively, you could set the Location property
' with a Point object as in:
' rShape.Location = New Point(20, 60)

' Assign a custom fill color to the shape.
rShape.BackColor = Color.Blue
' Assign a size to the shape.
rShape.Width = 100
rShape.Height = 100
' Alternatively, you could set the Size property
' with a Size object as in:
' rShape.Size = New Size(100, 100)
' Add the shape to the sheet so that it appears on that sheet.
FpSpread1.ActiveSheet.AddShape(rShape)
```

Creating Camera Shapes

You can create a snapshot of a range of cells and use that as a shape in the Spread control. The cell range can contain other shapes including charts. The following image displays a camera shape that contains a chart.



For more information about using the Spread Designer to add camera shapes, see the **Insert Menu (on-line**

documentation) topic. In general, properties that apply to the interior of the shape, do not apply to the camera shape.

 The camera shape cannot use another camera shape.

Using Code

1. Create a camera shape object by using the **SpreadCameraShape** ('**SpreadCameraShape Class**' in the **online documentation**) class.
2. Specify the range of cells that will become the shape with the **Formula** property.
3. Set any other shape properties.
4. Add the camera shape to the sheet.

Example

This example creates a blue triangle, adds text to a cell, and creates a camera shape that includes both.

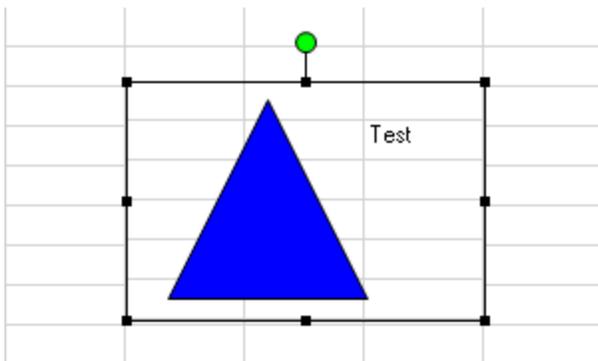
C#

```
fpSpread1.Sheets[0].Cells[1, 3].Text = "Test";
FarPoint.Win.Spread.DrawingSpace.TriangleShape a = new
FarPoint.Win.Spread.DrawingSpace.TriangleShape ();
a.BackColor = Color.Blue;
fpSpread1.ActiveSheet.AddShape(a, 1, 1);
FarPoint.Win.Spread.DrawingSpace.SpreadCameraShape test = new
FarPoint.Win.Spread.DrawingSpace.SpreadCameraShape ();
test.Formula = "B1:D6";
test.Location = new System.Drawing.Point(20, 20);
fpSpread1.Sheets[0].AddShape(test);
```

VB

```
fpSpread1.Sheets(0).Cells(1, 3).Text = "Test"
Dim a As New FarPoint.Win.Spread.DrawingSpace.TriangleShape
a.BackColor = Color.Blue
fpSpread1.ActiveSheet.AddShape(a, 1, 1)
Dim test As New FarPoint.Win.Spread.DrawingSpace.SpreadCameraShape()
test.Formula = "B1:D6"
test.Location = New System.Drawing.Point(20, 20)
fpSpread1.Sheets(0).AddShape(test)
```

The following image displays a camera shape that contains a triangle shape and a cell with text.

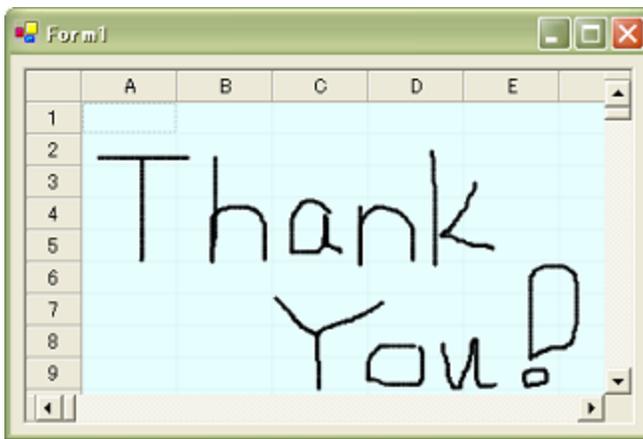


Using the Spread Designer

1. Select a block of cells in the designer.
2. Select the **Insert** menu.
3. Select the camera shape icon.
4. Click on the shape to move it.
5. The **Drawing Tools** menu with additional options is displayed.
6. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Allowing the User to Draw with a Tablet PC

Spread provides a limited set of functionality in support of "inking" on Tablet PCs that are enabled with Microsoft's Tablet PC SDK. Microsoft Windows XP Tablet PC Edition is a superset of the Windows XP Professional operating system that adds pen-based capabilities to full notebook computers. This feature in Spread is called "ink notation". In this version inking on a given viewport of the spreadsheet is allowed.



The drawing feature allows the use of a pen (stylus) on the Tablet PC for drawing and writing on the spreadsheet. This process, sometimes called inking, allows users to write in digital ink, which appears as natural-looking handwriting on the screen. The spreadsheet saves the ink notation.

Getting Set Up

Before you can use this feature, you must perform these steps:

1. Go to the Microsoft site and download the latest version Tablet PC SDK.
2. Make sure the Ink assembly is in the GAC before running the **StartInkNotation** method in Spread.

Using Ink Notation

To use ink notation, use one of the `FpSpread.StartInkNotation` (**'StartInkNotation Method' in the on-line documentation**) methods. There are various overloads to allow you to specify the viewport, alpha-blending, and background color of the ink viewport, as follows:

- The default setting that specifies the active viewport as the viewport that can be inked in.
- You specify the viewport that will allow inking.
- You can specify the background color and alpha-blending (transparency) to allow you to highlight the inking viewport and see the spreadsheet underneath.

When you call the **StartInkNotation** method, Spread checks to see if the Ink assembly is in the GAC. If it cannot find it, the method returns `False` and goes no further. If it is there, it loads it and then it loads the `FarPoint.Ink` assembly (which is installed with Spread, but is not loaded until now). Spread attempts to load `FpInk` dynamically at run time, and

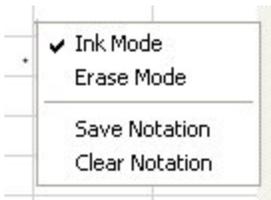
it will succeed if FpInk is in the GAC, along with Microsoft.Ink (the MS Ink assembly). Microsoft.Ink is part of the Microsoft Tablet PC SDK.

When inking, the cursor changes to a drawing point; you can draw with the stylus. You finish drawing by clicking the right mouse button to call up the context menu. Click **Save Notation** to save the notation, the inking that you have done, to the Spread component. The notation is saved as a "shape" residing over a part of the spreadsheet. Inking is done on the drawing/presentation space layer, so similar to shapes, the inking is drawn on top of the spreadsheet. It is drawn only in the viewport, not over the headers or scroll bars or other parts of the Spread component. It appears as a shape, so it can be selected, moved, rotated, and resized. When it is saved, it is given a name beginning with "inkShape" and a unique number.

When you want to erase what you have done, right-click to get the context menu and select **Erase Mode**, and the stylus becomes an eraser, or more accurately a selector of what gets erased. If you click on a stroke or part of an ink notation, it erases that segment. If you write in cursive and all the notation is continuous, then if selected, the entire notation is erased. The context menu choice, **Clear Notation**, clears the entire notation, regardless of the number of discontinuous strokes.

In ink notation, the cursor changes to a pen point with the thickness and color (black is the default) of the pen.

When the user right-clicks on the mouse, a context menu is displayed as shown in the following figure.



The **Ink Mode** is displayed checked while the user is inking (or writing or drawing). In **Erase Mode**, the user is erasing the inking previously done, which erases the segment or stroke that is clicked when in **Erase Mode**. With **Save Notation**, the user accepts the ink notation and it is permanently displayed on the sheet. The **Clear Notation** clears all of the ink notation drawn since the last save.

Working With Shapes (Enhanced Shape Engine)

Spread for Winforms provides support for adding and customizing shapes and group shapes in the worksheets. Shapes are compatible with Excel and with all built-in shapes. Users can copy shapes from Excel to Spread and vice versa.

Enable Shape Engine

The new shape engine is not enabled by default. Users must use the following code to manually turn it on before working with shapes in the spreadsheets:

```
C#  
// Enable the shape engine  
fpSpread1.Features.EnhancedShapeEngine = true;
```

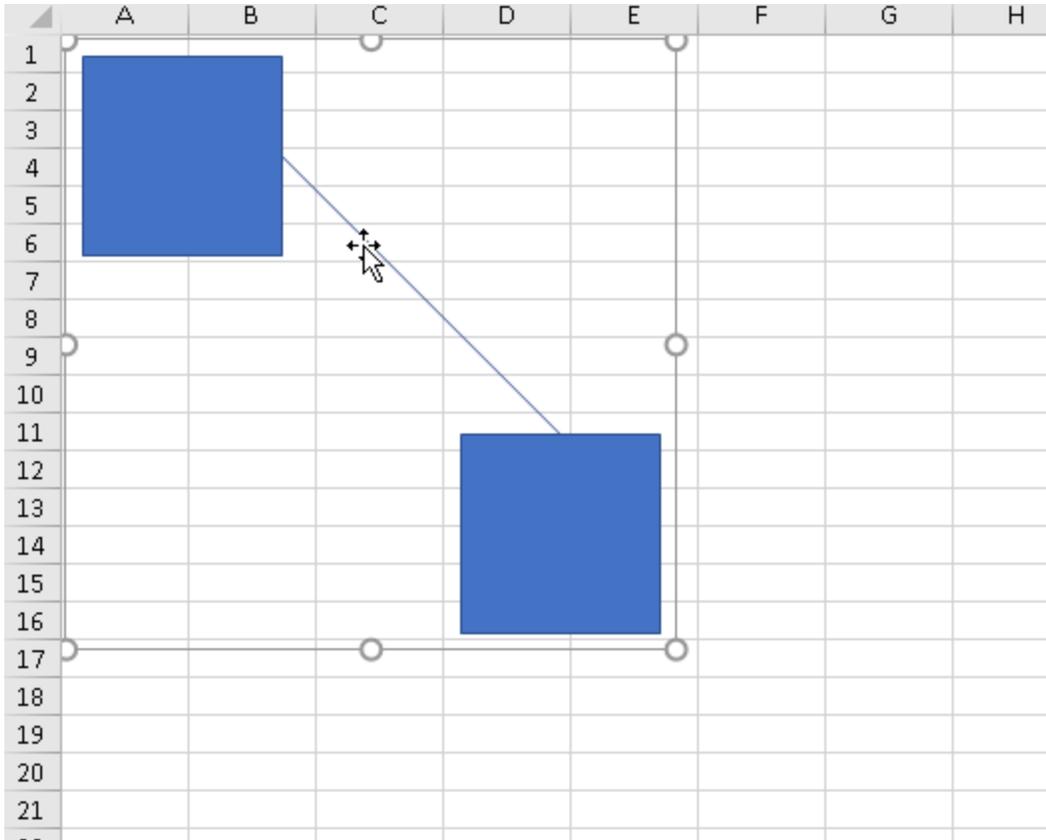
Grouping and Ungrouping Shapes

Users can group or ungroup shapes and move or rotate the grouped shapes just like in Excel by using the **Group()** (**'Group Method' in the on-line documentation**) method of the **IShapeRange** (**'IShapeRange Interface' in the on-line documentation**) interface and the **Ungroup()** (**'Ungroup Method' in the on-line documentation**) method of the **IShapeBase** (**'IShapeBase Interface' in the on-line documentation**) interface.

Group shapes are generated when multiple drawing objects are clustered together in a spreadsheet. It is beneficial to use group shapes in a scenario wherein the spreadsheet contains multiple shapes and users need to optimize the entire execution of similar kind of tasks executed on those shapes (like adding similar kind of style to two or more shapes,

rotating or moving them together). This will ensure that the desired consistency is maintained in all the shapes. While working with shapes in Spread WinForms Designer, users can use the ribbon bar to change the text settings for the shape.

Example - For instance, let's say you want to rotate or move two or more shapes inserted in a worksheet. One way to do this would be to rotate each shape individually by setting its rotation angle. However, this task can become too cumbersome and time consuming to handle manually especially when there are multiple shapes in the worksheet. Instead, you can simply group two or more shapes in the worksheet and rotate the grouped shape as depicted in the image shared below.



The following example code shows how to add multiple shapes and group/ungroup the shapes embedded in the spreadsheet.

C#

```
// Add Multiple Shapes in activeSheet using AddShape() method
fpSpread1.AsWorkbook().ActiveSheet.Shapes.AddShape(AutoShapeType.Rectangle, 10, 10,
100, 100);
fpSpread1.AsWorkbook().ActiveSheet.Shapes.AddConnector(ConnectorType.Straight, 110,
100, 200, 200);
fpSpread1.AsWorkbook().ActiveSheet.Shapes.AddShape(AutoShapeType.Rectangle, 200, 200,
100, 100);

// Group Shapes using Group() method.
IGroupShape groupShape = fpSpread1.AsWorkbook().ActiveSheet.Shapes.Range(new int[] { 0,
1, 2 }).Group();

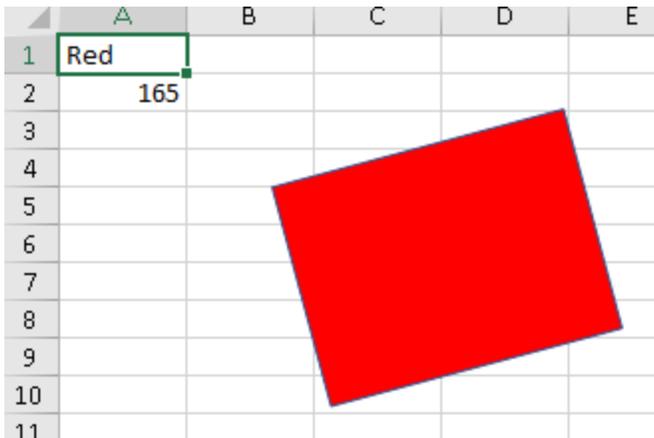
// Ungroup Shapes using Ungroup() method.
```

```
groupShape.Ungroup();
```

Binding Shape Properties

Users can bind the shape properties to the cell and set property bindings to add additional functionalities to the shapes.

For example - In the following image, a rectangular shape has been added in the worksheet and then the binding operation has been done on the shape to change its color (defined in cell A1) and value (defined in cell A2) at runtime as shown in the image shared below.



The following example code binds the shapes in the spreadsheet with different properties.

C#

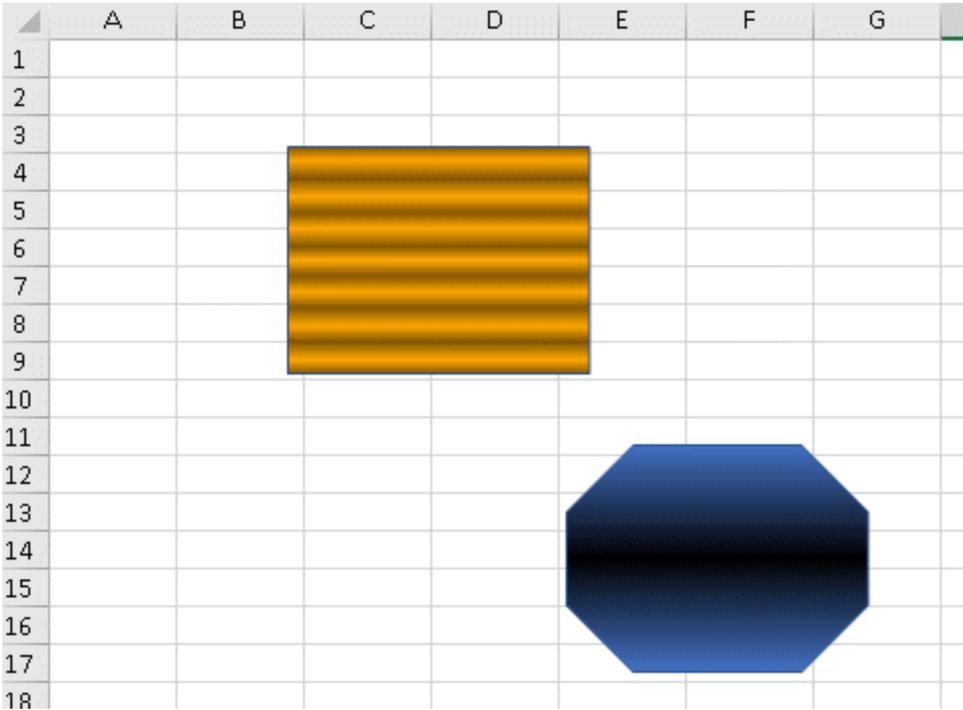
```
// Binding Shapes with different properties
// Set cell values
fpSpread1.ActiveSheet.Cells[0, 0].Value = "Red";
fpSpread1.ActiveSheet.Cells[1, 0].Value = 165;

// Add a rectangle shape
var rectangle =
fpSpread1.AsWorkbook().ActiveSheet.Shapes.AddShape(AutoShapeType.Rectangle, 118.5,
53.25, 151.5, 114);
// Binding shape properties with cell A1 & A2 values
// If cell A1 & A2 values are changed at runtime then it is reflected back in shape too
rectangle.Bindings.Add(nameof(IShape.Fill), "A1");
rectangle.Bindings.Add("Rotation", "A2");
```

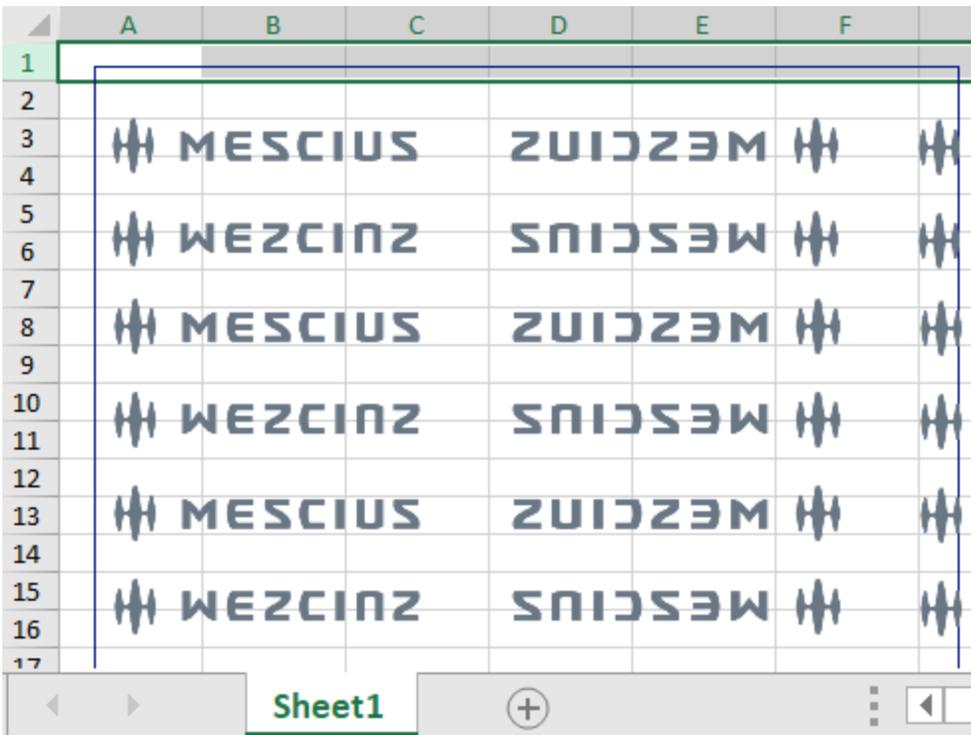
Shape Fills and Effects

While working with shapes in spreadsheets, users can also customize the fill format (solid fill, pattern fill, gradient fill, and texture fill etc.), modify the shape lines (solid and gradient lines), shape text and shape effects (inner shadow, outer shadow, glow, soft edge, reflection) along with a lot of other predefined settings.

The following image depicts two shapes - one rectangle and other octagon with different gradient settings applied to them.



The following image depicts shape texture with image.



The following code shows how to customize shape fill in the spreadsheet.

```
C#
// Add a rectangle shape
```

```

var rectangle =
fpSpread1.AsWorkbook().ActiveSheet.Shapes.AddShape(AutoShapeType.Rectangle, 118.5,
53.25, 151.5, 114);

// Set shapes PresetGradient with GradientStyle
rectangle.Fill.PresetGradient(GradientStyle.Horizontal, 3, 0);

// Add an Octagon shape
var Octagon = fpSpread1.AsWorkbook().ActiveSheet.Shapes.AddShape(AutoShapeType.Octagon,
258.5, 203.25, 151.5, 114);

// Set shapes OneColorGradient with GradientStyle
Octagon.Fill.OneColorGradient(GradientStyle.Horizontal, 3, 0);

// Set Shape's texture with Image
shapes.Fill.UserTextured(@"..\..\Mescius_Logo.png");
shapes.Fill.TextureAlignment = GrapeCity.Drawing.RectAlignment.TopLeft;

```

 **Note:** The following points must be kept in mind while working with shapes in Spread for Winforms:

- The border color of the shape is darker than the border color in Excel.
- The preview row feature is not supported while adding shapes to the spreadsheets.
- The text in the shape may appear blurred as compared to the text in normal cell.
- When the PresetShadow() method is used, applying the blur effect will not work.
- The Shape feature doesn't support undo action, drag, drop and move operations on the rows and columns (i.e. SheetView.MoveColumn(0, 1, true) is not supported).
- Adding 3-D shapes is not supported.
- While importing a shape, it is important to ensure that the data is imported properly but the Preset type is not kept like PresetShadowType or ShapeStyle etc.
- Shapes can't be added when width/height = 0.
- The shape painting style of connector points in group shapes will be different in Excel when shapes are selected. This is a bug of Excel.
- Working with old shape engine - The new shape engine is different from the old shape engine. Using both of them at the same time may cause some unexpected behavior (for instance - while moving/resizing the old shape, the display order of shapes may be changed. This issue can be seen when worksheet contains both the new shape and chart/camera shape (that are still using the old shape engine). It's the limitation in V13 and this issue will be resolved after the Chart/Camera Shape feature is moved to the new shape engine.
- Painting shapes of size larger than 8K for each dimension (for e.g.- 7680 × 7680) is not supported. IColorFormat.ARGB may return different values than the assigned value because the alpha channel is stored as percentage in the range 0....to 100.

Shape Text Orientation

Users can stack text vertically within a shape using the **HorizontalRotatedFarEast** property of **TextOrientation** (**'TextOrientation Enumeration' in the on-line documentation**) enumeration. With this API, you can control the orientation and formatting of text within the shapes.

The following example code shows how to change text orientation in shape.

C#

```

// Add support paint stacked text direction for shape
fpSpread1.Features.EnhancedShapeEngine = true;

```

```

var TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;
IShape shape = TestActiveSheet.Shapes.AddShape(AutoShapeType.Rectangle, 20, 20, 200,
200);
shape.TextFrame.TextRange.Text = "Horizontal Rotated\n Far East";
shape.TextFrame.WordWrap = true;
shape.TextFrame.Orientation =
GrapeCity.Spreadsheet.Drawing.TextOrientation.HorizontalRotatedFarEast;
IShape shape2 = TestActiveSheet.Shapes.AddShape(AutoShapeType.Rectangle, 225, 20, 200,
200);
shape2.TextFrame.TextRange.Text = "Support for vertical text in comment";
shape2.TextFrame.WordWrap = true;
shape2.TextFrame.Orientation =
GrapeCity.Spreadsheet.Drawing.TextOrientation.HorizontalRotatedFarEast;

```

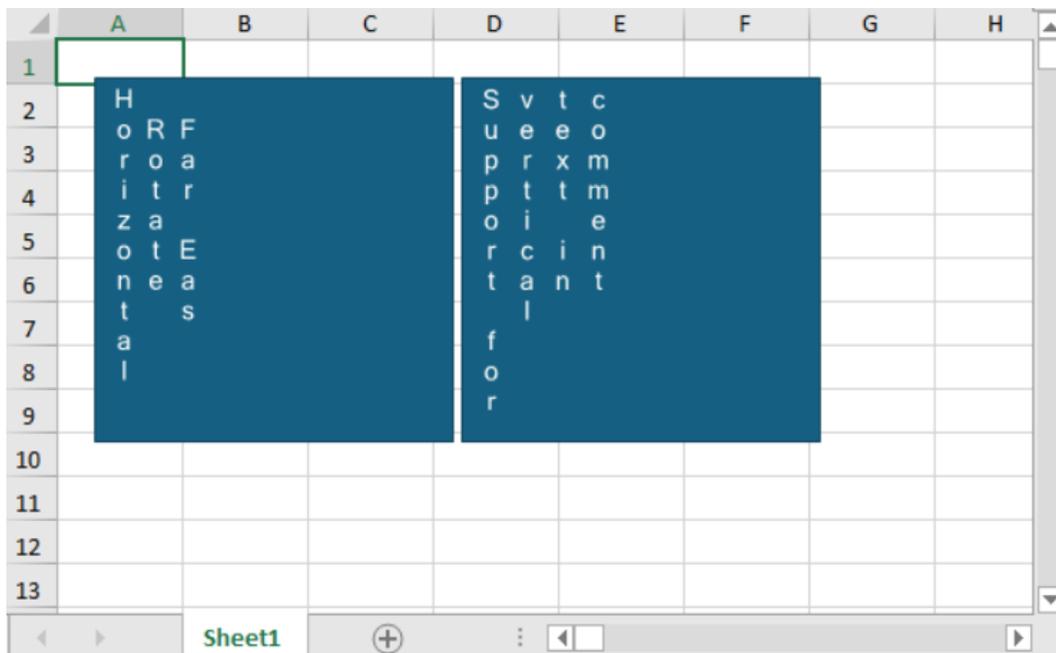
VB

```

' Add support paint stacked text direction for shape
fpSpread1.Features.EnhancedShapeEngine = True
Dim TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet
Dim shape As IShape = TestActiveSheet.Shapes.AddShape(AutoShapeType.Rectangle, 20, 20,
200, 200)
shape.TextFrame.TextRange.Text = "Horizontal Rotated" & vbLf & " Far East"
shape.TextFrame.WordWrap = True
shape.TextFrame.Orientation =
GrapeCity.Spreadsheet.Drawing.TextOrientation.HorizontalRotatedFarEast
Dim shape2 As IShape = TestActiveSheet.Shapes.AddShape(AutoShapeType.Rectangle, 225,
20, 200, 200)
shape2.TextFrame.TextRange.Text = "Support for vertical text in comment"
shape2.TextFrame.WordWrap = True
shape2.TextFrame.Orientation =
GrapeCity.Spreadsheet.Drawing.TextOrientation.HorizontalRotatedFarEast

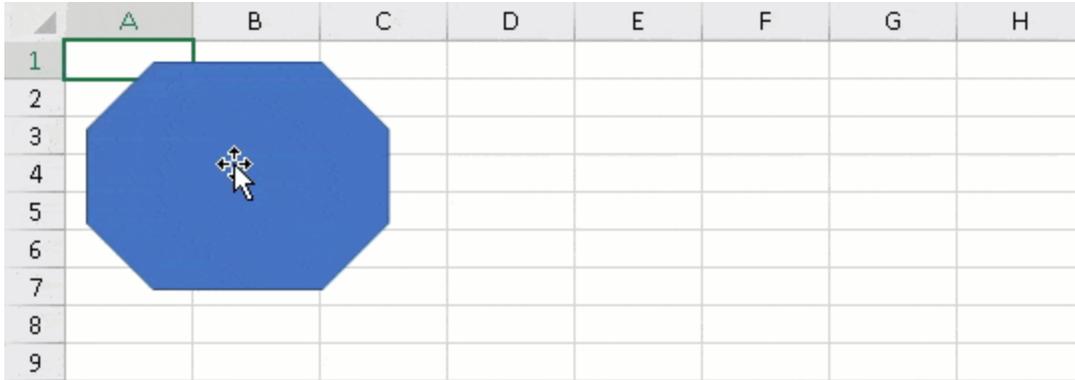
```

The image below depicts vertically oriented text in a shape.

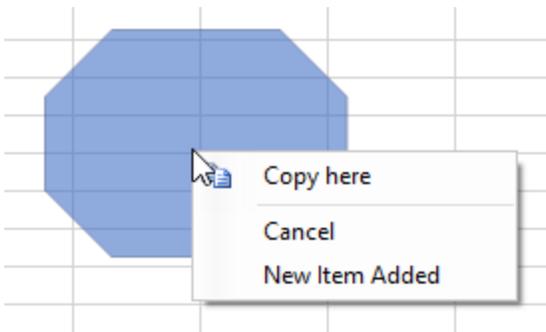


Drag and Copy Shapes

Users have the ability to move or copy a shape by holding the right-mouse button and dragging shapes to the target location. A context-menu pops up after releasing the mouse button and shows the option to move or copy the shape.



You can also customize the above context menu by accessing it through the **ContextMenuType.ShapeDrag** (**'ContextMenuType Enumeration' in the on-line documentation**) enumeration option. The following image illustrates a customized context menu where the "Move" option is hidden and a new item has been added.



The following code shows how to access and customize the shape drag context menu.

C#

```
private void ShapeContextMenu_Load(object sender, EventArgs e)
{
    // Enable the shape engine
    fpSpread1.Features.EnhancedShapeEngine = true;
    var worksheet = fpSpread1_Sheet1.AsWorksheet();

    // Add shape in worksheet
    worksheet.Shapes.AddShape(GrapeCity.Spreadsheet.Drawing.AutoShapeType.Octagon, 10,
    10, 151.5, 114);
    // Capture BeforeRightClick event
    fpSpread1.BeforeRightClick += OnBeforeRightClick;
}
private void OnBeforeRightClick(object sender,
FarPoint.Win.Spread.BeforeRightClickEventArgs e)
{
    if (e.ContextMenuType == FarPoint.Win.Spread.ContextMenuType.ShapeDrag)
    {
        GrapeCity.Spreadsheet.Drawing.IShapeRange movingShapes =
(GrapeCity.Spreadsheet.Drawing.IShapeRange) sender;
        // To remove context menu
    }
}
```

```

        // e.ContextMenuStrip = null;
        // To remove item from context menu
        e.ContextMenuStrip.Items[0].Visible = false;
        // To add a new item in context menu
        e.ContextMenuStrip.Items.Add("New Item Added");
    }
}

```

VB

```

Private Sub ShapeContextMenu_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    'Enable the shape engine
    FpSpread1.Features.EnhancedShapeEngine = True
    Dim worksheet = FpSpread1_Sheet1.AsWorksheet()
    'Add shape in worksheet
    worksheet.Shapes.AddShape(GrapeCity.Spreadsheet.Drawing.AutoShapeType.Octagon, 10,
10, 151.5, 114)
    'Capture BeforeRightClick event
    AddHandler FpSpread1.BeforeRightClick, AddressOf OnBeforeRightClick
End Sub
Private Sub OnBeforeRightClick(ByVal sender As Object, ByVal e As
FarPoint.Win.Spread.BeforeRightClickEventArgs)
    If e.ContextMenuType = FarPoint.Win.Spread.ContextMenuType.ShapeDrag Then
        Dim movingShapes As GrapeCity.Spreadsheet.Drawing.IShapeRange = CType(sender,
GrapeCity.Spreadsheet.Drawing.IShapeRange)
        'To remove context menu
        'e.ContextMenuStrip = Nothing
        'To remove item from context menu
        e.ContextMenuStrip.Items(0).Visible = False
        'To add a new item in context menu
        e.ContextMenuStrip.Items.Add("New Item Added")
    End If
End Sub

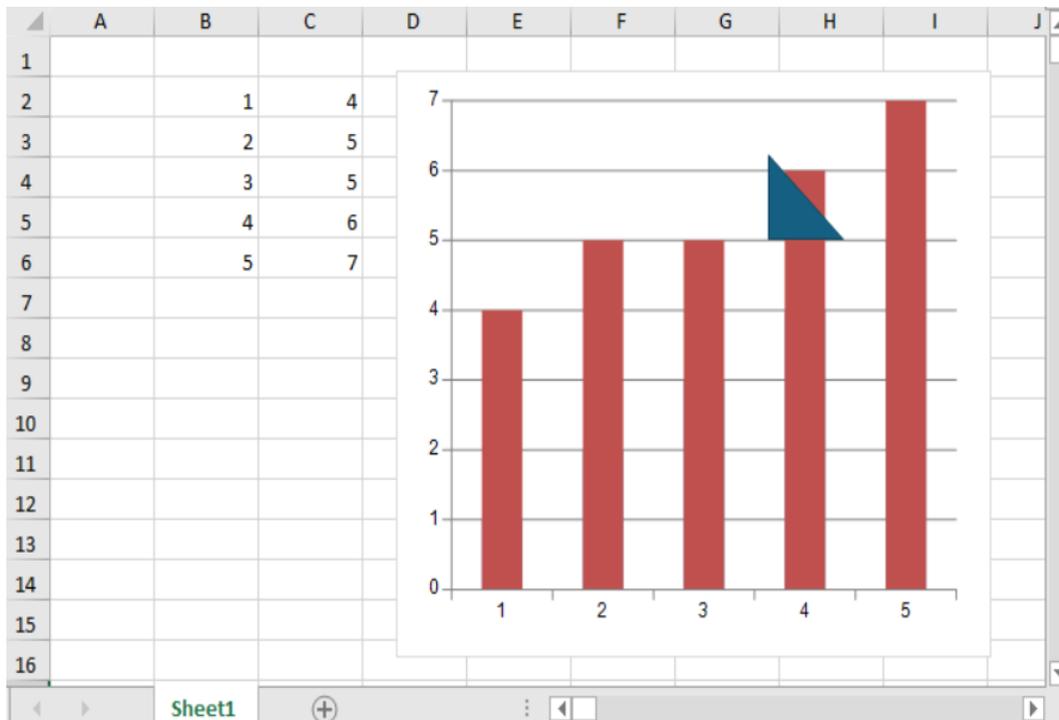
```

Add Shapes to Charts/ChartSheet

Users can embed shapes inside a chart object or in a `ChartSheet`. Once the shapes are added to a chart or `ChartSheet`, these shapes interact as normal shapes, but their boundaries are limited to the extent of the chart only. To work with shapes, use the properties and methods of the **IShape** and **IShapeRange** interfaces.

Examples

The following image depicts a triangle shape embedded inside the chart object.



The following example code shows how to add a shape to a chart object.

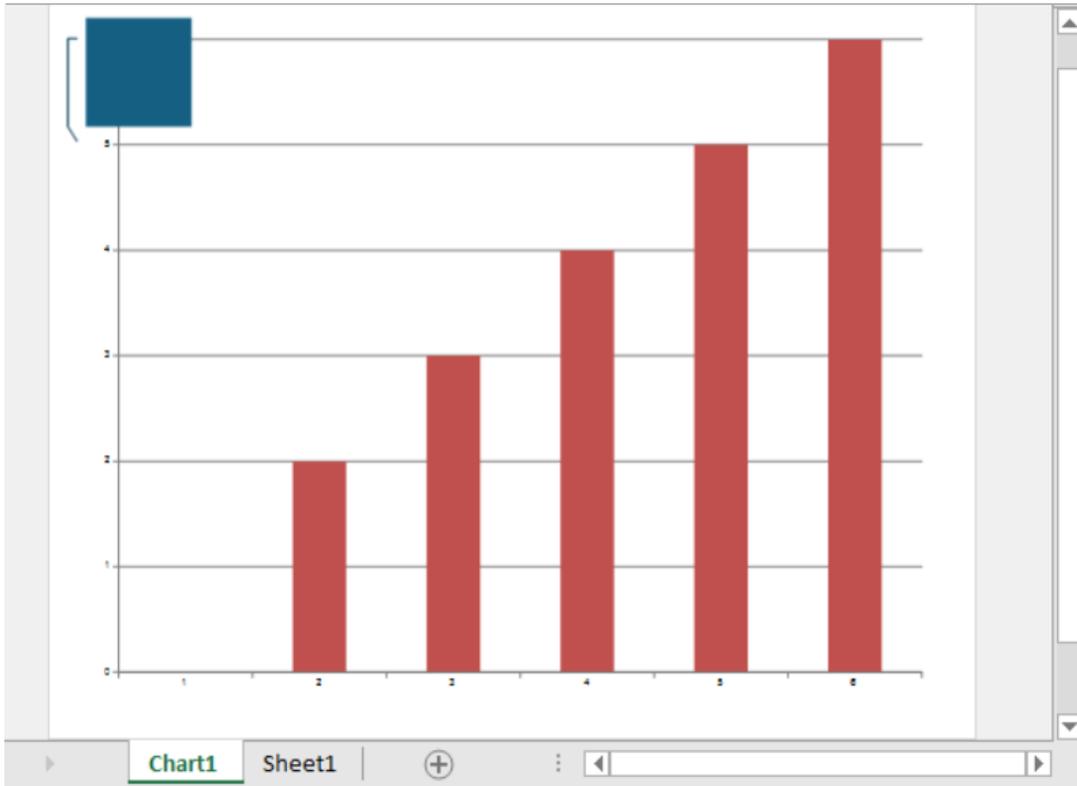
C#

```
// Add shape to ChartObject
IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;
fpSpread1.Features.EnhancedShapeEngine = true;
TestActiveSheet.SetValue(1, 1, new int[,] { { 1, 4 }, { 2, 5 }, { 3, 5 }, { 4, 6 }, {
5, 7 } });
FarPoint.Win.Spread.Model.CellRange range = new FarPoint.Win.Spread.Model.CellRange(1,
1, 5, 2);
fpSpread1.ActiveSheet.AddChart(range, typeof(BarSeries), 400, 350, 250, 120,
ChartViewType.View2D, false);
TestActiveSheet.ChartObjects[0].Chart.Shapes.AddShape(AutoShapeType.RightTriangle, 250,
50, 50, 50);
```

VB

```
' Add shape to ChartObject
Dim TestActiveSheet As IWorksheet = fpSpread1.AsWorkbook().ActiveSheet
fpSpread1.Features.EnhancedShapeEngine = True
TestActiveSheet.SetValue(1, 1, New Integer(,) {
{1, 4},
{2, 5},
{3, 5},
{4, 6},
{5, 7}})
Dim range As FarPoint.Win.Spread.Model.CellRange = New
FarPoint.Win.Spread.Model.CellRange(1, 1, 5, 2)
fpSpread1.ActiveSheet.AddChart(range, GetType(BarSeries), 400, 350, 250, 120,
ChartViewType.View2D, False)
TestActiveSheet.ChartObjects(0).Chart.Shapes.AddShape(AutoShapeType.RightTriangle, 250,
50, 50, 50)
```

Similarly, the image below shows a callout shape added to the ChartSheet.



The following example code shows how to add a shape to a ChartSheet.

C#

```
// Add shape to ChartSheet
IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;
fpSpread1.Features.EnhancedShapeEngine = true;
fpSpread1.AsWorkbook().ActiveSheet.SetValue(1, 1, new object[,] { { "data1", "data2",
"data3", "data4", "data5", "data6" }, { "Series1", 2, 3, 4, 5, 6 }, { "Series2", 2, 3,
4, 5, 6 }, { "Series3", 12, 3, 4, 5, 6 } });
fpSpread1.AsWorkbook().Charts.Add();
var spreadchart = fpSpread1.Sheets[0].Charts[0];
spreadchart.DataFormula = "Sheet1!B2:G3";
var shape = fpSpread1.AsWorkbook().Charts[0].Shapes.AddCallout(CalloutType.Three, 15,
15, 100, 100);
```

VB

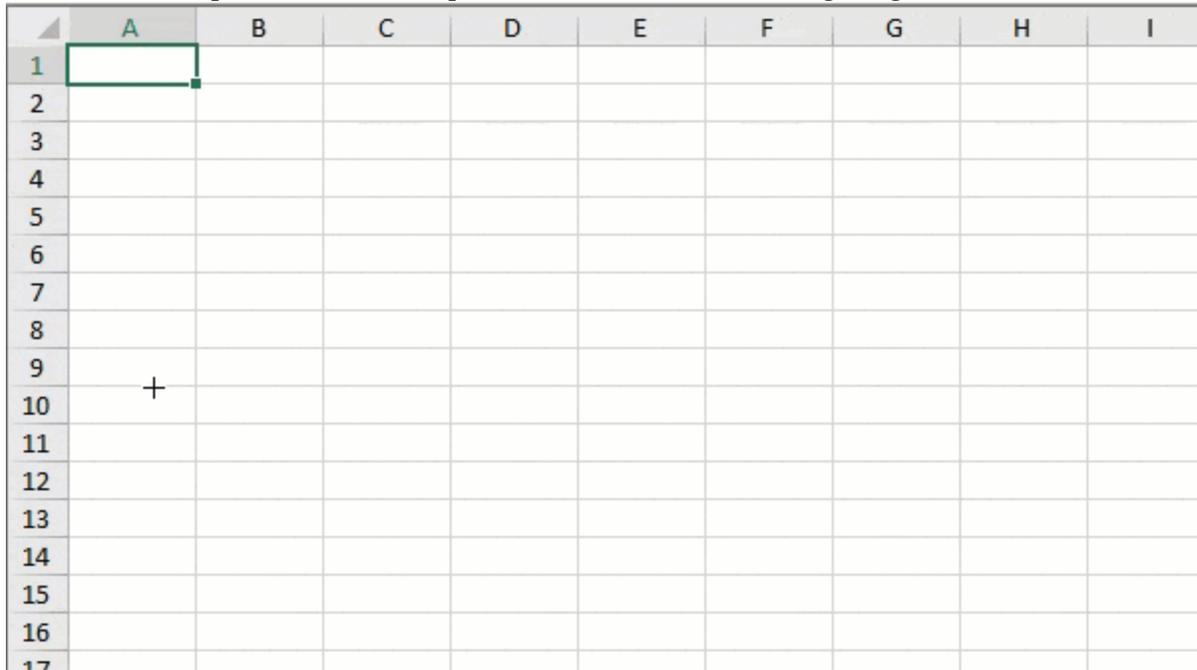
```
' Add shape to ChartSheet
Dim TestActiveSheet As IWorksheet = fpSpread1.AsWorkbook().ActiveSheet
fpSpread1.Features.EnhancedShapeEngine = True
fpSpread1.AsWorkbook().ActiveSheet.SetValue(1, 1, New Object(,) {
    {"data1", "data2", "data3", "data4", "data5", "data6"},
    {"Series1", 2, 3, 4, 5, 6},
    {"Series2", 2, 3, 4, 5, 6},
    {"Series3", 12, 3, 4, 5, 6}})
fpSpread1.AsWorkbook().Charts.Add()
Dim spreadchart = fpSpread1.Sheets(0).Charts(0)
```

```
spreadchart.DataFormula = "Sheet1!B2:G3"
Dim shape = fpSpread1.AsWorkbook().Charts(0).Shapes.AddCallout(CalloutType.Three, 15, 15, 100, 100)
```

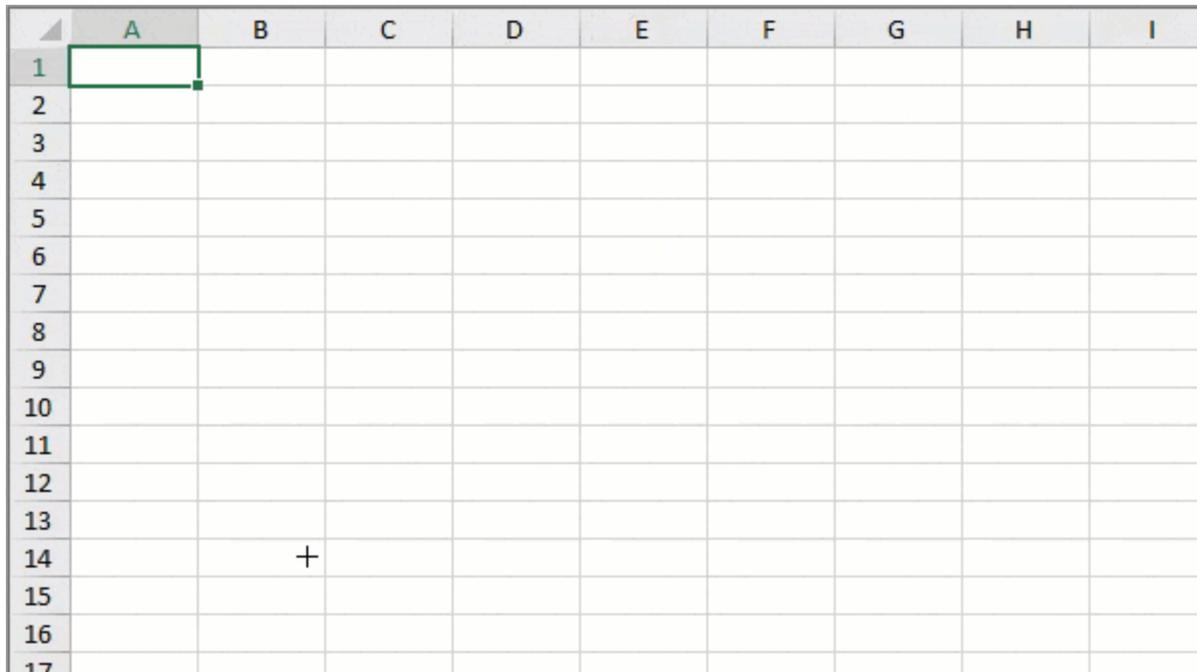
Drawing Freeform

Users can create open and closed custom shapes using **annotation mode ('AnnotationMode Enumeration' in the on-line documentation)** options such as the Freeform Shape, Freeform Scribble, and Curve.

- The Freeform Shape mode draws a shape that has both curved and straight segments.

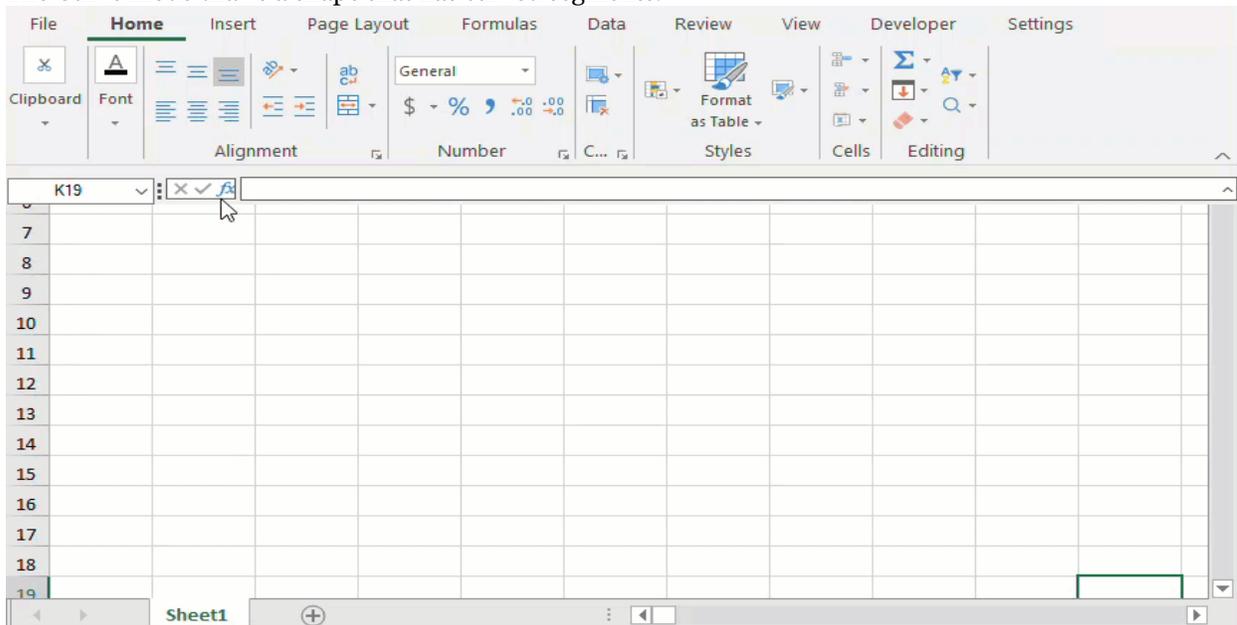


- The Freeform Scribble mode draws a shape that looks like it was drawn with a pen by hand or to create smooth curves.



The freeform shape and freeform scribble modes show the following behavior:

- Pressing the Enter key finishes the drawing.
 - Pressing the ESC key skips the current segment and finishes the drawing.
 - In Freeform Shape, holding Shift makes the lines aligned by multiple(s) of 45 degrees.
- The Curve mode draws a shape that has curved segments.



To draw a shape with curve annotation, click where you want the curve to start, move the mouse to draw, and then click wherever you want to add a curve. However, to end the drawing, perform any of the following:

- Make a single click at the start point if you want a close shape.
- Double-click or press the Esc/Enter key at any point if you want an open shape.

The following code shows how to enable the freeform shape, freeform scribble, and curve modes:

C#

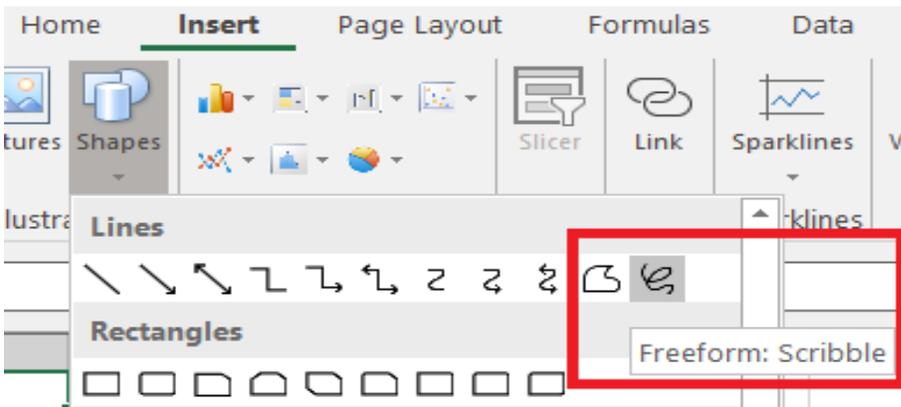
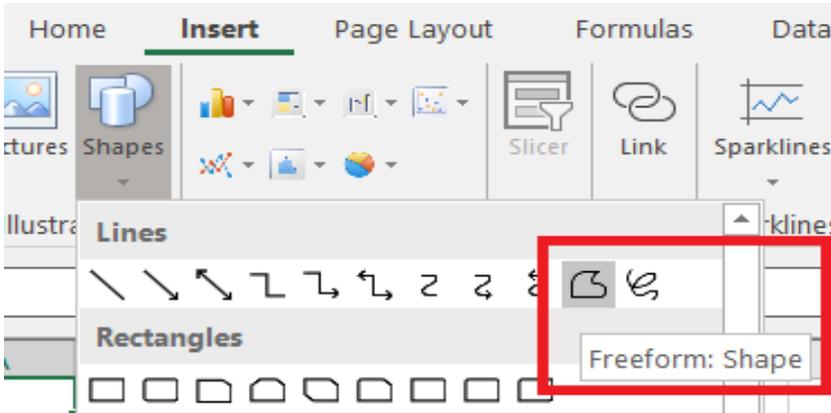
```
// Annotation Mode - FreeForm
fpSpread1.StartAnnotationMode(FarPoint.Win.Spread.AnnotationMode.Freeform);
// Annotation Mode - Scribble
fpSpread1.StartAnnotationMode(FarPoint.Win.Spread.AnnotationMode.Scribble);
// Annotation Mode - Curve
fpSpread1.LegacyBehaviors = LegacyBehaviors.None;
fpSpread1.StartAnnotationMode(FarPoint.Win.Spread.AnnotationMode.Curve);
```

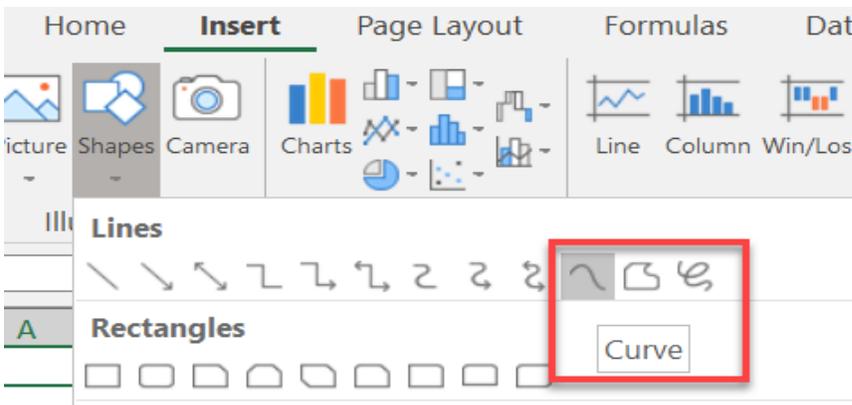
VB

```
'Annotation Mode - FreeForm
FpSpread1.StartAnnotationMode(FarPoint.Win.Spread.AnnotationMode.Freeform)
'Annotation Mode - Scribble
FpSpread1.StartAnnotationMode(FarPoint.Win.Spread.AnnotationMode.Scribble)
'Annotation Mode - Curve
fpSpread1.LegacyBehaviors = LegacyBehaviors.None
FpSpread1.StartAnnotationMode(FarPoint.Win.Spread.AnnotationMode.Curve)
```

Using Spread Designer

1. Enable **EnhancedShapeEngine** from the Spread property side pane.
2. Select the **Insert** menu.
3. Select the **Shapes** dropdown.
4. Select the freeform mode of choice.





5. Draw the shape on the sheet.
6. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Supported Shortcut Keys

The following keyboard shortcuts can be used while working with shapes:

Shortcut Key	Description
Left/Right/Up/Down key	Use this key to move the shape to different directions.
Tab or Shift+Tab	Use this key to move from one shape to other.
Delete	Use this key to remove the selected shape from the spreadsheet.
Alt+ Left/Right	Use this key to rotate the shape to left or right side.
Shift + Up/Down/Left/Right	Use this key to resize the shape to smaller or bigger follow vertical or horizontal.
Ctrl+A	Use this key to select all shapes in SheetView.
Ctrl+D or Ctrl+move mouse	Use these keys to duplicate the selected shapes.
Ctrl+Shift + Up/Down/Left/Right	Use this key to resize the shape (increase or decrease its dimensions) and rotate it vertically or horizontally.
Keep right mouse down and move	Use this action to display the menu to select the "copy shape" option or the "move shape" option.
Esc	Use this key to deselect the shape.

Creating Enhanced Camera Shape

Camera shape, as the name suggests, is a mirror image of a referenced area in a spreadsheet. It is a dynamic image, meaning that any change in the referenced region is reflected in the image as well. You can create a camera shape by referring to **Creating Camera Shapes** topic.

Additionally, Spread for Winforms provides an enhanced camera shape that inherits all the features of the enhanced shape engine. For example, the enhanced camera shape can be moved, resized, rotated, and supported for Excel I/O. They can also be grouped or ungrouped with other shapes and copy-pasted from one sheet to another.

The enhanced camera shape displays the contents of a cell range by linking the cell range to a shape. On selecting the camera shape, it displays the source cell range in the formula bar. You can edit this cell range or defined name (of any cell range) to dynamically switch the camera shape's source data.

Picture 1										
= \$A\$1:\$D\$7										
	A	B	C	D	E	F	G	H	I	J
1	Item	Units	Unit Cost	Total		Item	Units	Unit Cost	Total	
2	Desk	2	125	250		Desk	2	125	250	
3	Pen Set	5	125	625		Pen Set	5	125	625	
4	Pencil	7	1.29	9.03		Pencil	7	1.29	9.03	
5	Binder	11	4.99	54.89		Binder	11	4.99	54.89	
6	Pen	28	19.99	559.72		Pen	28	19.99	559.72	
7	Sum Total	53		1498.64		Sum Total	53		1498.64	
8										
9										
10										
11										
12	OrderDate	Region	Rep	Item	Units					
13	01-06-2020	East	Jones	Pencil	95					
14	1/23/2020	Central	Kivell	Binder	50					
15	02-09-2020	Central	Jardine	Pencil	36					
16	2/26/2020	Central	Gill	Pen	27					
17	3/15/2020	West	Sorvino	Pencil	56					
18	04-01-2020	East	Jones	Binder	60					
19	4/18/2020	Central	Andrews	Pencil	75					
20										

Adding Camera Shape

You can add the enhanced camera shape using the **IPicture.Paste ('Paste Method' in the on-line documentation)** method. It accepts parameters such as the destination in the sheet and whether to establish a link to the source of the pasted picture. The **RichClipboard ('RichClipboard Property' in the on-line documentation)** property needs to be true to use this method.

You can also convert a picture to a camera shape by using **IShape.Formula ('Formula Property' in the on-line documentation)** property to set the shape formula.

The implementation of the new camera shape takes effect if **EnhancedShapeEngine ('EnhancedShapeEngine Property' in the on-line documentation)** is true.

C#

```
// Get workbook and activesheet
GrapeCity.Spreadsheet.IWorkbook TestWorkBook = fpSpread1.AsWorkbook();
GrapeCity.Spreadsheet.IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;
// Set values in worksheet
TestActiveSheet.Cells["A1"].Value = 5;
TestActiveSheet.Cells["B1"].Formula = "=SUM(A1,5)";
TestActiveSheet.Cells["A2"].Value = "Enhanced";
TestActiveSheet.Cells["B2"].Value = "Camera";
TestActiveSheet.Cells["C2"].Value = "Shape";
fpSpread1.Features.EnhancedShapeEngine = true;
fpSpread1.Features.RichClipboard = true;
// Adding camera shape by using IPictures.Paste method
TestActiveSheet.Cells["A1:E7"].Copy(true); // Have to Copy with bool showUI = true
TestActiveSheet.Pictures.Paste("D11", true); // CameraShape refers to Sheet1!$A$1:$E$7 is created
TestActiveSheet.Pictures.Paste("F11", false); // A picture that snap content of A1:E7 is created
// Set size for picture without autosize behavior
fpSpread1.Features.EnhancedShapeEngine = true;
int rowHeight = TestActiveSheet.Rows[0].RowHeight;
int colWidth = TestActiveSheet.Columns[0].ColumnWidth;
IShape picture = TestActiveSheet.Shapes.AddPictureToCell(null, true, true, 1, 3, colWidth * 5, rowHeight * 7);
picture.Formula = "A1:E7";
// Add by Pictures.Paste to have auto-size behavior
fpSpread1.Features.EnhancedShapeEngine = true;
fpSpread1.Features.RichClipboard = true;
TestActiveSheet.Cells["A1:E5"].Copy(true);
IPicture picture = TestActiveSheet.Pictures.Paste("D4", true);
picture.Formula = "A1:E7";
```

VB

```
' Get workbook and activesheet
Dim TestWorkBook As GrapeCity.Spreadsheet.IWorkbook = fpSpread1.AsWorkbook()
Dim TestActiveSheet As GrapeCity.Spreadsheet.IWorksheet = fpSpread1.AsWorkbook().ActiveSheet
' Set values in worksheet
TestActiveSheet.Cells("A1").Value = 5
TestActiveSheet.Cells("B1").Formula = "=SUM(A1,5)"
TestActiveSheet.Cells("A2").Value = "Enhanced"
TestActiveSheet.Cells("B2").Value = "Camera"
TestActiveSheet.Cells("C2").Value = "Shape"
fpSpread1.Features.EnhancedShapeEngine = True
fpSpread1.Features.RichClipboard = True
' Adding camera shape by using IPictures.Paste method
TestActiveSheet.Cells("A1:E7").Copy(True) ' Have to Copy with bool showUI = true
TestActiveSheet.Pictures.Paste("D11", True) ' CameraShape refers to Sheet1!$A$1:$E$7 is created
TestActiveSheet.Pictures.Paste("F11", False) ' A picture that snap content of A1:E7 is created
' Set size for picture without autosize behavior
fpSpread1.Features.EnhancedShapeEngine = True
Dim rowHeight As Integer = TestActiveSheet.Rows(0).RowHeight
Dim colWidth As Integer = TestActiveSheet.Columns(0).ColumnWidth
Dim picture As IShape = TestActiveSheet.Shapes.AddPictureToCell(Nothing, True, True, 1, 3, colWidth * 5,
rowHeight * 7)
picture.Formula = "A1:E7"
' Add by Pictures.Paste to have auto-size behavior
fpSpread1.Features.EnhancedShapeEngine = True
fpSpread1.Features.RichClipboard = True
TestActiveSheet.Cells("A1:E5").Copy(True)
Dim picture As IPicture = TestActiveSheet.Pictures.Paste("D4", True)
picture.Formula = "A1:E7"
```

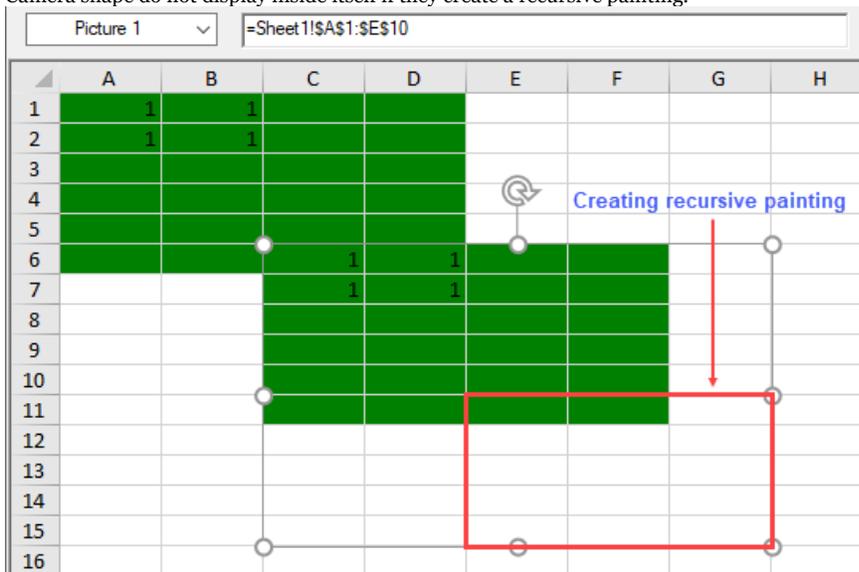
 **Note:** Set fill doesn't have an effect on the enhanced camera shape.

Recursive Painting in Camera Shape

An enhanced camera shape is capable of being displayed inside another camera shape. Unlike Excel, camera shapes in Spread for Winforms do not display inside itself or another camera shapes if they create a recursive painting.

You can observe the different behaviors in the examples below:

- Camera shape do not display inside itself if they create a recursive painting.



Show Code

C#

```
// Camera shape creating recursive painting over itself
// Without autosize behavior
fpSpread1.Features.EnhancedShapeEngine = true;
```

```

GrapeCity.Spreadsheet.IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;
GrapeCity.Spreadsheet.Color greenColor =
GrapeCity.Spreadsheet.Color.FromKnownColor(GrapeCity.Core.KnownColor.Green);
TestActiveSheet.Cells["A1:D6"].Interior.Color = greenColor;
TestActiveSheet.Cells[0, 0, 1, 1].Value = 1;
int rowHeight = TestActiveSheet.Rows[0].RowHeight;
int colWidth = TestActiveSheet.Columns[0].ColumnWidth;
IShape picture = TestActiveSheet.Shapes.AddPictureToCell(null, true, true, 5, 2, colWidth * 5,
rowHeight * 7);
picture.Formula = "Sheet1!$A$1:$E$10";
// With autosize behavior
fpSpread1.Features.EnhancedShapeEngine = true;
fpSpread1.Features.RichClipboard = true;
GrapeCity.Spreadsheet.IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;
GrapeCity.Spreadsheet.Color greenColor =
GrapeCity.Spreadsheet.Color.FromKnownColor(GrapeCity.Core.KnownColor.Green);
TestActiveSheet.Cells["A1:D6"].Interior.Color = greenColor;
TestActiveSheet.Cells[0, 0, 1, 1].Value = 1;
TestActiveSheet.Cells["A1:E5"].Copy(true);
IPicture picture = TestActiveSheet.Pictures.Paste("D4", true);
picture.Formula = "Sheet1!$A$1:$E$10";

```

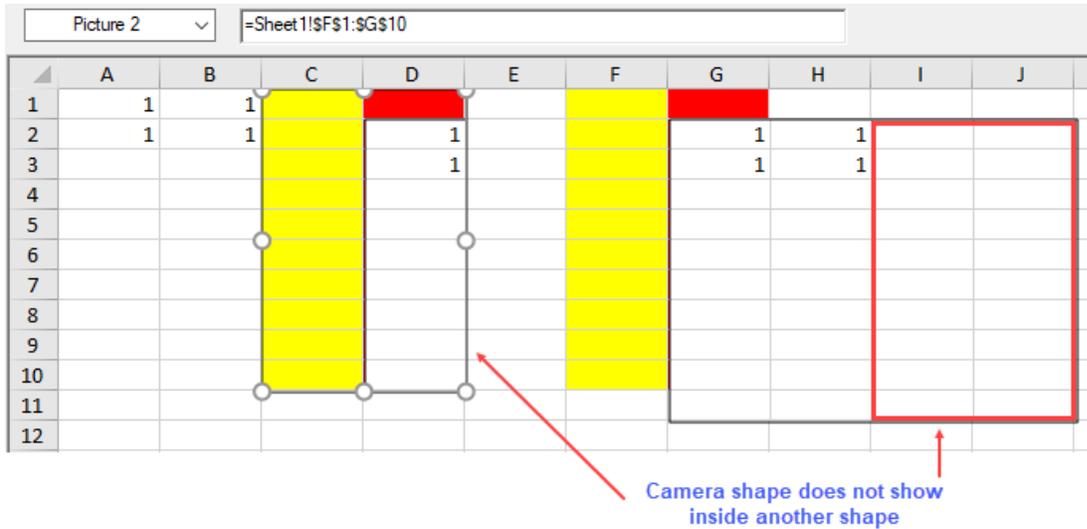
VB

```

' Camera shape creating recursive painting over itself
' Without autosize behavior
fpSpread1.Features.EnhancedShapeEngine = True
Dim TestActiveSheet As GrapeCity.Spreadsheet.IWorksheet = fpSpread1.AsWorkbook().ActiveSheet
Dim greenColor As GrapeCity.Spreadsheet.Color =
GrapeCity.Spreadsheet.Color.FromKnownColor(GrapeCity.Core.KnownColor.Green)
TestActiveSheet.Cells("A1:D6").Interior.Color = greenColor
TestActiveSheet.Cells(0, 0, 1, 1).Value = 1
Dim rowHeight As Integer = TestActiveSheet.Rows(0).RowHeight
Dim colWidth As Integer = TestActiveSheet.Columns(0).ColumnWidth
Dim picture As IShape = TestActiveSheet.Shapes.AddPictureToCell(Nothing, True, True, 5, 2, colWidth *
5, rowHeight * 7)
picture.Formula = "Sheet1!$A$1:$E$10"
' With autosize behavior
fpSpread1.Features.EnhancedShapeEngine = True
fpSpread1.Features.RichClipboard = True
Dim TestActiveSheet As GrapeCity.Spreadsheet.IWorksheet = fpSpread1.AsWorkbook().ActiveSheet
Dim greenColor As GrapeCity.Spreadsheet.Color =
GrapeCity.Spreadsheet.Color.FromKnownColor(GrapeCity.Core.KnownColor.Green)
TestActiveSheet.Cells("A1:D6").Interior.Color = greenColor
TestActiveSheet.Cells(0, 0, 1, 1).Value = 1
TestActiveSheet.Cells("A1:E5").Copy(True)
Dim picture As IPicture = TestActiveSheet.Pictures.Paste("D4", True)
picture.Formula = "Sheet1!$A$1:$E$10"

```

-
- Camera shape do not display inside another camera shape if they create a recursive painting.



Show Code

C#

```
// Camera shape creating recursive painting over another camera shape
// Without autosize behavior
fpSpread1.Features.EnhancedShapeEngine = true;
GrapeCity.Spreadsheet.IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;
GrapeCity.Spreadsheet.Color yellowColor =
GrapeCity.Spreadsheet.Color.FromKnownColor(GrapeCity.Core.KnownColor.Yellow);
GrapeCity.Spreadsheet.Color redColor =
GrapeCity.Spreadsheet.Color.FromKnownColor(GrapeCity.Core.KnownColor.Red);
TestActiveSheet.Cells[0, 0, 1, 1].Value = 1;
TestActiveSheet.Cells["G1:G10"].Interior.Color = redColor;
TestActiveSheet.Cells["F1:F10"].Interior.Color = yellowColor;
int rowHeight = TestActiveSheet.Rows[0].RowHeight;
int colWidth = TestActiveSheet.Columns[0].ColumnWidth;
IShape picture = TestActiveSheet.Shapes.AddPictureToCell(null, true, true, 1, 6, colWidth * 5, rowHeight
* 7);
picture.Formula = "Sheet1!$A$1:$D$10";
IShape picture1 = TestActiveSheet.Shapes.AddPictureToCell(null, true, true, 0, 2, colWidth * 2,
rowHeight * 3);
picture1.Formula = "Sheet1!$F$1:$G$10";
TestActiveSheet.Shapes.Range(new int[] { 0, 0 }).Line.ForeColor.ARGB =
System.Drawing.Color.Black.ToArgb();
// With autosize behavior
fpSpread1.Features.EnhancedShapeEngine = true;
fpSpread1.Features.RichClipboard = true;
GrapeCity.Spreadsheet.IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;
GrapeCity.Spreadsheet.Color yellowColor =
GrapeCity.Spreadsheet.Color.FromKnownColor(GrapeCity.Core.KnownColor.Yellow);
GrapeCity.Spreadsheet.Color redColor = GrapeCity.Spreadsheet.Color.FromKnownColor(GrapeCity.Core.KnownColor.Red);
TestActiveSheet.Cells[0, 0, 1, 1].Value = 1;
TestActiveSheet.Cells["G1:G10"].Interior.Color = redColor;
TestActiveSheet.Cells["F1:F10"].Interior.Color = yellowColor;
TestActiveSheet.Cells["A1:E5"].Copy(true);
IPicture picture = TestActiveSheet.Pictures.Paste("D4", true);
picture.Formula = "Sheet1!$A$1:$D$10";
IPicture picture1 = TestActiveSheet.Pictures.Paste("C1", true);
picture1.Formula = "Sheet1!$F$1:$G$10";
TestActiveSheet.Shapes.Range(new int[] { 0, 0 }).Line.ForeColor.ARGB =
System.Drawing.Color.Black.ToArgb();
```

VB

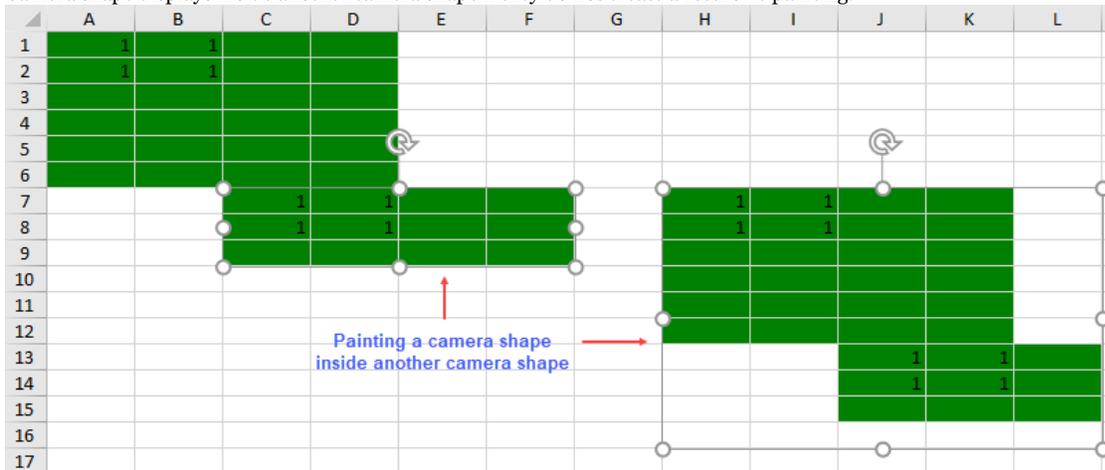
```
' Camera shape creating recursive painting over another camera shape
' Without autosize behavior
fpSpread1.Features.EnhancedShapeEngine = True
Dim TestActiveSheet As GrapeCity.Spreadsheet.IWorksheet = fpSpread1.AsWorkbook().ActiveSheet
```

```

Dim yellowColor As GrapeCity.Spreadsheet.Color =
GrapeCity.Spreadsheet.Color.FromKnownColor(GrapeCity.Core.KnownColor.Yellow)
Dim redColor As GrapeCity.Spreadsheet.Color =
GrapeCity.Spreadsheet.Color.FromKnownColor(GrapeCity.Core.KnownColor.Red)
TestActiveSheet.Cells(0, 0, 1, 1).Value = 1
TestActiveSheet.Cells("G1:G10").Interior.Color = redColor
TestActiveSheet.Cells("F1:F10").Interior.Color = yellowColor
Dim rowHeight As Integer = TestActiveSheet.Rows(0).RowHeight
Dim colWidth As Integer = TestActiveSheet.Columns(0).ColumnWidth
Dim picture As IShape = TestActiveSheet.Shapes.AddPictureToCell(Nothing, True, True, 1, 6, colWidth *
5, rowHeight * 7)
picture.Formula = "Sheet1!$A$1:$D$10"
Dim picture1 As IShape = TestActiveSheet.Shapes.AddPictureToCell(Nothing, True, True, 0, 2, colWidth *
2, rowHeight * 3)
picture1.Formula = "Sheet1!$F$1:$G$10"
TestActiveSheet.Shapes.Range((New Integer() {0, 0})).Line.ForeColor.ARGB =
Drawing.Color.Black.ToArgb()
' With autosize behavior
fpSpread1.Features.EnhancedShapeEngine = True
fpSpread1.Features.RichClipboard = True
Dim TestActiveSheet As GrapeCity.Spreadsheet.IWorksheet = fpSpread1.AsWorkbook().ActiveSheet
Dim yellowColor As GrapeCity.Spreadsheet.Color =
GrapeCity.Spreadsheet.Color.FromKnownColor(GrapeCity.Core.KnownColor.Yellow)
Dim redColor As GrapeCity.Spreadsheet.Color =
GrapeCity.Spreadsheet.Color.FromKnownColor(GrapeCity.Core.KnownColor.Red)
TestActiveSheet.Cells(0, 0, 1, 1).Value = 1
TestActiveSheet.Cells("G1:G10").Interior.Color = redColor
TestActiveSheet.Cells("F1:F10").Interior.Color = yellowColor
TestActiveSheet.Cells("A1:E5").Copy(True)
Dim picture As IPicture = TestActiveSheet.Pictures.Paste("D4", True)
picture.Formula = "Sheet1!$A$1:$D$10"
Dim picture1 As IPicture = TestActiveSheet.Pictures.Paste("C1", True)
picture1.Formula = "Sheet1!$F$1:$G$10"
TestActiveSheet.Shapes.Range((New Integer() {0, 0})).Line.ForeColor.ARGB =
Drawing.Color.Black.ToArgb()

```

- Camera shape displays inside another camera shape if they do not create a recursive painting.



Show Code

C#

```

// Camera shape displays inside another camera shape without creating recursive painting
// Without autosize behavior
fpSpread1.Features.EnhancedShapeEngine = true;
GrapeCity.Spreadsheet.IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;
GrapeCity.Spreadsheet.Color greenColor =
GrapeCity.Spreadsheet.Color.FromKnownColor(GrapeCity.Core.KnownColor.Green);
TestActiveSheet.Cells["A1:D6"].Interior.Color = greenColor;
TestActiveSheet.Cells[0, 0, 1, 1].Value = 1;
int rowHeight = TestActiveSheet.Rows[0].RowHeight;
int colWidth = TestActiveSheet.Columns[0].ColumnWidth;
IShape picture = TestActiveSheet.Shapes.AddPictureToCell(null, true, true, 6, 2, colWidth * 5, rowHeight

```

```

* 7);
picture.Formula = "Sheet1!$A$1:$D$3";
IShape picture1 = TestActiveSheet.Shapes.AddPictureToCell(null, true, true, 6, 7, colWidth * 3,
rowHeight * 3);
picture1.Formula = "Sheet1!$A$1:$E$10";
// With autosize behavior
fpSpread1.Features.EnhancedShapeEngine = true;
fpSpread1.Features.RichClipboard = true;
GrapeCity.Spreadsheet.IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;
GrapeCity.Spreadsheet.Color greenColor =
GrapeCity.Spreadsheet.Color.FromKnownColor(GrapeCity.Core.KnownColor.Green);
TestActiveSheet.Cells["A1:D6"].Interior.Color = greenColor;
TestActiveSheet.Cells[0, 0, 1, 1].Value = 1;
TestActiveSheet.Cells["A1:E5"].Copy(true);
IPicture picture = TestActiveSheet.Pictures.Paste("D4", true);
picture.Formula = "Sheet1!$A$1:$D$3";
IPicture picture1 = TestActiveSheet.Pictures.Paste("H7", true);
picture1.Formula = "Sheet1!$A$1:$E$10";

```

VB

```

' Camera shape displays inside another camera shape without creating recursive painting
' Without autosize behavior
fpSpread1.Features.EnhancedShapeEngine = True
Dim TestActiveSheet As GrapeCity.Spreadsheet.IWorksheet = fpSpread1.AsWorkbook().ActiveSheet
Dim greenColor As GrapeCity.Spreadsheet.Color =
GrapeCity.Spreadsheet.Color.FromKnownColor(GrapeCity.Core.KnownColor.Green)
TestActiveSheet.Cells("A1:D6").Interior.Color = greenColor
TestActiveSheet.Cells(0, 0, 1, 1).Value = 1
Dim rowHeight As Integer = TestActiveSheet.Rows(0).RowHeight
Dim colWidth As Integer = TestActiveSheet.Columns(0).ColumnWidth
Dim picture As IShape = TestActiveSheet.Shapes.AddPictureToCell(Nothing, True, True, 6, 2, colWidth * 5,
rowHeight * 7)
picture.Formula = "Sheet1!$A$1:$D$3"
Dim picture1 As IShape = TestActiveSheet.Shapes.AddPictureToCell(Nothing, True, True, 6, 7, colWidth *
3, rowHeight * 3)
picture1.Formula = "Sheet1!$A$1:$E$10"
' With autosize behavior
fpSpread1.Features.EnhancedShapeEngine = True
fpSpread1.Features.RichClipboard = True
Dim TestActiveSheet As GrapeCity.Spreadsheet.IWorksheet = fpSpread1.AsWorkbook().ActiveSheet
Dim greenColor As GrapeCity.Spreadsheet.Color =
GrapeCity.Spreadsheet.Color.FromKnownColor(GrapeCity.Core.KnownColor.Green)
TestActiveSheet.Cells("A1:D6").Interior.Color = greenColor
TestActiveSheet.Cells(0, 0, 1, 1).Value = 1
TestActiveSheet.Cells["A1:E5"].Copy(True)
Dim picture As IPicture = TestActiveSheet.Pictures.Paste("D4", True)
picture.Formula = "Sheet1!$A$1:$D$3"
Dim picture1 As IPicture = TestActiveSheet.Pictures.Paste("H7", True)
picture1.Formula = "Sheet1!$A$1:$E$10"

```

Camera Shape with ExcelIO

The enhanced camera shape is imported if **EnhancedShapeEngine** is true. Otherwise, the old camera shape is imported.

When exporting camera shapes to Excel:

- If the workbook contains only the enhanced camera shape, it is exported adequately.
- If the workbook contains only the old camera shape, it is exported irrespective of the EnhancedShapeEngine property.
- If the workbook contains both the old and enhanced camera shape, with EnhancedShapeEngine set to true:
 - The enhanced camera shape is exported.
 - The old camera shape is exported if the Exchangeable flag is used otherwise, it is exported as a normal picture.

Usage Scenario

Consider a scenario where the sales data of different products across a supermarket is maintained to analyze the sales trends. The data for different product categories like Fruits, Vegetables, Bakery, Meat, etc., is managed in worksheets of a spreadsheet.

You can display the summarized monthly sales data on a consolidated 'Dashboard' worksheet which shows camera shapes for the products' sales across different product categories. Any change made to the sales data is reflected in the 'Dashboard' sheet as well.



Show Code

C#

```
// Set sheet count
fpSpread1.Sheets.Count = 5;

fpSpread1.Features.EnhancedShapeEngine = true;
fpSpread1.Features.RichClipboard = true;

// Get the sheets
var sheetDashboard = fpSpread1.Sheets[0];
var sheet1 = fpSpread1.Sheets[1];
var sheet2 = fpSpread1.Sheets[2];
var sheet3 = fpSpread1.Sheets[3];
var sheet4 = fpSpread1.Sheets[4];
var worksheet0 = sheetDashboard.AsWorksheet();
var worksheet1 = sheet1.AsWorksheet();
var worksheet2 = sheet2.AsWorksheet();
var worksheet3 = sheet3.AsWorksheet();
var worksheet4 = sheet4.AsWorksheet();

// Set sheet names
sheetDashboard.SheetName = "Dashboard";
sheet1.SheetName = "Fruits";
sheet2.SheetName = "Vegetables";
sheet3.SheetName = "Meat";
sheet4.SheetName = "Bakery";

// Hide column & row headers
sheetDashboard.ColumnHeader.Visible = false;
sheetDashboard.RowHeader.Visible = false;

sheet1.ColumnHeader.Visible = false;
sheet1.RowHeader.Visible = false;

sheet2.ColumnHeader.Visible = false;
sheet2.RowHeader.Visible = false;

sheet3.ColumnHeader.Visible = false;
```

```
sheet3.RowHeader.Visible = false;

sheet4.ColumnHeader.Visible = false;
sheet4.RowHeader.Visible = false;

// Hide gridlines
sheetDashboard.VerticalGridLine = new FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Flat,
Color.Empty);
sheetDashboard.HorizontalGridLine = new FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Flat,
Color.Empty);

sheet1.VerticalGridLine = new FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Flat,
Color.Empty);
sheet1.HorizontalGridLine = new FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Flat,
Color.Empty);

sheet2.VerticalGridLine = new FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Flat,
Color.Empty);
sheet2.HorizontalGridLine = new FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Flat,
Color.Empty);

sheet3.VerticalGridLine = new FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Flat,
Color.Empty);
sheet3.HorizontalGridLine = new FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Flat,
Color.Empty);

sheet4.VerticalGridLine = new FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Flat,
Color.Empty);
sheet4.HorizontalGridLine = new FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Flat,
Color.Empty);

// Set column widths
sheet1.Columns[0].Width = 100;
sheet1.Columns[1].Width = 140;
sheet1.Columns[2].Width = 200;
for (var i = 3; i < 8; i++)
    sheet1.Columns[i].Width = 80;

sheet2.Columns[0].Width = 100;
sheet2.Columns[1].Width = 140;
sheet2.Columns[2].Width = 200;
for (var i = 3; i < 8; i++)
    sheet2.Columns[i].Width = 80;

sheet3.Columns[0].Width = 100;
sheet3.Columns[1].Width = 140;
sheet3.Columns[2].Width = 200;
for (var i = 3; i < 8; i++)
    sheet3.Columns[i].Width = 80;

sheet4.Columns[0].Width = 100;
sheet4.Columns[1].Width = 140;
sheet4.Columns[2].Width = 200;
for (var i = 3; i < 8; i++)
    sheet4.Columns[i].Width = 80;

// Set row heights
sheetDashboard.Rows[0].Height = 35;
sheetDashboard.Rows[1].Height = 5;
sheet1.Rows[0].Height = 35;
sheet2.Rows[0].Height = 35;
sheet3.Rows[0].Height = 35;
sheet4.Rows[0].Height = 35;
for (var i = 1; i < 8; i++)
{
    sheet1.Rows[i].Height = 30;
    sheet4.Rows[i].Height = 30;
}
for (var i = 1; i < 7; i++)
{
    sheet2.Rows[i].Height = 30;
```

```

}
for (var i = 1; i < 6; i++)
{
    sheet3.Rows[i].Height = 30;
}

// Create and set data arrays for different sheets
worksheet1.SetValue(0, 0, new object[,]
{
    {"Fruits", "Monthly Sales(%)", "Trend", "Week 1(Kg)", "Week 2(Kg)", "Week 3(Kg)", "Week 4(Kg)", "Total"},
    {"Apple",null , null , 1031, 927, 1287, 1484,null},
    {"Avacado",null , null , 923, 1468, 791, 981, null},
    {"Banana",null , null , 789, 571, 827, 671, null},
    {"Grapes", null,null , 782, 871, 900, 1100,null},
    {"Mango",null ,null , 829, 450, 837, 671,null},
    {"Strawberry",null ,null , 1500, 1817, 1981, 1383,null},
    {"Watermelon",null ,null , 980, 1011, 956, 817,null}
});
worksheet2.SetValue(0, 0, new object[,]
{
    {"Vegetables", "Monthly Sales(%)", "Trend", "Week 1(Kg)", "Week 2(Kg)", "Week 3(Kg)", "Week 4(Kg)",
"Total"},
    {"Carrot",null , null , 782, 490, 1012, 659,null},
    {"Onion",null , null , 1274, 1290, 721, 671,null},
    {"Potato",null , null , 2001, 2301, 1987, 2401,null},
    {"Pumpkin",null , null ,582, 771, 861, 491,null},
    {"Spinach",null , null , 302, 233, 251, 292,null},
    {"Tomato",null , null ,938, 1002, 1139, 1039,null}
});
worksheet3.SetValue(0, 0, new object[,]
{
    {"Meat", "Monthly Sales(%)", "Trend", "Week 1(Kg)", "Week 2(Kg)", "Week 3(Kg)", "Week 4(Kg)", "Total"},
    {"Beaf",null , null ,5711, 4567, 4519, 5698, null},
    {"Chicken",null , null ,6261, 5627, 3987, 4238, null},
    {"Lamb",null , null ,4789, 4571, 5827, 4671, null},
    {"Pork",null , null ,6561, 5871, 5900, 5119, null},
    {"Mutton",null , null ,5501, 4817, 5981, 6383, null}
});

worksheet4.SetValue(0, 0, new object[,]
{
    {"Bakery", "Monthly Sales(%)", "Trend", "Week 1(Kg)", "Week 2(Kg)", "Week 3(Kg)", "Week 4(Kg)", "Total"},
    {"Bread", null , null , 1031, 927, 1287, 1484, null},
    {"Brownie", null , null , 923, 1468, 791, 981, null},
    {"Cake", null , null , 789, 571, 827, 671, null},
    {"Cookie", null , null , 782, 871, 900, 1100, null},
    {"Pastry",null , null , 1500, 1817, 1981, 1383, null},
    {"Pie",null , null , 1360, 1328, 1238, 1238, null},
    {"Tarte",null , null , 1671, 1782, 2019, 1983, null}
});

// Set style for title row of sheet1, sheet2, sheet 3, sheet4
// Get range
var range = worksheet1.Cells["A1:H1"];
range.Font.ApplyFont(new GrapeCity.Spreadsheet.Font() { Name = "Arial", Size = 10, Bold = true });
range.VerticalAlignment = GrapeCity.Spreadsheet.VerticalAlignment.Center;
range.HorizontalAlignment = GrapeCity.Spreadsheet.HorizontalAlignment.Center;

range = worksheet2.Cells["A1:H1"];
range.Font.ApplyFont(new GrapeCity.Spreadsheet.Font() { Name = "Arial", Size = 10, Bold = true });
range.VerticalAlignment = GrapeCity.Spreadsheet.VerticalAlignment.Center;
range.HorizontalAlignment = GrapeCity.Spreadsheet.HorizontalAlignment.Center;

range = worksheet3.Cells["A1:H1"];
range.Font.ApplyFont(new GrapeCity.Spreadsheet.Font() { Name = "Arial", Size = 10, Bold = true });
range.VerticalAlignment = GrapeCity.Spreadsheet.VerticalAlignment.Center;
range.HorizontalAlignment = GrapeCity.Spreadsheet.HorizontalAlignment.Center;

range = worksheet4.Cells["A1:H1"];
range.Font.ApplyFont(new GrapeCity.Spreadsheet.Font() { Name = "Arial", Size = 10, Bold = true });

```

```

range.VerticalAlignment = GrapeCity.Spreadsheet.VerticalAlignment.Center;
range.HorizontalAlignment = GrapeCity.Spreadsheet.HorizontalAlignment.Center;

// Set bgcolor to first row of sheet1, sheet2, sheet3 and sheet4
worksheet1.Cells["A1:H1"].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(checked((int)0xFFE8F6FA));
worksheet2.Cells["A1:H1"].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(checked((int)0xFFE8F6FA));
worksheet3.Cells["A1:H1"].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(checked((int)0xFFE8F6FA));
worksheet4.Cells["A1:H1"].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(checked((int)0xFFE8F6FA));

// Set formulas for sheet1
sheet1.Cells[1, 7, 7, 7].Formula = "SUM(D2:G2)";
worksheet1.Cells[1, 2].Formula = "PIESPARKLINE(H2:H8,\"#919F81\",\"#D7913E\",\"#CEA722\", \"#D2DD3E\",
\"#B58091\",\"#8974A9\",\"#728BAD\")";
sheet1.AddSpanCell(1, 2, 7, 1);
sheet1.AddSpanCell(0, 1, 1, 2);
sheet1.Cells[1, 1, 7, 1].Formula = "H2 / SUM(H2: H8)";
// set bgcolor for sheet1
worksheet1.Cells[1, 1].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(checked((int)0xFF919F81));
worksheet1.Cells[2, 1].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(checked((int)0xFFD7913E));
worksheet1.Cells[3, 1].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(checked((int)0xFFCEA722));
worksheet1.Cells[4, 1].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(checked((int)0xFFD2DD3E));
worksheet1.Cells[5, 1].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(checked((int)0xFFB58091));
worksheet1.Cells[6, 1].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(checked((int)0xFF8974A9));
worksheet1.Cells[7, 1].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(checked((int)0xFF728BAD));

// Set formulas for sheet2
sheet2.Cells[1, 7, 6, 7].Formula = "SUM(D2:G2)";
worksheet2.Cells[1, 2].Formula = "PIESPARKLINE(H2:H8,\"#919F81\",\"#D7913E\",\"#CEA722\", \"#D2DD3E\",
\"#B58091\",\"#8974A9\")";
sheet2.AddSpanCell(1, 2, 6, 1);
sheet2.AddSpanCell(0, 1, 1, 2);
sheet2.Cells[1, 1, 6, 1].Formula = "H2 / SUM(H2: H8)";
// set bgcolor for sheet2
worksheet2.Cells[1, 1].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(checked((int)0xFF919F81));
worksheet2.Cells[2, 1].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(checked((int)0xFFD7913E));
worksheet2.Cells[3, 1].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(checked((int)0xFFCEA722));
worksheet2.Cells[4, 1].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(checked((int)0xFFD2DD3E));
worksheet2.Cells[5, 1].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(checked((int)0xFFB58091));
worksheet2.Cells[6, 1].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(checked((int)0xFF8974A9));

// Set formulas for sheet3
sheet3.Cells[1, 7, 5, 7].Formula = "SUM(D2:G2)";
worksheet3.Cells[1, 2].Formula = "PIESPARKLINE(H2:H8,\"#919F81\",\"#D7913E\",\"#CEA722\", \"#D2DD3E\",
\"#B58091\")";
sheet3.AddSpanCell(1, 2, 5, 1);
sheet3.AddSpanCell(0, 1, 1, 2);
sheet3.Cells[1, 1, 5, 1].Formula = "H2 / SUM(H2: H8)";
// set bgcolor for sheet3
worksheet3.Cells[1, 1].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(checked((int)0xFF919F81));
worksheet3.Cells[2, 1].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(checked((int)0xFFD7913E));
worksheet3.Cells[3, 1].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(checked((int)0xFFCEA722));
worksheet3.Cells[4, 1].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(checked((int)0xFFD2DD3E));
worksheet3.Cells[5, 1].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(checked((int)0xFFB58091));

// Set formulas for sheet4
sheet4.Cells[1, 7, 7, 7].Formula = "SUM(D2:G2)";
worksheet4.Cells[1, 2].Formula = "PIESPARKLINE(H2:H8,\"#919F81\",\"#D7913E\",\"#CEA722\", \"#D2DD3E\",
\"#B58091\",\"#8974A9\",\"#728BAD\")";
sheet4.AddSpanCell(1, 2, 7, 1);
sheet4.AddSpanCell(0, 1, 1, 2);
sheet4.Cells[1, 1, 7, 1].Formula = "H2 / SUM(H2: H8)";
// set bgcolor for sheet4
worksheet4.Cells[1, 1].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(checked((int)0xFF919F81));
worksheet4.Cells[2, 1].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(checked((int)0xFFD7913E));
worksheet4.Cells[3, 1].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(checked((int)0xFFCEA722));
worksheet4.Cells[4, 1].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(checked((int)0xFFD2DD3E));
worksheet4.Cells[5, 1].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(checked((int)0xFFB58091));
worksheet4.Cells[6, 1].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(checked((int)0xFF8974A9));
worksheet4.Cells[7, 1].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(checked((int)0xFF728BAD));

worksheet1.Range("B2:B8").NumberFormat = "0.00%";

```

```

worksheet2.Range("B2:B7").NumberFormat = "0.00%";
worksheet3.Range("B2:B6").NumberFormat = "0.00%";
worksheet4.Range("B2:B8").NumberFormat = "0.00%";

// Set header data in row 0 of Dashboard sheet and its setting
sheetDashboard.Cells[0, 0].Text = "Monthly Trend Analysis";
sheetDashboard.AddSpanCell(0, 0, 1, 14);
// Set style for header text
// Get range
range = worksheet0.Cells["A1:A14"];
// Apply style to range
range.Font.ApplyFont(new GrapeCity.Spreadsheet.Font() { Name = "Arial", Size = 12, Bold = true });
range.VerticalAlignment = GrapeCity.Spreadsheet.VerticalAlignment.Center;
range.HorizontalAlignment = GrapeCity.Spreadsheet.HorizontalAlignment.Center;
worksheet0.Cells[0, 0].Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(unchecked((int)0xFF8CA4B9));

// Add camera shapes in Dashboard sheet
worksheet1.Cells["A1:C8"].Copy(true); // Have to Copy with bool showUI = true
worksheet0.Pictures.Paste("A3", true); // CameraShape refers to worksheet1!$A$1:$C$8 is created

worksheet2.Cells["A1:C7"].Copy(true); // Have to Copy with bool showUI = true
worksheet0.Pictures.Paste("H3", true); // CameraShape refers to worksheet2!$A$1:$C$7 is created

worksheet3.Cells["A1:C6"].Copy(true); // Have to Copy with bool showUI = true
worksheet0.Pictures.Paste("A17", true); // CameraShape refers to worksheet2!$A$1:$C$5 is created

worksheet4.Cells["A1:C8"].Copy(true); // Have to Copy with bool showUI = true
worksheet0.Pictures.Paste("H17", true); // CameraShape refers to worksheet2!$A$1:$C$8 is created

// Set activesheet to dashboard sheet
fpSpread1.ActiveSheetIndex = 0;

```

Visual Basic

```

'Set sheet count
FpSpread1.Sheets.Count = 5

FpSpread1.Features.EnhancedShapeEngine = True
FpSpread1.Features.RichClipboard = True

'Get the sheets
Dim sheetDashboard = FpSpread1.Sheets(0)
Dim sheet1 = FpSpread1.Sheets(1)
Dim sheet2 = FpSpread1.Sheets(2)
Dim sheet3 = FpSpread1.Sheets(3)
Dim sheet4 = FpSpread1.Sheets(4)
Dim worksheet0 = sheetDashboard.AsWorksheet()
Dim worksheet1 = sheet1.AsWorksheet()
Dim worksheet2 = sheet2.AsWorksheet()
Dim worksheet3 = sheet3.AsWorksheet()
Dim worksheet4 = sheet4.AsWorksheet()

'Set sheet names
sheetDashboard.SheetName = "Dashboard"
sheet1.SheetName = "Fruits"
sheet2.SheetName = "Vegetables"
sheet3.SheetName = "Meat"
sheet4.SheetName = "Bakery"

'Hide column & row headers
sheetDashboard.ColumnHeader.Visible = False
sheetDashboard.RowHeader.Visible = False

sheet1.ColumnHeader.Visible = False
sheet1.RowHeader.Visible = False

sheet2.ColumnHeader.Visible = False
sheet2.RowHeader.Visible = False

sheet3.ColumnHeader.Visible = False
sheet3.RowHeader.Visible = False

```

```
sheet4.ColumnHeader.Visible = False
sheet4.RowHeader.Visible = False

'Hide gridlines
sheetDashboard.VerticalGridLine = New FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Flat,
Color.Empty)
sheetDashboard.HorizontalGridLine = New FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Flat,
Color.Empty)

sheet1.VerticalGridLine = New FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Flat, Color.Empty)
sheet1.HorizontalGridLine = New FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Flat,
Color.Empty)

sheet2.VerticalGridLine = New FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Flat, Color.Empty)
sheet2.HorizontalGridLine = New FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Flat,
Color.Empty)

sheet3.VerticalGridLine = New FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Flat, Color.Empty)
sheet3.HorizontalGridLine = New FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Flat,
Color.Empty)

sheet4.VerticalGridLine = New FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Flat, Color.Empty)
sheet4.HorizontalGridLine = New FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Flat,
Color.Empty)

'Set column widths
sheet1.Columns(0).Width = 100
sheet1.Columns(1).Width = 140
sheet1.Columns(2).Width = 200

For i = 3 To 8 - 1
    sheet1.Columns(i).Width = 80
Next

sheet2.Columns(0).Width = 100
sheet2.Columns(1).Width = 140
sheet2.Columns(2).Width = 200

For i = 3 To 8 - 1
    sheet2.Columns(i).Width = 80
Next

sheet3.Columns(0).Width = 100
sheet3.Columns(1).Width = 140
sheet3.Columns(2).Width = 200

For i = 3 To 8 - 1
    sheet3.Columns(i).Width = 80
Next

sheet4.Columns(0).Width = 100
sheet4.Columns(1).Width = 140
sheet4.Columns(2).Width = 200

For i = 3 To 8 - 1
    sheet4.Columns(i).Width = 80
Next

'Set row heights
sheetDashboard.Rows(0).Height = 35
sheetDashboard.Rows(1).Height = 5
sheet1.Rows(0).Height = 35
sheet2.Rows(0).Height = 35
sheet3.Rows(0).Height = 35
sheet4.Rows(0).Height = 35

For i = 1 To 8 - 1
    sheet1.Rows(i).Height = 30
    sheet4.Rows(i).Height = 30
Next
```

```

For i = 1 To 7 - 1
    sheet2.Rows(i).Height = 30
Next

For i = 1 To 6 - 1
    sheet3.Rows(i).Height = 30
Next

'Create and set data arrays for different sheets
worksheet1.SetValue(0, 0, New Object(,) {
    {"Fruits", "Monthly Sales(%)", "Trend", "Week 1(Kg)", "Week 2(Kg)", "Week 3(Kg)", "Week 4(Kg)", "Total"},
    {"Apple", Nothing, Nothing, 1031, 927, 1287, 1484, Nothing},
    {"Avacado", Nothing, Nothing, 923, 1468, 791, 981, Nothing},
    {"Banana", Nothing, Nothing, 789, 571, 827, 671, Nothing},
    {"Grapes", Nothing, Nothing, 782, 871, 900, 1100, Nothing},
    {"Mango", Nothing, Nothing, 829, 450, 837, 671, Nothing},
    {"Strawberry", Nothing, Nothing, 1500, 1817, 1981, 1383, Nothing},
    {"Watermelon", Nothing, Nothing, 980, 1011, 956, 817, Nothing}
})
worksheet2.SetValue(0, 0, New Object(,) {
    {"Vegetables", "Monthly Sales(%)", "Trend", "Week 1(Kg)", "Week 2(Kg)", "Week 3(Kg)", "Week 4(Kg)",
"Total"},
    {"Carrot", Nothing, Nothing, 782, 490, 1012, 659, Nothing},
    {"Onion", Nothing, Nothing, 1274, 1290, 721, 671, Nothing},
    {"Potato", Nothing, Nothing, 2001, 2301, 1987, 2401, Nothing},
    {"Pumpkin", Nothing, Nothing, 582, 771, 861, 491, Nothing},
    {"Spinach", Nothing, Nothing, 302, 233, 251, 292, Nothing},
    {"Tomato", Nothing, Nothing, 938, 1002, 1139, 1039, Nothing}
})
worksheet3.SetValue(0, 0, New Object(,) {
    {"Meat", "Monthly Sales(%)", "Trend", "Week 1(Kg)", "Week 2(Kg)", "Week 3(Kg)", "Week 4(Kg)", "Total"},
    {"Beaf", Nothing, Nothing, 5711, 4567, 4519, 5698, Nothing},
    {"Chicken", Nothing, Nothing, 6261, 5627, 3987, 4238, Nothing},
    {"Lamb", Nothing, Nothing, 4789, 4571, 5827, 4671, Nothing},
    {"Pork", Nothing, Nothing, 6561, 5871, 5900, 5119, Nothing},
    {"Mutton", Nothing, Nothing, 5501, 4817, 5981, 6383, Nothing}
})
worksheet4.SetValue(0, 0, New Object(,) {
    {"Bakery", "Monthly Sales(%)", "Trend", "Week 1(Kg)", "Week 2(Kg)", "Week 3(Kg)", "Week 4(Kg)", "Total"},
    {"Bread", Nothing, Nothing, 1031, 927, 1287, 1484, Nothing},
    {"Brownie", Nothing, Nothing, 923, 1468, 791, 981, Nothing},
    {"Cake", Nothing, Nothing, 789, 571, 827, 671, Nothing},
    {"Cookie", Nothing, Nothing, 782, 871, 900, 1100, Nothing},
    {"Pastry", Nothing, Nothing, 1500, 1817, 1981, 1383, Nothing},
    {"Pie", Nothing, Nothing, 1360, 1328, 1238, 1238, Nothing},
    {"Tarte", Nothing, Nothing, 1671, 1782, 2019, 1983, Nothing}
})

'Set style for title row of sheet1, sheet2, sheet3, sheet4
Dim range = worksheet1.Cells("A1:H1")
range.Font.ApplyFont(New GrapeCity.Spreadsheet.Font() With {
    .Name = "Arial",
    .Size = 10,
    .Bold = True
})
range.VerticalAlignment = GrapeCity.Spreadsheet.VerticalAlignment.Center
range.HorizontalAlignment = GrapeCity.Spreadsheet.HorizontalAlignment.Center

range = worksheet2.Cells("A1:H1")
range.Font.ApplyFont(New GrapeCity.Spreadsheet.Font() With {
    .Name = "Arial",
    .Size = 10,
    .Bold = True
})
range.VerticalAlignment = GrapeCity.Spreadsheet.VerticalAlignment.Center
range.HorizontalAlignment = GrapeCity.Spreadsheet.HorizontalAlignment.Center

range = worksheet3.Cells("A1:H1")
range.Font.ApplyFont(New GrapeCity.Spreadsheet.Font() With {
    .Name = "Arial",

```

```

        .Size = 10,
        .Bold = True
    })
    range.VerticalAlignment = GrapeCity.Spreadsheet.VerticalAlignment.Center
    range.HorizontalAlignment = GrapeCity.Spreadsheet.HorizontalAlignment.Center

    range = worksheet4.Cells("A1:H1")
    range.Font.ApplyFont(New GrapeCity.Spreadsheet.Font() With {
        .Name = "Arial",
        .Size = 10,
        .Bold = True
    })
    range.VerticalAlignment = GrapeCity.Spreadsheet.VerticalAlignment.Center
    range.HorizontalAlignment = GrapeCity.Spreadsheet.HorizontalAlignment.Center

    'Set bgcolor to first row of sheet1, sheet2, sheet3, and sheet4
    worksheet1.Cells("A1:H1").Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFFFE8F6FA)
    worksheet2.Cells("A1:H1").Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFFFE8F6FA)
    worksheet3.Cells("A1:H1").Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFFFE8F6FA)
    worksheet4.Cells("A1:H1").Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFFFE8F6FA)

    'Set formulas for sheet1
    sheet1.Cells(1, 7, 7, 7).Formula = "SUM(D2:G2)"
    worksheet1.Cells(1, 2).Formula = "PIESPARKLINE(H2:H8, ""#919F81"", ""#D7913E"", ""#CEA722"", ""#D2DD3E"", ""#B58091"", ""#8974A9"", ""#728BAD"")"
    sheet1.AddSpanCell(1, 2, 7, 1)
    sheet1.AddSpanCell(0, 1, 1, 2)
    sheet1.Cells(1, 1, 7, 1).Formula = "H2 / SUM(H2: H8)"
    'Set bgcolor for sheet1
    worksheet1.Cells(1, 1).Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFF919F81)
    worksheet1.Cells(2, 1).Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFFD7913E)
    worksheet1.Cells(3, 1).Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFFCEA722)
    worksheet1.Cells(4, 1).Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFFD2DD3E)
    worksheet1.Cells(5, 1).Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFFB58091)
    worksheet1.Cells(6, 1).Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFF8974A9)
    worksheet1.Cells(7, 1).Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFF728BAD)

    'Set formulas for sheet2
    sheet2.Cells(1, 7, 6, 7).Formula = "SUM(D2:G2)"
    worksheet2.Cells(1, 2).Formula = "PIESPARKLINE(H2:H8, ""#919F81"", ""#D7913E"", ""#CEA722"", ""#D2DD3E"", ""#B58091"", ""#8974A9"")"
    sheet2.AddSpanCell(1, 2, 6, 1)
    sheet2.AddSpanCell(0, 1, 1, 2)
    sheet2.Cells(1, 1, 6, 1).Formula = "H2 / SUM(H2: H8)"
    'Set bgcolor for sheet2
    worksheet2.Cells(1, 1).Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFF919F81)
    worksheet2.Cells(2, 1).Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFFD7913E)
    worksheet2.Cells(3, 1).Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFFCEA722)
    worksheet2.Cells(4, 1).Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFFD2DD3E)
    worksheet2.Cells(5, 1).Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFFB58091)
    worksheet2.Cells(6, 1).Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFF8974A9)

    'Set formulas for sheet3
    sheet3.Cells(1, 7, 5, 7).Formula = "SUM(D2:G2)"
    worksheet3.Cells(1, 2).Formula = "PIESPARKLINE(H2:H8, ""#919F81"", ""#D7913E"", ""#CEA722"", ""#D2DD3E"", ""#B58091"")"
    sheet3.AddSpanCell(1, 2, 5, 1)
    sheet3.AddSpanCell(0, 1, 1, 2)
    sheet3.Cells(1, 1, 5, 1).Formula = "H2 / SUM(H2: H8)"
    'Set bgcolor for sheet3
    worksheet3.Cells(1, 1).Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFF919F81)
    worksheet3.Cells(2, 1).Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFFD7913E)
    worksheet3.Cells(3, 1).Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFFCEA722)
    worksheet3.Cells(4, 1).Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFFD2DD3E)
    worksheet3.Cells(5, 1).Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFFB58091)

    'Set formulas for sheet4
    sheet4.Cells(1, 7, 7, 7).Formula = "SUM(D2:G2)"
    worksheet4.Cells(1, 2).Formula = "PIESPARKLINE(H2:H8, ""#919F81"", ""#D7913E"", ""#CEA722"", ""#D2DD3E"", ""#B58091"", ""#8974A9"", ""#728BAD"")"
    sheet4.AddSpanCell(1, 2, 7, 1)

```

```

sheet4.AddSpanCell(0, 1, 1, 2)
sheet4.Cells(1, 1, 7, 1).Formula = "H2 / SUM(H2: H8)"
'Set bgcolor for sheet4
worksheet4.Cells(1, 1).Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFF919F81)
worksheet4.Cells(2, 1).Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFFD7913E)
worksheet4.Cells(3, 1).Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFFCEA722)
worksheet4.Cells(4, 1).Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFFD2DD3E)
worksheet4.Cells(5, 1).Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFFB58091)
worksheet4.Cells(6, 1).Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFF8974A9)
worksheet4.Cells(7, 1).Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFF728BAD)

worksheet1.Range("B2:B8").NumberFormat = "0.00%"
worksheet2.Range("B2:B7").NumberFormat = "0.00%"
worksheet3.Range("B2:B6").NumberFormat = "0.00%"
worksheet4.Range("B2:B8").NumberFormat = "0.00%"

'Set header data in row 0 of Dashboard sheet and its setting
sheetDashboard.Cells(0, 0).Text = "Monthly Trend Analysis"
sheetDashboard.AddSpanCell(0, 0, 1, 14)
'Set style for header text
'Get range
range = worksheet0.Cells("A1:A14")
'Apply style to range
range.Font.ApplyFont(New GrapeCity.Spreadsheet.Font() With {
    .Name = "Arial",
    .Size = 12,
    .Bold = True
})
range.VerticalAlignment = GrapeCity.Spreadsheet.VerticalAlignment.Center
range.HorizontalAlignment = GrapeCity.Spreadsheet.HorizontalAlignment.Center
worksheet0.Cells(0, 0).Interior.Color = GrapeCity.Spreadsheet.Color.FromArgb(&HFF8CA4B9)

'Add camera shapes in Dashboard sheet
worksheet1.Cells("A1:C8").Copy(True) 'Have to Copy with bool showUI = true
worksheet0.Pictures.Paste("A3", True) 'CameraShape refers to worksheet1!$A$1:$C$8 is created

worksheet2.Cells("A1:C7").Copy(True) 'Have to Copy with bool showUI = true
worksheet0.Pictures.Paste("H3", True) 'CameraShape refers to worksheet1!$A$1:$C$7 is created

worksheet3.Cells("A1:C6").Copy(True) 'Have to Copy with bool showUI = true
worksheet0.Pictures.Paste("A17", True) 'CameraShape refers to worksheet1!$A$1:$C$5 is created

worksheet4.Cells("A1:C8").Copy(True) 'Have to Copy with bool showUI = true
worksheet0.Pictures.Paste("H17", True) 'CameraShape refers to worksheet1!$A$1:$C$8 is created

'Set activesheet to dashboard sheet
FpSpread1.ActiveSheetIndex = 0

```

 **Note:** You must update camera shape manually if the source data is modified using the FarPoint namespace. The following APIs can be used to refresh the enhanced camera shape:

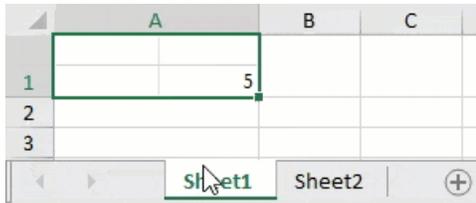
- **IWorksheet.Pictures.Refresh** - Re-paints all camera shape in the worksheet.
- **CalculationEngine.NotifyVisualChanged** - Notifies the calculation engine that the visuals are repainted.

Creating a Custom Camera

Spread for Winforms provides the **ICamera** (**'ICamera Interface' in the on-line documentation**) interface that represents the visual of a cell range in a single cell. You can use the following ICamera members:

- **Formula** (**'Formula Property' in the on-line documentation**) property gets or sets the formula that indicates source cell range of the camera.
- **Image** (**'Image Property' in the on-line documentation**) property gets the image brush of the camera object.
- **ICamera.Delete** (**'Delete Method' in the on-line documentation**) method deletes the camera.

The following GIF illustrates a custom CAMERA function that you can apply in cells. The function is used to show a cell range inside a cell.

**Show Code**

You can create the CAMERA function shown above by following the code below.

C#

```
public partial class CameraShapeInterface : Form
{
    public CameraShapeInterface()
    {
        InitializeComponent();
    }

    private void CameraShapeInterface_Load(object sender, EventArgs e)
    {
        fpSpread1.AddCustomFunction(new CameraFunction());
        IWorkbook workbook = fpSpread1.AsWorkbook();
        workbook.Worksheets.Add();
        IWorksheet sheet2 = workbook.Worksheets[1];
        sheet2.Cells["B2"].Value = 5;

        IWorksheet sheet1 = workbook.Worksheets[0];
        sheet1.Cells["A1"].ColumnWidth = sheet2.Cells["A1"].ColumnWidth * 2 + 1;
        sheet1.Cells["A1"].RowHeight = sheet2.Cells["A1"].RowHeight * 2 + 1;
        sheet1.Cells["A1"].Formula = "CAMERA(Sheet2!A1:B2)";
    }
}

public class CameraFunction : VisualFunction
{
    public CameraFunction() : base("CAMERA", 1, 1, FunctionAttributes.Variant,
(IFunctionVisualizer)CameraVisualizer.Instance, false)
    {
    }

    protected override bool IsArrayParameter(int argIndex)
    {
        return true;
    }

    protected override bool Evaluate(IArguments arguments, IValue result)
    {
        IEvaluationContext context = arguments.EvaluationContext;
        IValue range = arguments[0];
        switch (range.ValueType)
        {
            case GrapeCity.CalcEngine.ValueType.Reference:
            case GrapeCity.CalcEngine.ValueType.AdjustableReference:
                if (range.GetReferenceSource(context) is GrapeCity.Spreadsheet.IWorksheet worksheet)
                {
                    ICamera camera = worksheet.Range(range.GetReference(context)).CreateCamera();
                    VisualizationData data = new VisualizationData(camera);
                    result.SetValue(data, true);
                    break;
                }
                else
                {
                    goto default;
                }
            default:
                context.Error = CalcError.Reference;
                break;
        }
    }
}
```

```

        return true;
    }

    private class CameraVisualizer : FunctionVisualizer
    {
        internal static CameraVisualizer Instance;

        static CameraVisualizer()
        {
            Instance = new CameraVisualizer();
        }

        private CameraVisualizer()
        {
        }

        public override bool IsVisual => true;
        public override bool IsBackgroundSupported => true;

        protected override void PaintCell(IPaintingContext context, Rectangle rect, object cellValue, ref
        StyleFormat styleFormat)
        {
            if (cellValue is VisualizationData data && data.Value.ValueType ==
            GrapeCity.CalcEngine.ValueType.Object && data.Value.GetObject() is ICamera camera)
            {
                GrapeCity.Drawing.ImageBrush imageBrush = camera.Image;
                if (imageBrush != null)
                {
                    Brush brush = context.ToGdiBrush(imageBrush, rect);
                    context.Graphics.FillRectangle(brush, rect);
                    brush.Dispose();
                }
            }
        }
    }
}

```

Visual Basic

```

Public Class CameraShapeInterface
    Private Sub CameraShapeInterface_Load(sender As Object, e As EventArgs) Handles MyBase.Load

        FpSpread1.AddCustomFunction(New CameraFunction())
        Dim workbook As IWorkbook = FpSpread1.AsWorkbook()
        workbook.Worksheets.Add()
        Dim sheet2 As IWorksheet = workbook.Worksheets(1)
        sheet2.Cells("B2").Value = 5
        Dim sheet1 As IWorksheet = workbook.Worksheets(0)
        sheet1.Cells("A1").ColumnWidth = sheet2.Cells("A1").ColumnWidth * 2 + 1
        sheet1.Cells("A1").RowHeight = sheet2.Cells("A1").RowHeight * 2 + 1
        sheet1.Cells("A1").Formula = "CAMERA(Sheet2!A1:B2)"

    End Sub
End Class

Public Class CameraFunction
    Inherits VisualFunction

    Public Sub New()
        MyBase.New("CAMERA", 1, 1, FunctionAttributes.[Variant], CType(CameraVisualizer.Instance,
        IFunctionVisualizer), False)
    End Sub

    Protected Overrides Function IsArrayParameter(ByVal argIndex As Integer) As Boolean
        Return True
    End Function

    <Obsolete>
    Protected Overrides Function Evaluate(ByVal arguments As IArguments, ByVal result As IValue) As Boolean

```

```

Dim context As IEvaluationContext = arguments.EvaluationContext
Dim range As IValue = arguments(0)
Dim worksheet As IWorksheet = Nothing

Select Case range.ValueType
    Case ValueType.Reference, ValueType.AdjustableReference

        If CSharpImpl.__Assign(worksheet, TryCast(range.GetReferenceSource(context), IWorksheet))
            IsNot Nothing Then
                Dim camera As ICamera = worksheet.Range(range.GetReference(context)).CreateCamera()
                Dim data As VisualizationData = New VisualizationData(camera)
                result.SetValue(data, True)
                Exit Select
            Else
                GoTo _Select0_CaseDefault
            End If

        Case Else
            _Select0_CaseDefault:
                context.[Error] = CalcError.Reference
            End Select

        Return True
    End Function

Private Class CameraVisualizer
    Inherits FunctionVisualizer

    Friend Shared Instance As CameraVisualizer

    Shared Sub New()
        Instance = New CameraVisualizer()
    End Sub

    Private Sub New()
    End Sub

    Public Overrides ReadOnly Property IsVisual As Boolean
        Get
            Return True
        End Get
    End Property

    Public Overrides ReadOnly Property IsBackgroundSupported As Boolean
        Get
            Return True
        End Get
    End Property

    <Obsolete>
    Protected Overrides Sub PaintCell(ByVal context As IPaintingContext, ByVal rect As Rectangle, ByVal
cellValue As Object, ByRef styleFormat As StyleFormat)
        Dim data As VisualizationData = Nothing, camera As ICamera = Nothing

        If CSharpImpl.__Assign(data, TryCast(cellValue, VisualizationData)) IsNot Nothing AndAlso
data.Value.ValueType = GrapeCity.CalcEngine.ValueType.Object AndAlso CSharpImpl.__Assign(camera,
TryCast(data.Value.GetObject(), ICamera)) IsNot Nothing Then
            Dim imageBrush As GrapeCity.Drawing.ImageBrush = camera.Image

            If imageBrush IsNot Nothing Then
                Dim brush As Brush = context.ToGdiBrush(imageBrush, rect)
                context.Graphics.FillRectangle(brush, rect)
                brush.Dispose()
            End If
        End If
    End Sub

    Private Class CSharpImpl
        <Obsolete("Please refactor calling code to use normal Visual Basic assignment")>
        Shared Function __Assign(Of T)(ByRef target As T, value As T) As T
            target = value
        End Function
    End Class

```

```

        Return value
    End Function
End Class
End Class

Private Class CSharpImpl
    <Obsolete("Please refactor calling code to use normal Visual Basic assignment")>
    Shared Function __Assign(Of T)(ByRef target As T, value As T) As T
        target = value
        Return value
    End Function
End Class
End Class

```

Using the Spread Designer

1. Enable **EnhancedShapeEngine** from the Spread property side pane.
2. Select a block of cells in the designer.
3. Select the **Insert** menu.
4. Select the camera shape icon.
5. Click on the shape to move it.
6. The **Shape Format** menu with additional options is displayed.
7. From the File menu choose Apply and Exit to apply your changes to the component and exit Spread Designer.

Working with Images

Spread for WinForms provides support for customizing images provided by the user in the Spread worksheet.

All the features listed below use the enhanced shape engine so its property must be set to true. As they inherit all the features of the enhanced shape engine, the output images are supported for Excel I/O as well.

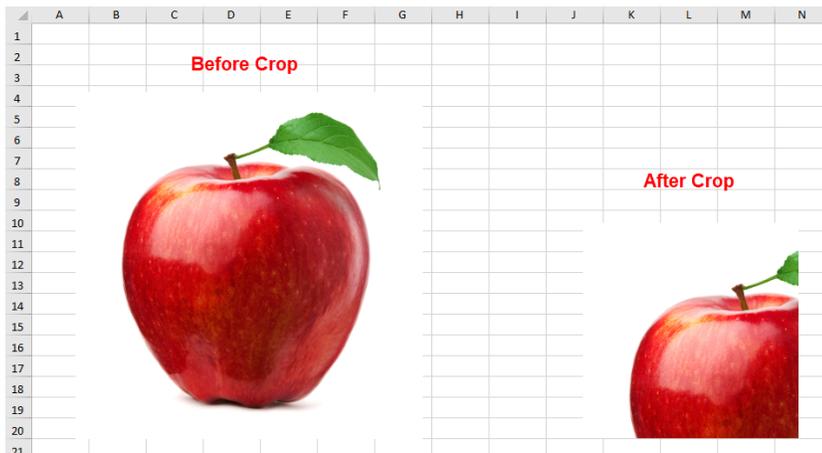
Crop Images

You can crop an image in the worksheet and trim any unnecessary outer edge parts of an image. Spread for WinForms provides the option to crop an image using code or through runtime UI crop handles.

Using Code

You can set the **ICrop** ('**ICrop Interface**' in the on-line documentation) interface properties such as *PictureOffsetX*, *ShapeTop*, *PictureWidth* to control the dimensions of an image and crop it accordingly.

The following image shows the output of an image before and after it has been modified in the code.



C#

```

// Cropping a picture using code
fpSpread1.LegacyBehaviors = FarPoint.Win.Spread.LegacyBehaviors.None;
fpSpread1.Features.EnhancedShapeEngine = true;

GrapeCity.Spreadsheet.IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;

GrapeCity.Spreadsheet.Drawing.IShape picture1 = TestActiveSheet.Shapes.AddPicture(@"D:\apple.jpg", false, false, 50, 50, 400, 400);
picture1.AutoShapeType = GrapeCity.Spreadsheet.Drawing.AutoShapeType.Rectangle;

GrapeCity.Spreadsheet.Drawing.IShape picture2 = TestActiveSheet.Shapes.AddPicture(@"D:\apple.jpg", true, true, 200, 200, 400, 400);
picture2.AutoShapeType = GrapeCity.Spreadsheet.Drawing.AutoShapeType.Rectangle;

picture2.PictureFormat.Crop.PictureOffsetX = 100;
picture2.PictureFormat.Crop.PictureOffsetY = 100;

picture2.PictureFormat.Crop.ShapeLeft = 150;

```

```
picture2.PictureFormat.Crop.ShapeTop = 150;
```

Visual Basic

```
' Cropping a picture using code
FpSpread1.LegacyBehaviors = FarPoint.Win.Spread.LegacyBehaviors.None
FpSpread1.Features.EnhancedShapeEngine = True

Dim TestActiveSheet As GrapeCity.Spreadsheet.IWorksheet = FpSpread1.AsWorkbook().ActiveSheet

Dim picture1 As GrapeCity.Spreadsheet.Drawing.IShape = TestActiveSheet.Shapes.AddPicture("D:\apple.jpg", False, False, 50, 50, 400, 400)
picture1.AutoShapeType = GrapeCity.Spreadsheet.Drawing.AutoShapeType.Rectangle

Dim picture2 As GrapeCity.Spreadsheet.Drawing.IShape = TestActiveSheet.Shapes.AddPicture("D:\apple.jpg", True, True, 200, 200, 400, 400)
picture2.AutoShapeType = GrapeCity.Spreadsheet.Drawing.AutoShapeType.Rectangle

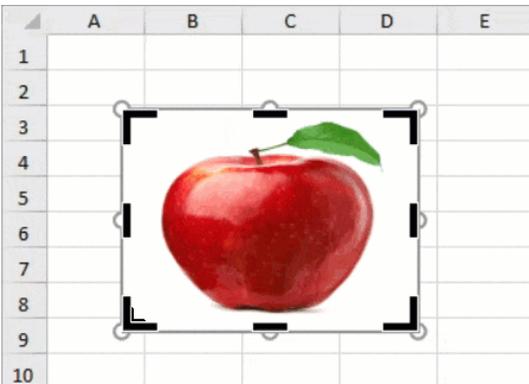
picture2.PictureFormat.Crop.PictureOffsetX = 100
picture2.PictureFormat.Crop.PictureOffsetY = 100

picture2.PictureFormat.Crop.ShapeLeft = 150
picture2.PictureFormat.Crop.ShapeTop = 150
```

Using Runtime UI

You can enable UI crop handles in runtime to crop images in a worksheet. The black crop handles appear on the edges and corners of an image when using the **FpSpread.StartCropping** ('**StartCropping Method**' in the **on-line documentation**) method.

You can also add a margin around a picture by dragging the cropping handles outward.



When you are done with the cropping, press Escape or click anywhere outside the picture within the sheet.

The following code example shows how to use the runtime UI crop in a Spread worksheet.

C#

```
// Cropping a picture at runtime
fpSpread1.LegacyBehaviors = FarPoint.Win.Spread.LegacyBehaviors.None;
fpSpread1.Features.EnhancedShapeEngine = true;

GrapeCity.Spreadsheet.IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;

GrapeCity.Spreadsheet.Drawing.IPicture pic1 = (GrapeCity.Spreadsheet.Drawing.IPicture)TestActiveSheet.Shapes.AddPicture(@"D:\apple.jpg", false,
false, 50, 50, 200, 150);
bool started = fpSpread1.StartCropping(pic1);
```

Visual Basic

```
'Cropping a picture at runtime
FpSpread1.LegacyBehaviors = FarPoint.Win.Spread.LegacyBehaviors.None
FpSpread1.Features.EnhancedShapeEngine = True

Dim TestActiveSheet As GrapeCity.Spreadsheet.IWorksheet = FpSpread1.AsWorkbook().ActiveSheet

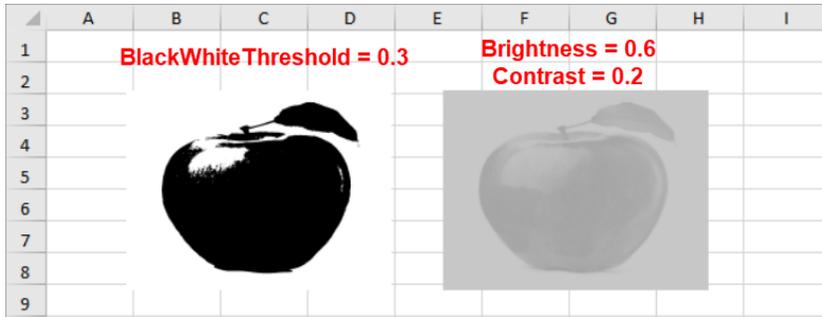
Dim pic1 As GrapeCity.Spreadsheet.Drawing.IPicture = CType(TestActiveSheet.Shapes.AddPicture("D:\apple.jpg", False, False, 50, 50, 200, 150),
GrapeCity.Spreadsheet.Drawing.IPicture)
Dim started As Boolean = FpSpread1.StartCropping(pic1)
```

Format Images

You can format the images in a worksheet according to your requirements with the help of the available picture format options in Spread for Winforms.

The **IPictureFormat** ('**IPictureFormat Interface**' in the **on-line documentation**) interface consists of formatting options such as brightness, contrast, black and white threshold, and transparency which can be used to enhance an image.

The following example shows an image customized with two different enhancement options.



C#

```
fpSpread1.Features.EnhancedShapeEngine = true;

IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;
GrapeCity.Spreadsheet.Drawing.IShape picture1 = TestActiveSheet.Shapes.AddPicture(@"D:\apple.jpg", false, false, 60, 45, 200, 150);

picture1.PictureFormat.ColorType = GrapeCity.Spreadsheet.Drawing.PictureColorType.BlackAndWhite;
picture1.PictureFormat.BlackWhiteThreshold = 0.3;

GrapeCity.Spreadsheet.Drawing.IShape picture2 = TestActiveSheet.Shapes.AddPicture(@"D:\apple.jpg", false, false, 300, 45, 200, 150);

picture2.PictureFormat.Brightness = 0.6;
picture2.PictureFormat.Contrast = 0.2;
```

Visual Basic

```
' Cropping a picture using code
FpSpread1.LegacyBehaviors = FarPoint.Win.Spread.LegacyBehaviors.None
FpSpread1.Features.EnhancedShapeEngine = True

Dim TestActiveSheet As GrapeCity.Spreadsheet.IWorksheet = FpSpread1.AsWorkbook().ActiveSheet

Dim picture1 As GrapeCity.Spreadsheet.Drawing.IShape = TestActiveSheet.Shapes.AddPicture("D:\apple.jpg", False, False, 50, 50, 400, 400)
picture1.AutoShapeType = GrapeCity.Spreadsheet.Drawing.AutoShapeType.Rectangle

Dim picture2 As GrapeCity.Spreadsheet.Drawing.IShape = TestActiveSheet.Shapes.AddPicture("D:\apple.jpg", True, True, 200, 200, 400, 400)
picture2.AutoShapeType = GrapeCity.Spreadsheet.Drawing.AutoShapeType.Rectangle

picture2.PictureFormat.Crop.PictureOffsetX = 100
picture2.PictureFormat.Crop.PictureOffsetY = 100

picture2.PictureFormat.Crop.ShapeLeft = 150
picture2.PictureFormat.Crop.ShapeTop = 150
```

For information on how to create images of the specified data in a Spread worksheet, refer to **Saving to an Image File**.

Picture Adjustments

You can customize an image in a worksheet by adding various artistic effects, and transform any image into a sketch or a painting.

The following picture adjustments take effect for painting in Spread.

- **Blur:** Adds a blur effect.
- **Brightness contrast:** Adjusts the brightness and contrast of the image.
- **Color tone:** Adjusts the brightness and color temperature of the image.
- **Film grain:** Applies a grainy effect to the image.
- **Glow edges:** Adds a glowing texture to the image edges.
- **Saturation:** Adds saturation effect to the image.
- **Sharpness soften:** Sharpens and softens the image.
- **Recolor:** Changes the color of a picture.

The following table provides a comparison between the original image and images under different picture adjustments.





Glow edges



Blur



Recolor



The **PictureEffectType** enumeration provides various options for defining picture effects.

The following code shows how to apply an artistic effect using the **PictureEffectType** enumeration:

C#

```
IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;
fpSpread1.AsWorkbook().WorkbookSet.CalculationEngine.CalcFeatures = GrapeCity.Spreadsheet.CalcFeatures.All;
fpSpread1.Features.EnhancedShapeEngine = true;
GrapeCity.Spreadsheet.Drawing.IShape picture1 = TestActiveSheet.Shapes.AddPicture(@"D:\logo.png", true, true, 40, 20, 150, 150);
GrapeCity.Spreadsheet.Drawing.IShape picture2 = TestActiveSheet.Shapes.AddPicture(@"D:\logo.png", true, true, 300, 20, 150, 150);
GrapeCity.Spreadsheet.Drawing.IPictureEffects pic = picture2.Fill.PictureEffects;
// Add brightness contrast effect
GrapeCity.Spreadsheet.Drawing.IPictureEffect pics2 = pic.Insert(GrapeCity.Spreadsheet.Drawing.PictureEffectType.BrightnessContrast);
pics2.EffectParameters[0].Value = 0.2d;
pics2.EffectParameters[1].Value = -0.9d;
```

VB

```
Dim TestActiveSheet As IWorksheet = fpSpread1.AsWorkbook().ActiveSheet
fpSpread1.AsWorkbook().WorkbookSet.CalculationEngine.CalcFeatures = GrapeCity.Spreadsheet.CalcFeatures.All
fpSpread1.Features.EnhancedShapeEngine = True
Dim picture1 As GrapeCity.Spreadsheet.Drawing.IShape = TestActiveSheet.Shapes.AddPicture("D:\logo.png", True, True, 40, 20, 150, 150)
Dim picture2 As GrapeCity.Spreadsheet.Drawing.IShape = TestActiveSheet.Shapes.AddPicture("D:\logo.png", True, True, 300, 20, 150, 150)
Dim pic As GrapeCity.Spreadsheet.Drawing.IPictureEffects = picture2.Fill.PictureEffects
Dim pics2 As GrapeCity.Spreadsheet.Drawing.IPictureEffect = pic.Insert(GrapeCity.Spreadsheet.Drawing.PictureEffectType.BrightnessContrast)
pics2.EffectParameters(0).Value = 0.2R
pics2.EffectParameters(1).Value = -0.9R
```

You can also change the color of a picture in the worksheet using the **Recolor** method of the **IPictureFormat (IPictureFormat Interface' in the on-line documentation)** interface.

C#

```
fpSpread1.Features.EnhancedShapeEngine = true;
IWorksheet TestActiveSheet = fpSpread1.AsWorkbook().ActiveSheet;
fpSpread1.AsWorkbook().WorkbookSet.CalculationEngine.CalcFeatures = GrapeCity.Spreadsheet.CalcFeatures.All;
fpSpread1.Features.EnhancedShapeEngine = true;
GrapeCity.Spreadsheet.Drawing.IShape picture1 = TestActiveSheet.Shapes.AddPicture(@"D:\logo.png", true, true, 40, 20, 150, 150);
GrapeCity.Spreadsheet.Drawing.IShape picture2 = TestActiveSheet.Shapes.AddPicture(@"D:\logo.png", true, true, 300, 20, 150, 150);
picture1.PictureFormat.Recolor(GrapeCity.Core.SchemeColor.CreateArgbColor(0, 246, 243), true);
```

VB

```
fpSpread1.Features.EnhancedShapeEngine = True
Dim TestActiveSheet As IWorksheet = fpSpread1.AsWorkbook().ActiveSheet
fpSpread1.AsWorkbook().WorkbookSet.CalculationEngine.CalcFeatures = GrapeCity.Spreadsheet.CalcFeatures.All
fpSpread1.Features.EnhancedShapeEngine = True
Dim picture1 As GrapeCity.Spreadsheet.Drawing.IShape = TestActiveSheet.Shapes.AddPicture("D:\logo.png", True, True, 40, 20, 150, 150)
Dim picture2 As GrapeCity.Spreadsheet.Drawing.IShape = TestActiveSheet.Shapes.AddPicture("D:\logo.png", True, True, 300, 20, 150, 150)
picture1.PictureFormat.Recolor(GrapeCity.Core.SchemeColor.CreateArgbColor(0, 246, 243), True)
```

You can also restore the original colors of a picture that was re-colored using the **RestoreOriginalColors** method.

Note: As of now, Spread for WinForms only supports the above-mentioned picture effects. We will support more effects in the future releases.

Touch Support with the Component

Spread supports touch gestures in many areas of the control. You can use touch gestures with filtering, grouping, sorting, and with many other types of interactions in Spread. A touch screen is required (either a touch monitor or a smartbook-type laptop with a touch screen).

The following topics provide information about touch support and the areas where touch support is available:

- **Understanding Touch Messages**
- **Using a Touch Keyboard**
- **Using the Touch Menu Bar**
- **Using Touch Support**

Understanding Touch Messages

Touch messages are processed in cell areas (Tap, Panning, Pinch, and so on); however, in header and footer areas (column header, row header, corner, and column footer) and the scroll bar area, touch messages are treated as mouse messages.

For example, a panning operation on the column header becomes a column selection action (similar to using the mouse). Spread does not scroll.

FpSpread provides an **InputDeviceType** (**'InputDeviceType Property' in the on-line documentation**) property that returns the message's device type.

Using a Touch Keyboard

You can display a touch keyboard when editing a cell.

Use the **ShowTouchKeyboard** (**'ShowTouchKeyboard Method' in the on-line documentation**) method to display the keyboard as in the following image.



You can also specify whether the cell being edited scrolls into view when the touch keyboard is displayed by setting the **AutoScrollWhenKeyboardShowing** (**'AutoScrollWhenKeyboardShowing Property' in the on-line documentation**) property. FpSpread will scroll up as soon as possible, but if there is not enough space to scroll, the active cell may not be visible. For example, if the control is completely covered by the touch keyboard, the control scrolls the active cell to the first row (cell still hidden by keyboard).

FpSpread provides an **InputScope** (**'InputScope Property' in the on-line documentation**) property that can be used to specify the touch keyboard's layout.

You can use the **ShowTouchKeyboard** (**'ShowTouchKeyboard Method' in the on-line documentation**) and **HideTouchKeyboard** (**'HideTouchKeyboard Method' in the on-line documentation**) methods in the **EditModeOn** (**'EditModeOn Event' in the on-line documentation**) event to show the touch keyboard when the

cell goes into edit mode.

Using Code

The following example displays the touch keyboard when the cell is in edit mode and hides it when the cell is no longer in edit mode.

C#

```
private void Form1_Load(object sender, EventArgs e)
{
    fpSpread1.AutoScrollWhenKeyboardShowing = true;
}
private void fpSpread1_EditModeOn(object sender, EventArgs e)
{
    fpSpread1.ShowTouchKeyboard();
}
private void fpSpread1_EditModeOff(object sender, EventArgs e)
{
    fpSpread1.HideTouchKeyboard();
}
```

Visual Basic

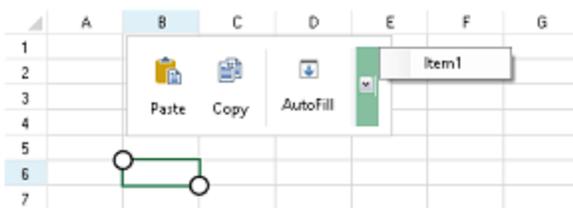
```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    FpSpread1.AutoScrollWhenKeyboardShowing = True
End Sub
Private Sub FpSpread1_EditModeOn(sender As Object, e As System.EventArgs) Handles
FpSpread1.EditModeOn
    FpSpread1.ShowTouchKeyboard()
End Sub
Private Sub FpSpread1_EditModeOff(sender As Object, e As System.EventArgs) Handles
FpSpread1.EditModeOff
    FpSpread1.HideTouchKeyboard()
End Sub
```

Using the Touch Menu Bar

You can use the default touch menu bar or touch strip to cut, copy, and paste cells. You can also customize the touch strip to provide additional options.

Tap a selected range to display the touch menu bar strip.

You can use the **TouchStripOpening ('TouchStripOpening Event' in the on-line documentation)** event to display a customized touch strip. You can also add menu items to the touch strip.



Using Code

You can add a drop-down menu item with following code. This example also hides the "Cut" option in the touch strip.

1. Create a customized touch strip item.
2. Create and add a menu item
3. Add the new items to the touch strip.
4. Cancel the default touch strip in the **TouchStripOpening** ('TouchStripOpening Event' in the on-line **documentation**) event.

C#

```
private FarPoint.Win.Spread.CellTouchStrip touchStripwithdropdownmenu;

private void Form1_Load(object sender, EventArgs e)
{
    // Remove Cut from default touch menu bar
    touchStripwithdropdownmenu = new
FarPoint.Win.Spread.CellTouchStrip(this.fpSpread1);
    touchStripwithdropdownmenu.Items["Cut"].Visible = false;
    ToolStripSeparator separator1 = new ToolStripSeparator();

    // Add AutoFill to touch menu bar
    FarPoint.Win.Spread.TouchStripButton autoFill2 = new
FarPoint.Win.Spread.TouchStripButton("AutoFill", Properties.Resources.AutoFill);
    autoFill2.Click += autoFill_Click;
    ToolStripSeparator separator2 = new ToolStripSeparator();

    // Add drop down menu to touch menu bar
    ToolStripDropDownButton dropDownMenu = new
ToolStripDropDownButton(Properties.Resources.TouchMenuItemDownArrow);
    dropDownMenu.ShowDropDownArrow = false;
    dropDownMenu.ImageScaling = ToolStripItemImageScaling.None;
    ContextMenuStrip menu = new System.Windows.Forms.ContextMenuStrip();
    ToolStripMenuItem newcontitem1 = new ToolStripMenuItem();
    newcontitem1.Text = "Copy";
    newcontitem1.Click += newcontitem1_Click;
    newcontitem1.AutoSize = false;
    newcontitem1.Height = 30;
    newcontitem1.Width = 100;
    newcontitem1.TextAlign = ContentAlignment.MiddleLeft;
    menu.Items.Add(newcontitem1);
    ToolStripMenuItem newcontitem2 = new ToolStripMenuItem();
    newcontitem2.Text = "Paste";
    newcontitem2.Click += newcontitem2_Click;
    newcontitem2.AutoSize = false;
    newcontitem2.Height = 30;
    newcontitem2.Width = 100;
    newcontitem2.TextAlign = ContentAlignment.MiddleLeft;
    menu.Items.Add(newcontitem2);
    ToolStripMenuItem newcontitem3 = new ToolStripMenuItem();
    newcontitem3.Text = "Clear";
    newcontitem3.Click += newcontitem3_Click;
    newcontitem3.AutoSize = false;
    newcontitem3.Height = 30;
    newcontitem3.Width = 100;
    newcontitem3.TextAlign = ContentAlignment.MiddleLeft;
    menu.Items.Add(newcontitem3);
    dropDownMenu.DropDown = menu;

    touchStripwithdropdownmenu.Items.AddRange(new ToolStripItem[] { separator1,
```

```
autoFill2, separator2, dropDownMenu });

    // Enable AutoFill
    fpSpread1.AllowDragFill = true;
}

void autoFill_Click(object sender, EventArgs e)
{
    FarPoint.Win.Spread.SpreadView activeView =
fpSpread1.GetRootWorkbook().GetActiveWorkbook();
    if (activeView != null)
    {
        activeView.ShowAutoFillIndicator();
    }
}

void newcontitem1_Click(object sender, EventArgs e)
{
    // Copy
    fpSpread1.ActiveSheet.ClipboardCopy();
}

void newcontitem2_Click(object sender, EventArgs e)
{
    // Paste

fpSpread1.ActiveSheet.ClipboardPaste(FarPoint.Win.Spread.ClipboardPasteOptions.Values);
}

void newcontitem3_Click(object sender, EventArgs e)
{
    // Clear
    int r1 = fpSpread1.ActiveSheet.Models.Selection.AnchorRow;
    int c1 = fpSpread1.ActiveSheet.Models.Selection.AnchorColumn;
    int r2 = fpSpread1.ActiveSheet.Models.Selection.LeadRow - r1 + 1;
    int c2 = fpSpread1.ActiveSheet.Models.Selection.LeadColumn - c1 + 1;
    FarPoint.Win.Spread.Model.DefaultSheetDataModel dataModel =
(FarPoint.Win.Spread.Model.DefaultSheetDataModel) fpSpread1.ActiveSheet.Models.Data;
    dataModel.ClearData(r1, c1, r2, c2);
}

void fpSpread1_TouchStripOpening(object sender,
FarPoint.Win.Spread.TouchStripOpeningEventArgs e)
{
    // Hide default touch menu bar
    e.Cancel = true;

    // Show customized touch menu bar
    touchStripwithdropdownmenu.Show(new Point(e.X - 20, e.Y - 35 -
touchStripwithdropdownmenu.Height));
}
```

Visual Basic

```
Private touchStripwithdropdownmenu As FarPoint.Win.Spread.CellTouchStrip

Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
```

```
' Remove Cut from default touch menu bar
touchStripwithdropdownmenu = New FarPoint.Win.Spread.CellTouchStrip(Me.FpSpread1)
touchStripwithdropdownmenu.Items("Cut").Visible = False

Dim separator1 As New ToolStripSeparator()

' Add AutoFill to touch menu bar
Dim autoFill2 As New FarPoint.Win.Spread.TouchStripButton("AutoFill",
My.Resources.AutoFill)
AddHandler autoFill2.Click, AddressOf autoFill_Click

Dim separator2 As New ToolStripSeparator()

' Add drop down menu to touch menu bar
Dim dropDownMenu As New
ToolStripDropDownButton(My.Resources.Resources.TouchMenuItemDownArrow)
dropDownMenu.ShowDropDownArrow = False
dropDownMenu.ImageScaling = ToolStripItemImageScaling.None
Dim menu As ContextMenuStrip = New System.Windows.Forms.ContextMenuStrip()
Dim newcontitem1 As New ToolStripMenuItem()
newcontitem1.Text = "Copy"
AddHandler newcontitem1.Click, AddressOf newcontitem1_Click
newcontitem1.AutoSize = False
newcontitem1.Height = 30
newcontitem1.Width = 100
newcontitem1.TextAlign = ContentAlignment.MiddleLeft
menu.Items.Add(newcontitem1)
Dim newcontitem2 As New ToolStripMenuItem()
newcontitem2.Text = "Paste"
AddHandler newcontitem2.Click, AddressOf newcontitem2_Click
newcontitem2.AutoSize = False
newcontitem2.Height = 30
newcontitem2.Width = 100
newcontitem2.TextAlign = ContentAlignment.MiddleLeft
menu.Items.Add(newcontitem2)
Dim newcontitem3 As New ToolStripMenuItem()
newcontitem3.Text = "Clear"
AddHandler newcontitem3.Click, AddressOf newcontitem3_Click
newcontitem3.AutoSize = False
newcontitem3.Height = 30
newcontitem3.Width = 100
newcontitem3.TextAlign = ContentAlignment.MiddleLeft
menu.Items.Add(newcontitem3)
dropDownMenu.DropDown = menu

touchStripwithdropdownmenu.Items.AddRange(New ToolStripItem() {separator1,
autoFill2, separator2, dropDownMenu})

' Enable AutoFill
FpSpread1.AllowDragFill = True
End Sub

Private Sub autoFill_Click(sender As Object, e As EventArgs)
Dim activeView As FarPoint.Win.Spread.SpreadView =
fpSpread1.GetRootWorkbook().GetActiveWorkbook()
If activeView IsNot Nothing Then
activeView.ShowAutoFillIndicator()
```

```
End If
End Sub

Private Sub newcontitem1_Click(sender As Object, e As EventArgs)
    ' Copy
    fpSpread1.ActiveSheet.ClipboardCopy()
End Sub

Private Sub newcontitem2_Click(sender As Object, e As EventArgs)
    ' Paste

    fpSpread1.ActiveSheet.ClipboardPaste(FarPoint.Win.Spread.ClipboardPasteOptions.Values)
End Sub

Private Sub newcontitem3_Click(sender As Object, e As EventArgs)
    ' Clear
    Dim r1 As Integer = fpSpread1.ActiveSheet.Models.Selection.AnchorRow
    Dim c1 As Integer = fpSpread1.ActiveSheet.Models.Selection.AnchorColumn
    Dim r2 As Integer = fpSpread1.ActiveSheet.Models.Selection.LeadRow - r1 + 1
    Dim c2 As Integer = fpSpread1.ActiveSheet.Models.Selection.LeadColumn - c1 + 1
    Dim dataModel As FarPoint.Win.Spread.Model.DefaultSheetDataModel =
    DirectCast(fpSpread1.ActiveSheet.Models.Data,
    FarPoint.Win.Spread.Model.DefaultSheetDataModel)
    dataModel.ClearData(r1, c1, r2, c2)
End Sub

Private Sub FpSpread1_TouchStripOpening(sender As Object, e As
FarPoint.Win.Spread.TouchStripOpeningEventArgs) Handles FpSpread1.TouchStripOpening
    ' Hide default touch menu bar
    e.Cancel = True

    ' Show customized touch menu bar
    touchStripwithdropdownmenu.Show(New Point(e.X - 20, e.Y - 35 -
touchStripwithdropdownmenu.Height))
End Sub
```

Using Touch Support

You can use touch support in many areas and in many types of interactions with the Spread control.

The following topics explain where touch support is available:

- **Using a Touch Keyboard**
- **Using the Touch Menu Bar**
- **Using Touch Support with AutoFit**
- **Using Touch Support with Cell Notes**
- **Using Touch Support with Charts**
- **Using Touch Support with Clipboard Operations**
- **Using Touch Support with Drag and Fill**
- **Using Touch Support with Drop-Down Elements**
- **Using Touch Support with Editable Cells**
- **Using Touch Support with InputMan Cells**
- **Using Touch Support with Filtering**
- **Using Touch Support with Grouping**

- **Using Touch Support with Range Grouping**
- **Using Touch Support when Moving Columns or Rows**
- **Using Touch Support when Resizing Columns or Rows**
- **Using Touch Support with Scrolling**
- **Using Touch Support with Selections**
- **Using Touch Support with Shapes**
- **Using Touch Support when Sorting**
- **Using Touch Support with the Tab Strip**
- **Using Touch Support with Viewports**
- **Using Touch Support with Zooming**

Using Touch Support with AutoFit

You can use touch support gestures with automatic fit.

Tap to select a column (resize handler becomes visible). Double-tap to resize the column automatically. Tap to select a row and double-tap to resize the row automatically. The **Resizable ('Resizable Property' in the on-line documentation)** property must be true for the column and row.

Using Touch Support with Cell Notes

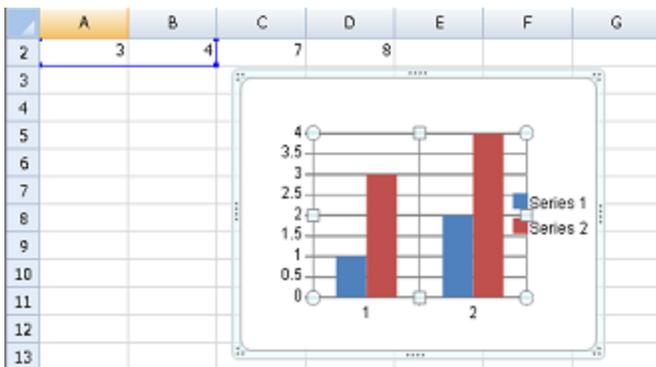
You can use touch gestures with editable cell notes.

Tap a cell note to select it. Double-tap the cell note to edit it. Press the edge of the cell note and slide the note to move it.

Set the **CanMove ('CanMove Property' in the on-line documentation)** and **CanSize ('CanSize Property' in the on-line documentation)** properties to true in order to move or resize the cell note. The **NoteStyle ('NoteStyle Property' in the on-line documentation)** property must be set to **StickyNote** for touch support.

Using Touch Support with Charts

You can use touch gestures with the Chart control.



The Chart control uses the following touch gestures:

Touch Gesture	Mouse Action	Action
Tap	Click	Selects a cell note, shape, or chart.
Double-tap	Double-Click	Edits a cell note, shape, or chart if editing is supported.

Touch Gesture

Press edge then slide

Press chart then slide

Press rotated handle and slide

Mouse Action

Press left button on edge then move

Press left button on chart then move

Press left button on rotated handle and move

Action

Resizes a cell note, shape, or chart if CanSize is set to true.

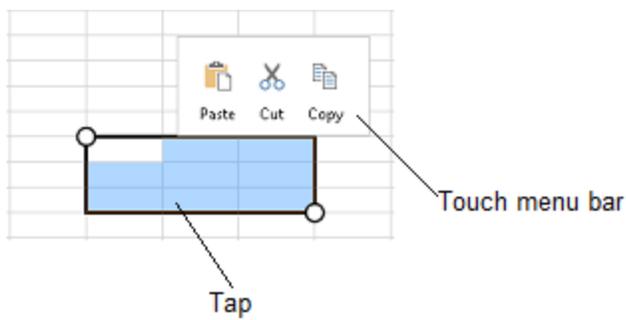
Moves a cell note, shape, or chart if CanMove is set to true.

Rotates a shape or chart if CanRotate is set to true.

Using Touch Support with Clipboard Operations

You can use touch gestures and the touch menu bar to cut, copy, and paste.

Select a range of cells. Tap the selected range to display the touch menu bar options. Tap the **Cut**, **Copy**, or **Paste** menu items.

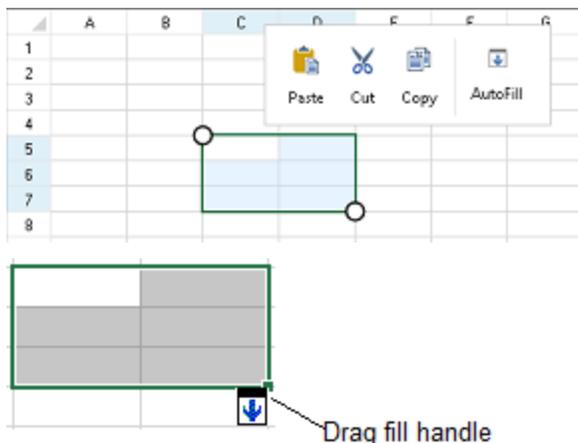


The **ClipboardOptions** ('ClipboardOptions Property' in the on-line documentation) property specifies what areas are part of the cut, copy, or paste.

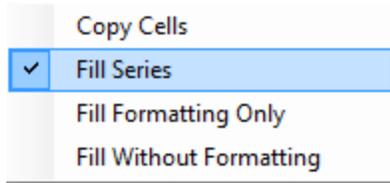
Using Touch Support with Drag and Fill

You can use touch support gestures and the touch menu bar or toolbar with drag and fill.

Select a range. Tap the range to display the touch menu bar. Tap the **AutoFill** menu item to display the drag fill handle at the bottom-right edge of the selected range. Press and slide the handle to drag and fill the range. The **AllowDragFill** ('AllowDragFill Property' in the on-line documentation) property must be true to display and use the drag fill handle.



The following image shows the Drag Fill Drop Down Context Menu:



Using Code

This example adds the drag fill icon to the touch menu bar.

1. Set the **AllowDragFill ('AllowDragFill Property' in the on-line documentation)** property to true.
2. Create a new touch strip button and separator in the **TouchStripOpening ('TouchStripOpening Event' in the on-line documentation)** event.
3. Create an image for the new button.
4. Add the new items to the touch strip.
5. Create and use an event to display the auto fill indicator.

CS

```
void autoFill_Click(object sender, EventArgs e)
{
    FarPoint.Win.Spread.SpreadView activeView =
    fpSpread1.GetRootWorkbook().GetActiveWorkbook();
    if (activeView != null)
    {
        activeView.ShowAutoFillIndicator();
    }
}

private void Form1_Load(object sender, EventArgs e)
{
    fpSpread1.AllowDragFill = true;
}

private void fpSpread1_TouchStripOpening(object sender,
FarPoint.Win.Spread.TouchStripOpeningEventArgs e)
{
    e.Cancel = true;
    FarPoint.Win.Spread.CellTouchStrip touchStrip = new
FarPoint.Win.Spread.CellTouchStrip(fpSpread1);
    ToolStripSeparator separator = new ToolStripSeparator();
    FarPoint.Win.Spread.TouchStripButton autoFill = new
FarPoint.Win.Spread.TouchStripButton("AutoFill",
System.Drawing.Image.FromFile("C:\\SpreadWin7\\dragfill.png"));
    autoFill.Click += autoFill_Click;
    touchStrip.Items.AddRange(new ToolStripItem[] { separator, autoFill });
    touchStrip.Show(new Point(e.X - 20, e.Y - 35 - touchStrip.Height));
}
}
```

VB

```
Private Sub autoFill_Click(sender As Object, e As EventArgs)
    Dim activeView As FarPoint.Win.Spread.SpreadView =
    fpSpread1.GetRootWorkbook().GetActiveWorkbook()
    If activeView IsNot Nothing Then
```

```

        activeView.ShowAutoFillIndicator()
    End If
End Sub

Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    fpSpread1.AllowDragFill = True
End Sub
Private Sub fpSpread1_TouchStripOpening(sender As Object, e As
FarPoint.Win.Spread.TouchStripOpeningEventArgs)
    e.Cancel = True
    Dim touchStrip As New FarPoint.Win.Spread.CellTouchStrip(fpSpread1)
    Dim separator As New ToolStripSeparator()
    Dim autoFill As New FarPoint.Win.Spread.TouchStripButton("AutoFill",
System.Drawing.Image.FromFile("C:\SpreadWin7\dragfill.png"))
    AddHandler autoFill.Click, AddressOf autoFill_Click
    touchStrip.Items.AddRange(New ToolStripItem() {separator, autoFill})
    touchStrip.Show(New Point(e.X - 20, e.Y - 35 - touchStrip.Height))
End Sub

```

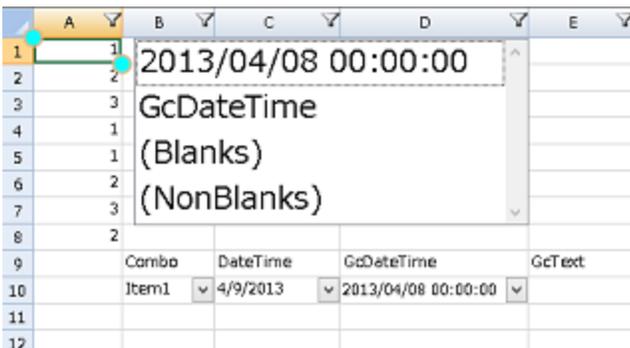
Using Touch Support with Drop-Down Elements

You can use touch gestures in drop-down cells, calendars, and other elements in the control.

The following items are drop-down elements or windows:

- FilterDropDown (Gadget)
- FilterDropDown (FilterBar's DropDown)
- DropDownList (ComboCellType)
- DropDownCalendar (DateTimeCellType)
- DropDownCalendar (GcDateTimeCellType)
- DropDownEdit (GcTextCellType)

Set the **TouchDropDownScale ('TouchDropDownScale Property' in the on-line documentation)** property to change the scale of the elements in the drop-down window. The default value is 1.5. The following drop-down filter has a scale of 2.



Note: If the **TouchDropDownScale ('TouchDropDownScale Property' in the on-line documentation)** property is set to 0, all drop-down windows are zoomed in mouse or touch mode. If the **TouchDropDownScale ('TouchDropDownScale Property' in the on-line documentation)** property is set to a valid value (1f to 4f), then padding is increased for touch support based on the property value.

Using Code

Set the **TouchDropDownScale** (**'TouchDropDownScale Property' in the on-line documentation**) property to change the scale of the drop-down window.

CS

```
fpSpread1.TouchDropDownScale = 1.0F;
fpSpread1.TouchSelectionGripperBackColor = Color.Aqua;
fpSpread1.TouchSelectionGripperLineColor = Color.BurlyWood;
fpSpread1.TouchSelectionGripperThickness = 2;
```

VB

```
fpSpread1.TouchDropDownScale = 1.0F
fpSpread1.TouchSelectionGripperBackColor = Color.Aqua
fpSpread1.TouchSelectionGripperLineColor = Color.BurlyWood
fpSpread1.TouchSelectionGripperThickness = 2
```

Using Touch Support with Editable Cells

You can use touch gestures to edit cells that allow editing.

Double-tap a cell to go into edit mode. Tap a cell to go into edit mode if the **EditModePermanent** (**'EditModePermanent Property' in the on-line documentation**) property is true. Typing a character in the cell also starts edit mode.

Button, check box, multiple option, hyperlink, slider, and filter bar cells use the following touch gestures:

Touch Gesture

Tap

Double-tap

Press and slide

Mouse Action

Click

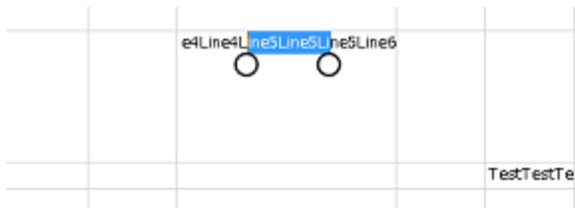
Double-click

Press left mouse button and move

GcTextBox, GcDateTime, number, regular expression, percent, currency, date time, general, and text cells have similar touch behaviors.

List box and rich text cells support touch gestures similar to standard controls.

If the **Editable** property is true for the combo box and multiple-column combo box cells, the gripper is displayed. The following image displays a gripper in the cell.



Using Code

You can set the **ShowGrippersInEditingStatus** (**'ShowGrippersInEditingStatus Property' in the on-line documentation**) property to true to display a gripper while the cell is in edit mode.

CS

```
private void Form1_Load(object sender, EventArgs e)
{
```

```

        fpSpread1.ShowGrippersInEditingStatus = true;
    }

    private void button1_Click(object sender, EventArgs e)
    {
        fpSpread1.ShowGrippersInEditingStatus = false;
    }

    private void fpSpread1_ShowGrippersInEditingStatusChanged(object sender, EventArgs
e)
    {
        listBox1.Items.Add("Status Changed");
    }

```

VB

```

Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    fpSpread1.ShowGrippersInEditingStatus = True
End Sub

Private Sub fpSpread1_ShowGrippersInEditingStatusChanged(sender As Object, e As
EventArgs) Handles fpSpread1.ShowGrippersInEditingStatusChanged
    ListBox1.Items.Add("Status Changed")
End Sub

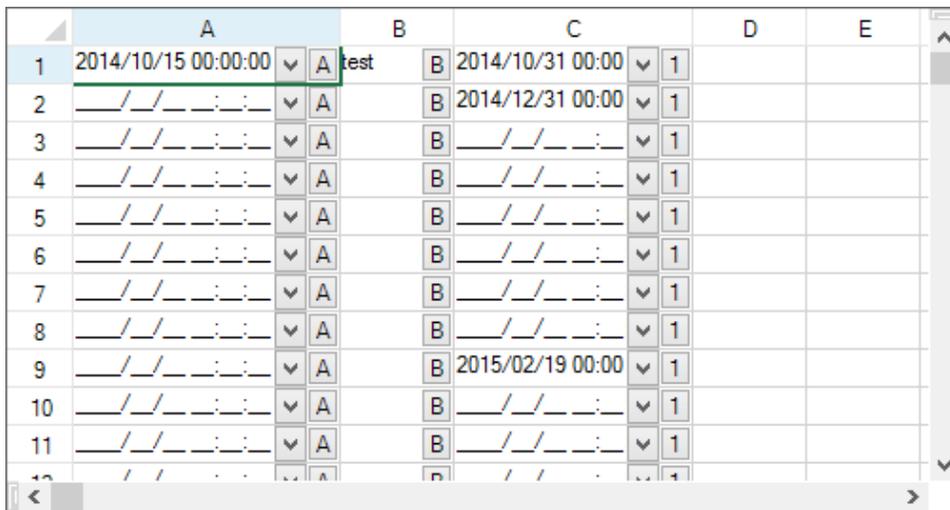
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    fpSpread1.ShowGrippersInEditingStatus = False
End Sub

```

Using Touch Support with InputMan Cells

You can use touch support with GcDateTime and GcTextBox cells.

You can tap side buttons in the cells to change the cell values.



Using Code

The following example creates side buttons for GcDateTime cells and a GcTextBox cell. The **DropDownOpening** event has a

ByTouch property that returns whether the drop-down button was opened with a touch gesture.

CS

```
GrapeCity.Win.Spread.InputMan.CellType.SideButtonInfo testbutton = new
GrapeCity.Win.Spread.InputMan.CellType.SideButtonInfo();
testbutton.Behavior = GrapeCity.Win.Spread.InputMan.CellType.SideButtonBehavior.SpinDown;
testbutton.Delay = 300;
testbutton.Interval = 5;
testbutton.Text = "1";

GrapeCity.Win.Spread.InputMan.CellType.GcDateTimeCellType dateCellType = new
GrapeCity.Win.Spread.InputMan.CellType.GcDateTimeCellType();
dateCellType.SideButtons.Add(new GrapeCity.Win.Spread.InputMan.CellType.SideButtonInfo() {
Text = "A" });
this.fpSpread1_Sheet1.Columns[0].CellType = dateCellType;

GrapeCity.Win.Spread.InputMan.CellType.GcTextBoxCellType textCellType = new
GrapeCity.Win.Spread.InputMan.CellType.GcTextBoxCellType();
textCellType.SideButtons.Add(new GrapeCity.Win.Spread.InputMan.CellType.SideButtonInfo() {
Text = "B" });
this.fpSpread1_Sheet1.Columns[1].CellType = textCellType;

GrapeCity.Win.Spread.InputMan.CellType.GcDateTimeCellType dateCellType2 = new
GrapeCity.Win.Spread.InputMan.CellType.GcDateTimeCellType();
dateCellType2.SideButtons.Add(testbutton);
this.fpSpread1_Sheet1.Columns[2].CellType = dateCellType2;

void IMCellType_DropDownOpening(object sender,
GrapeCity.Win.Spread.InputMan.CellType.DropDownOpeningEventArgs e)
{
    listBox1.Items.Add(e.ByTouch.ToString());
}

private void fpSpread1_EditModeOn(object sender, EventArgs e)
{
    if (fpSpread1.EditingControl is
GrapeCity.Win.Spread.InputMan.CellType.GcDateTime)
((GrapeCity.Win.Spread.InputMan.CellType.GcDateTime) fpSpread1.EditingControl).DropDownOpening
+= new EventHandler(IMCellType_DropDownOpening);
}

private void fpSpread1_EditModeOff(object sender, EventArgs e)
{
    if (fpSpread1.EditingControl is
GrapeCity.Win.Spread.InputMan.CellType.GcDateTime)
((GrapeCity.Win.Spread.InputMan.CellType.GcDateTime) fpSpread1.EditingControl).DropDownOpening
-= new EventHandler(IMCellType_DropDownOpening);
}
```

VB

```
Dim testbutton As New GrapeCity.Win.Spread.InputMan.CellType.SideButtonInfo()
testbutton.Behavior = GrapeCity.Win.Spread.InputMan.CellType.SideButtonBehavior.SpinDown
testbutton.Delay = 300
testbutton.Interval = 5
testbutton.Text = "1"
```

```
Dim dateCellType As New GrapeCity.Win.Spread.InputMan.CellType.GcDateTimeCellType()
dateCellType.SideButtons.Add(New GrapeCity.Win.Spread.InputMan.CellType.SideButtonInfo()
With {.Text = "A"})
fpSpread1_Sheet1.Columns(0).CellType = dateCellType

Dim textCellType = New GrapeCity.Win.Spread.InputMan.CellType.GcTextBoxCellType()
textCellType.SideButtons.Add(New GrapeCity.Win.Spread.InputMan.CellType.SideButtonInfo()
With {.Text = "B"})
fpSpread1_Sheet1.Columns(1).CellType = textCellType

Dim dateCellType2 As New GrapeCity.Win.Spread.InputMan.CellType.GcDateTimeCellType()
dateCellType2.SideButtons.Add(testbutton)
fpSpread1_Sheet1.Columns(2).CellType = dateCellType2

Private Sub IMCellType_DropDownOpening(ByVal sender As Object, ByVal e As
GrapeCity.Win.Spread.InputMan.CellType.DropDownOpeningEventArgs)
    ListBox1.Items.Add(e.ByTouch.ToString())
End Sub

Private Sub fpSpread1_EditModeOff(sender As Object, e As EventArgs) Handles
fpSpread1.EditModeOff
    If TypeOf (fpSpread1.EditingControl) Is
GrapeCity.Win.Spread.InputMan.CellType.GcDateTime Then
        RemoveHandler CType(fpSpread1.EditingControl,
GrapeCity.Win.Spread.InputMan.CellType.GcDateTime).DropDownOpening, AddressOf
IMCellType_DropDownOpening
    End If
End Sub

Private Sub fpSpread1_EditModeOn(sender As Object, e As EventArgs) Handles
fpSpread1.EditModeOn
    If TypeOf (fpSpread1.EditingControl) Is
GrapeCity.Win.Spread.InputMan.CellType.GcDateTime Then
        AddHandler CType(fpSpread1.EditingControl,
GrapeCity.Win.Spread.InputMan.CellType.GcDateTime).DropDownOpening, AddressOf
IMCellType_DropDownOpening
    End If
End Sub
```

Using Touch Support with Filtering

You can use touch gestures when filtering.

Use the **ZoomFactor** (**'ZoomFactor Property' in the on-line documentation**) property to increase the size of the drop-down list. The **AllowAutoFilter** (**'AllowAutoFilter Property' in the on-line documentation**) property must be true to allow filtering with touch gestures.



Note: If the user selects a column that contains sorting and filtering indicators, the resize gripper is displayed. The gripper has a higher priority than the filter list or sort operation. Set the **HeaderIndicatorPositionAdjusting** (**'HeaderIndicatorPositionAdjusting Property' in the on-line documentation**) property to specify the distance between the sorting and filtering indicators and the right edge of the column so that the user can sort or filter the column while the gripper is displayed.

Using Code

The following example sets the **AllowAutoFilter** (**'AllowAutoFilter Property' in the on-line documentation**)

and **ZoomFactor** ('**ZoomFactor Property**' in the on-line documentation) properties.

CS

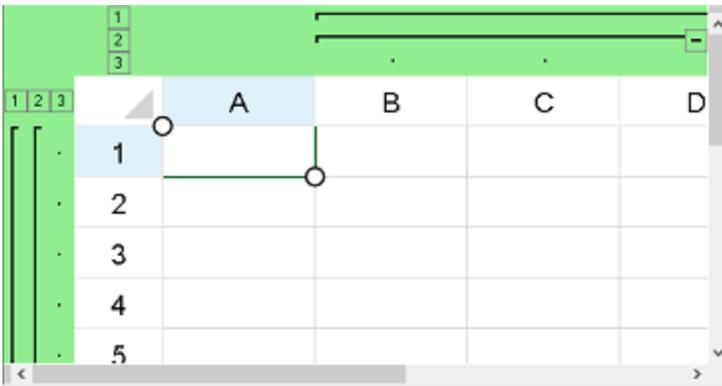
```
fpSpread1.ActiveSheet.Columns[1].AllowAutoFilter = true;
fpSpread1.ActiveSheet.ZoomFactor = 2;
```

VB

```
fpSpread1.ActiveSheet.Columns(1).AllowAutoFilter = True
fpSpread1.ActiveSheet.ZoomFactor = 2
```

Using Touch Support with Range Grouping

You can use touch gestures when expanding or collapsing range groups.



Tap the expand or collapse button to expand or collapse the range group.

Set the **ZoomFactor** ('**ZoomFactor Property**' in the on-line documentation) property to change the size of the control. Touch gestures are easier to use if the control is zoomed.

Using Code

The following code creates a range or outline group and sets the **ZoomFactor** ('**ZoomFactor Property**' in the on-line documentation) property.

CS

```
fpSpread1.ActiveSheet.Rows.Count = 11;
fpSpread1.ActiveSheet.Columns.Count = 6;
fpSpread1.InterfaceRenderer = null;
fpSpread1.ActiveSheet.RangeGroupBackgroundColor = Color.LightGreen;
fpSpread1.ActiveSheet.RangeGroupButtonStyle =
FarPoint.Win.Spread.RangeGroupButtonStyle.Enhanced;
fpSpread1.ActiveSheet.AddRangeGroup(0, 8, true);
fpSpread1.ActiveSheet.AddRangeGroup(0, 5, true);
fpSpread1.ActiveSheet.AddRangeGroup(1, 3, false);
fpSpread1.ActiveSheet.AddRangeGroup(1, 2, false);
fpSpread1.ZoomFactor = 2;
```

VB

```
fpSpread1.ActiveSheet.Rows.Count = 11
fpSpread1.ActiveSheet.Columns.Count = 6
```

```

fpSpread1.InterfaceRenderer = Nothing
fpSpread1.ActiveSheet.RangeGroupBackgroundColor = Color.LightGreen
fpSpread1.ActiveSheet.RangeGroupButtonStyle =
FarPoint.Win.Spread.RangeGroupButtonStyle.Enhanced
fpSpread1.ActiveSheet.AddRangeGroup(0, 8, True)
fpSpread1.ActiveSheet.AddRangeGroup(0, 5, True)
fpSpread1.ActiveSheet.AddRangeGroup(1, 3, False)
fpSpread1.ActiveSheet.AddRangeGroup(1, 2, False)
fpSpread1.ZoomFactor = 2

```

Using Touch Support with Grouping

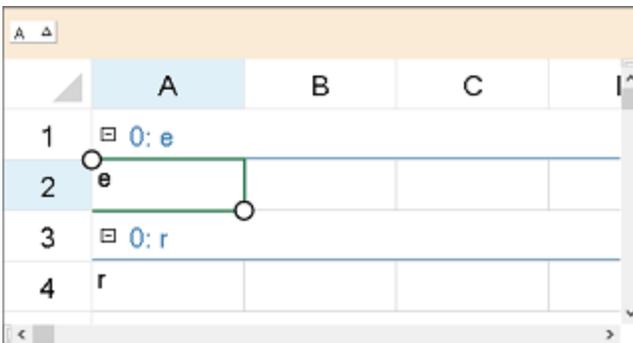
You can use touch gestures when grouping.

Press down on a column header and then slide to the group bar area. Release to create a group.

Tap the group header button area to sort.

You can change the group order. Press down on the group header button and then slide. Release over the target position to change the order.

You can expand or collapse the group by tapping the plus or minus symbol.



Set the **ZoomFactor** ('**ZoomFactor Property**' in the on-line documentation) property to change the size of the control. Touch gestures are easier to use if the control is zoomed.

Set the **AllowColumnMove** ('**AllowColumnMove Property**' in the on-line documentation), **AllowGroup** ('**AllowGroup Property**' in the on-line documentation), and **GroupBarInfo.Visible** ('**Visible Property**' in the on-line documentation) properties to true to allow grouping with touch gestures.

Using Code

The following example sets the **AllowColumnMove** ('**AllowColumnMove Property**' in the on-line documentation), **AllowGroup** ('**AllowGroup Property**' in the on-line documentation), **GroupBarInfo.Visible** ('**Visible Property**' in the on-line documentation), and **ZoomFactor** ('**ZoomFactor Property**' in the on-line documentation) properties.

CS

```

fpSpread1.AllowColumnMove = true;
fpSpread1.ActiveSheet.GroupBarInfo.Visible = true;
fpSpread1.ActiveSheet.AllowGroup = true;
fpSpread1.ActiveSheet.ZoomFactor = 2;

```

VB

```

fpSpread1.AllowColumnMove = True

```

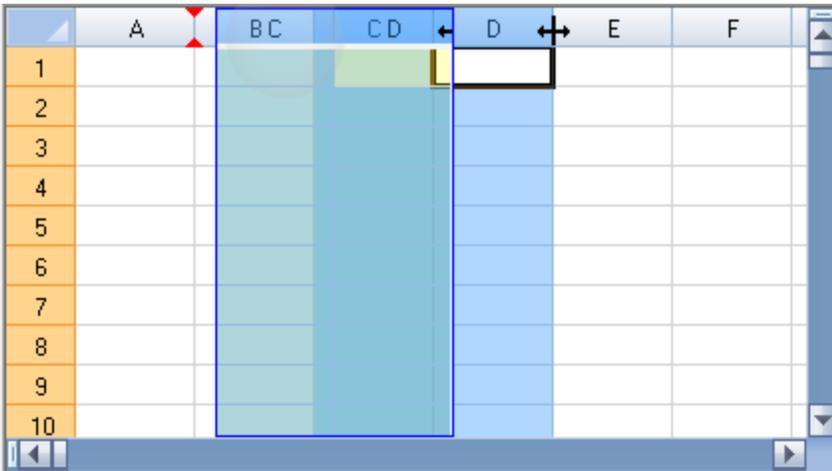
```
fpSpread1.ActiveSheet.GroupBarInfo.Visible = True
fpSpread1.ActiveSheet.AllowGroup = True
fpSpread1.ActiveSheet.ZoomFactor = 2
```

Using Touch Support when Moving Columns or Rows

You can use touch gestures to move columns or rows.

Press the column header or row header to select it, then slide to the target location. Release to move the column or row.

Select a column or row header range and then press and slide to move the range. Release to complete the action.



The **AllowColumnMove** ('AllowColumnMove Property' in the on-line documentation) property must be true to move columns. **AllowColumnMove** ('AllowColumnMove Property' in the on-line documentation) and **AllowColumnMoveMultiple** ('AllowColumnMoveMultiple Property' in the on-line documentation) must be true to move multiple columns. The **AllowRowMove** ('AllowRowMove Property' in the on-line documentation) property must be true to move rows. **AllowRowMove** ('AllowRowMove Property' in the on-line documentation) and **AllowRowMoveMultiple** ('AllowRowMoveMultiple Property' in the on-line documentation) must be true to move multiple rows. The **AllowRowMoveDataAllowAddNew** ('AllowRowMoveDataAllowAddNew Property' in the on-line documentation) property must be true to move a row below the add new row or asterisk row.

Refer to **Using Touch Support with Selections** for more information on how to select a column or row.

Using Code

This example sets the **AllowColumnMove** ('AllowColumnMove Property' in the on-line documentation), **AllowColumnMoveMultiple** ('AllowColumnMoveMultiple Property' in the on-line documentation), **AllowRowMove** ('AllowRowMove Property' in the on-line documentation), and **AllowRowMoveMultiple** ('AllowRowMoveMultiple Property' in the on-line documentation) properties.

CS

```
fpSpread1.AllowColumnMove = true;
fpSpread1.AllowColumnMoveMultiple = true;
fpSpread1.AllowRowMove = true;
fpSpread1.AllowRowMoveMultiple = true;
```

VB

```
fpSpread1.AllowColumnMove = True
```

```
fpSpread1.AllowColumnMoveMultiple = True
fpSpread1.AllowRowMove = True
fpSpread1.AllowRowMoveMultiple = True
```

Using Code

This example sets the **AllowRowMoveDataAllowAddNew** (**'AllowRowMoveDataAllowAddNew Property' in the on-line documentation**) property after binding the control. The **DataAllowAddNew** (**'DataAllowAddNew Property' in the on-line documentation**) property must be true to allow the asterisk row.

CS

```
DataSet ds = new DataSet();
DataTable emp = new DataTable("Employees");
DataTable div = new DataTable("Division");
emp.Columns.Add("LastName");
emp.Columns.Add("FirstName");
emp.Rows.Add(new Object[] { "Jones", "Marianne" });
emp.Rows.Add(new Object[] { "Fieldes", "Anna" });
div.Columns.Add("Section");
div.Columns.Add("Specialty");
div.Rows.Add(new Object[] { "Finance", "Taxes" });
div.Rows.Add(new Object[] { "Mergers", "Legal" });
ds.Tables.AddRange(new DataTable[] { emp, div });
fpSpread1.DataSource = ds;
fpSpread1.DataMember = "Division";
fpSpread1.AllowRowMove = true;
fpSpread1.AllowRowMoveMultiple = true;
fpSpread1.ActiveSheet.DataAllowAddNew = true;
fpSpread1.AllowRowMoveDataAllowAddNew = true;
```

VB

```
Dim ds As New DataSet()
Dim emp As New DataTable("Employees")
Dim div As New DataTable("Division")
emp.Columns.Add("LastName")
emp.Columns.Add("FirstName")
emp.Rows.Add(New Object() {"Jones", "Marianne"})
emp.Rows.Add(New Object() {"Fieldes", "Anna"})
div.Columns.Add("Section")
div.Columns.Add("Specialty")
div.Rows.Add(New Object() {"Finance", "Taxes"})
div.Rows.Add(New Object() {"Mergers", "Legal"})
ds.Tables.AddRange(New DataTable() {emp, div})
fpSpread1.DataSource = ds
fpSpread1.DataMember = "Division"
fpSpread1.AllowRowMove = True
fpSpread1.AllowRowMoveMultiple = True
fpSpread1.ActiveSheet.DataAllowAddNew = True
fpSpread1.AllowRowMoveDataAllowAddNew = True
```

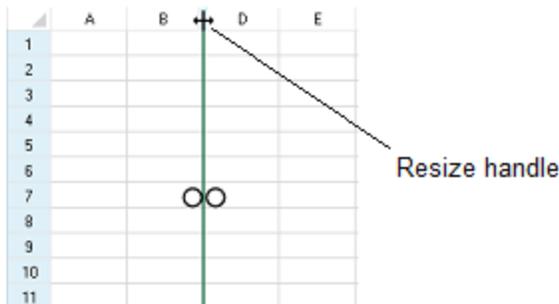
Using Touch Support when Resizing Columns or Rows

You can resize columns or rows using touch gestures.

Select a column or row (tap to select), press the column or row resize handle and slide to change the width or height, and then release.

FpSpread provides a **ResizeZeroIndicator** ('**ResizeZeroIndicator Property**' in the on-line documentation) property that specifies the policy when a column or row is resized to zero. If the property is set to Default when using touch gestures, a column with zero width cannot be resized (or a row with a height of zero). If the property value is Enhanced, then a zero width column (or zero height row), can be resized. Select the column or row, and press and hold the resize handle to resize the visible column or row. Tap the center of the two short lines to select the column with a width of zero (or row with a height of zero), then press and hold the resize handle to resize the column (or row).

The following image displays a selected, zero width column.



The column or row **Resizable** ('**Resizable Property**' in the on-line documentation) property must be true to resize a column or row.



FpSpread provides a **ResizeZeroIndicator** ('**ResizeZeroIndicator Property**' in the on-line documentation) property that specifies the policy when a column or row is resized to zero. When using the mouse, the default value displays the same mouse cursor for resizing and resizing-out. The Enhanced value displays a resize cursor to the left of a column header border and a resize-out cursor to the right of a column header border.

Using Code

The following example allows the user to resize zero width columns or zero height rows by setting the **ResizeZeroIndicator** ('**ResizeZeroIndicator Property**' in the on-line documentation) property to Enhanced.

CS

```
fpSpread1.ResizeZeroIndicator = FarPoint.Win.Spread.ResizeZeroIndicator.Enhanced;
fpSpread1.ActiveSheet.Columns[0, 5].Resizable = true;
fpSpread1.ActiveSheet.Rows[0, 10].Resizable = true;
fpSpread1.ActiveSheet.Columns[2].Width = 0;
```

VB

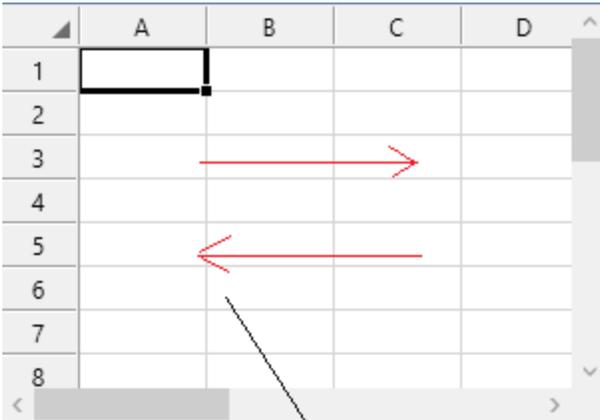
```
fpSpread1.ResizeZeroIndicator = FarPoint.Win.Spread.ResizeZeroIndicator.Enhanced
fpSpread1.ActiveSheet.Columns(0, 5).Resizable = True
fpSpread1.ActiveSheet.Rows(0, 10).Resizable = True
fpSpread1.ActiveSheet.Columns(2).Width = 0
```

Using Touch Support with Scrolling

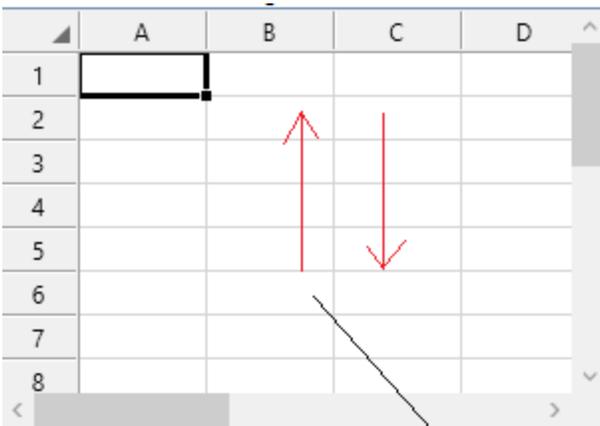
You can use touch gestures when scrolling in the control.

You can tap the scroll bar or press and slide the scroll bar to scroll. You can also use panning gestures in the cell area of the control (vertical, horizontal, diagonal, or oblique). Panning in the diagonal direction scrolls horizontally and vertically. Specify the type of panning mode with the **PanningMode** ('**PanningMode Property**' in the on-line

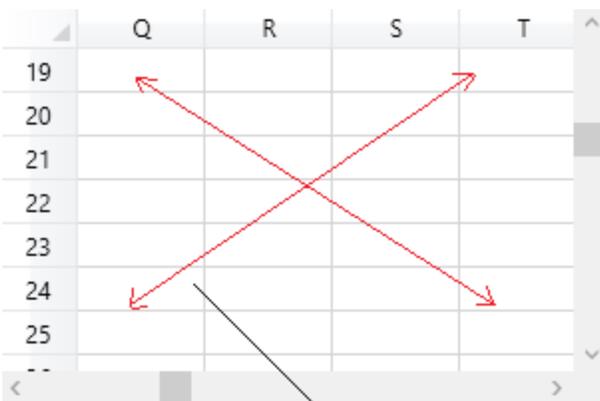
documentation) property.



Horizontal panning



Vertical panning



Diagonal panning

You can specify feedback when scrolling with the **BoundaryFeedbackMode** ('**BoundaryFeedbackMode Property**' in the on-line documentation) property (standard or split). For more information on standard, see

[BeginPanningFeedback function \(uxtheme.h\)](#). The Split option separates frozen and scrollable areas or headers and scrollable areas.

Using Code

This example sets the **PanningMode** ('**PanningMode Property**' in the on-line documentation) property to allow horizontal and vertical panning and specifies the type of feedback.

CS

```
fpSpread1.PanningMode = FarPoint.Win.Spread.SpreadPanningMode.Both;
fpSpread1.BoundaryFeedbackMode = FarPoint.Win.Spread.BoundaryFeedbackMode.Split;
```

VB

```
fpSpread1.PanningMode = FarPoint.Win.Spread.SpreadPanningMode.Both
fpSpread1.BoundaryFeedbackMode = FarPoint.Win.Spread.BoundaryFeedbackMode.Split
```

Using Touch Support with Selections

You can select columns, rows, cell ranges, and the entire control using touch gestures.

Tap a cell to select the cell and display the selection gripper. Press the cell selection gripper and slide. Release to select a cell range.

Tap a column header (or row header) to select a column (or row). You can then press the selection gripper and slide to select a column range (or row range). Release to complete the selection. You can also select a column or row range by tapping a header and then sliding in the header area. Release to complete the selection. This action requires that the **AllowColumnMove** ('**AllowColumnMove Property**' in the on-line documentation) property (or **AllowRowMove** ('**AllowRowMove Property**' in the on-line documentation) property) be set to false.

You can select the entire control by tapping the corner header.

You can change the size of the cell range selection by pressing the selection gripper and sliding in any direction. Release to complete the action.

You can select multiple ranges. Select a range, then tap a cell in a different location to start the next selection. Use the gripper to select the second range. Set the **TapToAddSelection** ('**TapToAddSelection Property**' in the on-line documentation) property to true. The **SelectionPolicy** ('**SelectionPolicy Property**' in the on-line documentation) property must be set to MultiRange and the **UseOptimizedSelectionForTouch** ('**UseOptimizedSelectionForTouch Property**' in the on-line documentation) property must be true. Set the **OperationMode** ('**OperationMode Property**' in the on-line documentation) property to Normal or ReadOnly. The following image displays multiple selections and grippers around one selected cell range.

The image shows a spreadsheet with columns A through E and rows 1 through 8. A blue shaded area covers cells A1 through B2. A green rectangular selection gripper is positioned over cells C4 through D6. Two small circular grippers are visible: one at the top-left corner of the green selection (cell C4) and one at the bottom-right corner (cell D6).

	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					
8					

You can tap to select or unselect a row when the **OperationMode** ('**OperationMode Property**' in the on-line documentation) property is set to MultiSelect. The gripper is not displayed when using MultiSelect. If the **OperationMode** property is set to ExtendedSelect, you can tap to select a row, but not to unselect the row. The gripper is displayed with ExtendedSelect and can be used to select a range of rows.

Set the **UseOptimizedSelectionForTouch** to true to display a selection gripper for selecting a cell range. The gripper is displayed when touching the cell, column, or row. The gripper is not displayed when using the mouse or keyboard.

The border is displayed around the selected cell range when using touch operations.

The selection grippers are displayed on the outside edge of the range (top-left and bottom-right edges, by default). You can customize the gripper appearance using the **TouchSelectionGripperThickness** (**'TouchSelectionGripperThickness Property' in the on-line documentation**), **TouchSelectionGripperLineColor** (**'TouchSelectionGripperLineColor Property' in the on-line documentation**), and **TouchSelectionGripperBackColor** (**'TouchSelectionGripperBackColor Property' in the on-line documentation**) properties. You can change the location of the grippers with the **RightToLeft** property.

Using Code

The following example allows multiple selections using touch support and sets the **TouchSelectionGripperBackColor** (**'TouchSelectionGripperBackColor Property' in the on-line documentation**), **TouchSelectionGripperLineColor** (**'TouchSelectionGripperLineColor Property' in the on-line documentation**), and **TouchSelectionGripperThickness** (**'TouchSelectionGripperThickness Property' in the on-line documentation**) properties.

CS

```
fpSpread1.TouchSelectionGripperBackColor = Color.Aquamarine;
fpSpread1.TouchSelectionGripperLineColor = Color.DarkMagenta;
fpSpread1.TouchSelectionGripperThickness = 2;
fpSpread1.TapToAddSelection = true;
fpSpread1.ActiveSheet.SelectionPolicy =
FarPoint.Win.Spread.Model.SelectionPolicy.MultiRange;
fpSpread1.UseOptimizedSelectionForTouch = true;
```

VB

```
fpSpread1.TouchSelectionGripperBackColor = Color.Aquamarine
fpSpread1.TouchSelectionGripperLineColor = Color.DarkMagenta
fpSpread1.TouchSelectionGripperThickness = 2
fpSpread1.TapToAddSelection = True
fpSpread1.ActiveSheet.SelectionPolicy =
FarPoint.Win.Spread.Model.SelectionPolicy.MultiRange
fpSpread1.UseOptimizedSelectionForTouch = True
```

Using Touch Support with Shapes

You can use touch gestures with shapes.

Use the following touch gestures with shapes:

Touch Gesture	Mouse Action	Action
Tap	Click	Selects a cell note, shape, or chart.
Double-tap	Double-Click	Edits a cell note, shape, or chart if editing is supported.
Press edge then slide	Press left button on edge then move	Resizes a cell note, shape, or chart if CanSize ('CanSize Property' in the on-line documentation) is set to true.
Press shape then slide	Press left button on shape then move	Moves a cell note, shape, or chart if CanMove ('CanMove Property' in the on-line documentation) is set to true.
Press rotated handle and slide	Press left button on rotated handle and move	Rotates a shape or chart if CanRotate ('CanRotate Property' in the on-line documentation) is set to true.

Using Touch Support when Sorting

You can use touch gestures when sorting.

Tap the sort indicator to sort. The **AllowAutoSort ('AllowAutoFilter Property' in the on-line documentation)** property must be true to use touch gestures. The following image displays sort indicators.

	A	B	C
1	Z		
2	W		
3	A		
4			
5			

Set the **ZoomFactor ('ZoomFactor Property' in the on-line documentation)** property to change the size of the control. Some touch gestures are easier to use if the control is zoomed.

 **Note:** If the user selects a column that contains sorting and filtering indicators, the resize gripper is displayed. The gripper has a higher priority than the filter list or sort operation. Set the **HeaderIndicatorPositionAdjusting ('HeaderIndicatorPositionAdjusting Property' in the on-line documentation)** property to specify the distance between the sorting and filtering indicators and the right edge of the column so that the user can sort or filter the column while the gripper is displayed.

Using Code

The following example allows sorting and zooms the control.

CS

```
fpSpread1.ActiveSheet.Columns[0, 3].AllowAutoSort = true;
fpSpread1.ActiveSheet.ZoomFactor = 2;
```

VB

```
fpSpread1.ActiveSheet.Columns(0, 3).AllowAutoSort = True
fpSpread1.ActiveSheet.ZoomFactor = 2
```

Using Touch Support with Viewports

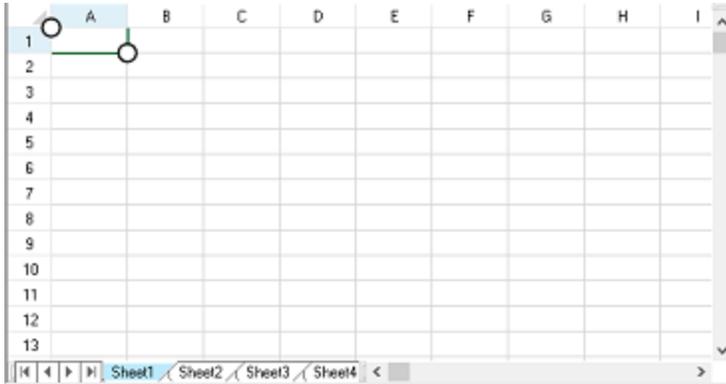
You can use touch gestures with viewports.

Press the split box and slide to display the split line or bar. Press on the line and slide to change the size of the viewport.

You can double-tab a split line to remove the viewport.

Using Touch Support with the Tab Strip

You can use touch gestures with the tab strip. You can change tabs, select a sheet, edit a sheet, and scroll.



Tap the tab strip buttons to navigate the tab strip. Tap a sheet name in the tab strip to select the sheet. Double-tap a sheet name in the tab strip to edit the sheet name. You can use panning to scroll through the sheets in the tab strip if there are more sheets than can be displayed in the tab strip area. The **Editable ('Editable Property' in the on-line documentation)** property must be true for the tab strip in order to edit the sheet names.

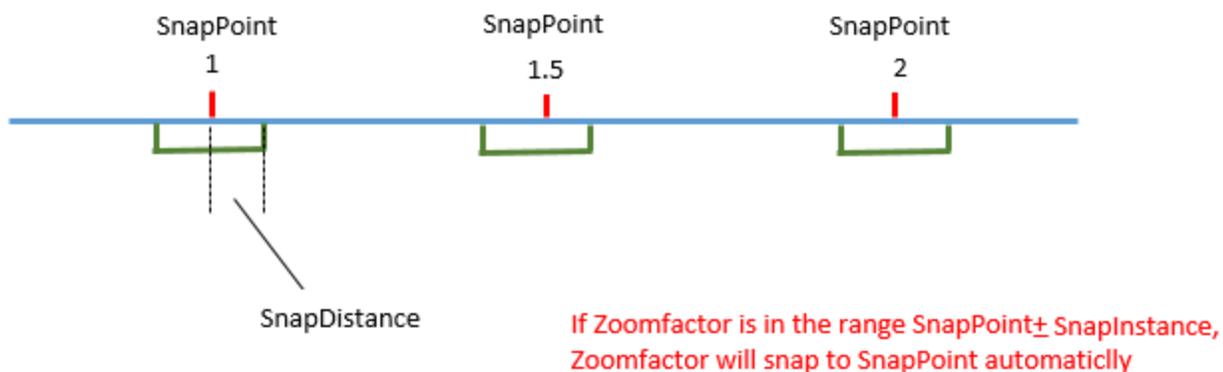
Set the **ZoomFactor ('ZoomFactor Property' in the on-line documentation)** property to change the size of the tab strip buttons. Some touch gestures are easier to use if the control is zoomed.

Using Touch Support with Zooming

You can use the pinch operation when zooming.

Set the **AllowUserToTouchZoom ('AllowUserToTouchZoom Property' in the on-line documentation)** property to allow zooming using the pinch operation.

You can use the **TouchZoomSnapPoints ('TouchZoomSnapPoints Property' in the on-line documentation)** property to configure snap points. When zooming, if the final zoom factor is close to (less than the **TouchZoomSnapDistance ('TouchZoomSnapDistance Property' in the on-line documentation)** property value) a snap point, the final zoom factor changes to the snap point. For example, add 1 to the snap point. When the user changes the zoom factor to 103%, the zoom factor changes to 100%.



FpSpread uses standard pinch and stretch gestures when zooming. For more information, see <https://docs.microsoft.com/en-us/>.

Using Code

The following example allows zooming with the pinch operation.

CS

```
fpSpread1.AllowUserToTouchZoom = true;
```

```
fpSpread1.MinZoomFactor = .1F;  
fpSpread1.TouchZoomSnapDistance = 1;  
fpSpread1.TouchZoomSnapPoints.Add(1f);  
fpSpread1.TouchZoomSnapPoints.Add(2f);
```

VB

```
fpSpread1.AllowUserToTouchZoom = True  
fpSpread1.MinZoomFactor = .1F  
fpSpread1.TouchZoomSnapDistance = 1  
fpSpread1.TouchZoomSnapPoints.Add(1f)  
fpSpread1.TouchZoomSnapPoints.Add(2f)
```

1 Index

Accessing Data from Header or Footer, 688-690
Activating the Product, 23-30
Add Sparklines using Formulas, 713-714
Add Sparklines Using Methods, 704-706
Adding a Comment to a Cell, 310-317
Adding a Component to a Visual Studio 2017 Project, 67-68
Adding a Component to a Visual Studio 2019 Project, 65-67
Adding a Context Menu to a Component, 552-553
Adding a Custom Sort Dialog, 616-617
Adding a Gradient to Header Cells, 211-212
Adding a Note to a Cell, 284-288
Adding a Page Break, 847-848
Adding a Row or Column, 173-174
Adding a Row to a Bound Sheet, 433-434
Adding a Sheet, 128-129
Adding a Status Bar, 95-96
Adding a Table, 554-555
Adding a Table Formula, 567-569
Adding a Tag to a Cell, 288-289
Adding a Tag to a Row or Column, 196
Adding a Tag to a Sheet, 162
Adding a Title and Subtitle to a Sheet, 151-153
Adding a Watermark to a Printed Page, 848-849
Adding an Unbound Column to a Bound Sheet, 426
Adding an Unbound Row to a Bound Sheet, 434-436
Adding ChartSheet, 129-132
Adding Formulas to Calculate Balances, 43-44
Adding Hyperlink in a Cell, 231-235
Adding Image in a Cell, 317-323
Adding NuGet Package in Spread Windows Forms, 55-60
Adding Spread to the Checkbook Project, 38-39
Adding Support for High DPI Settings, 74-76
Adding to Bound Data, 433
Advanced chart settings, 958
Aligning Cell Contents, 276-279
Allowing a Combo Box Cell to Handle a Double Click, 386-388
Allowing Cell Data to Overflow, 274-276
Allowing Cells to Merge Automatically, 282-284
Allowing the Display of Buttons in a Cell, 407-408
Allowing the User to Automatically Sort Rows, 587-589

- Allowing the User to Change the Chart, 919-920**
- Allowing the User to Draw with a Tablet PC, 966-967**
- Allowing the User to Enter Formulas, 655-657**
- Allowing the User to Filter Rows, 595-597**
- Allowing the User to Group Rows, 623-624**
- Allowing the User to Perform a Standard Search, 637-638**
- Allowing the User to Perform an Advanced Search, 638**
- Allowing the User to Resize Rows or Columns, 180-182**
- Allowing the User to Zoom the Display of the Component, 88-89**
- Allowing User Functionality, 87-88**
- Applying a Skin to a Sheet, 477-479**
- Applying a Skin to the Component, 473-474**
- Applying Theme to Customize the Appearance, 506-507**
- Area Charts, 868-869**
- Area Sparkline, 714-717**
- Auto Expand Row Headers, 212-213**
- Auto Format Formulas, 690-692**
- Available Language Packages for WinForms, 685-686**
- Axis, 901-902**
- Bar Charts, 865-868**
- barcode**
 - cell type, 367-372
- Binding a Cell Range in Spread as a Data Source to an External Control, 422-424**
- Binding a Cell Range in Spread to an External Data Source, 420-422**
- Binding a Combo Box to a DataReader, 424-425**
- Binding a Table, 563-567**
- Binding Spread to an External Data Set, 419-420**
- Binding to Data, 419**
- Binding with cell range, 955-958**
- Box Whisker Charts, 869-872**
- BoxPlot Sparkline, 717-719**
- Bullet Sparkline, 720-721**
- Cascade Sparkline, 721-723**
- Cell Spans in the Sheet Corner, 225-226**
- Cell Types, 50-52 , 325**
 - barcode, 367-372
 - color picker, 379-383
- Cells, 229-230**
- Changing an Input Map for a Child View, 776-778**
- Changing the Default Keyboard Map, 769-770**
- Chart Control, 857-858**
- Chart Line Style, 902-904**
- Chart Object Model, 861-862**

- Chart Types and Views, 862-864**
- Chart User Interface Elements, 859-861**
- Clicking Actions, 785-788**
- color picker**
 - cell type, 379-383
- Color Scale Rules, 487-488**
- Coloring a Cell, 456-459**
- colors**
 - color picker, 379-383
- Column, Line, and Winloss Sparkline, 714**
- Combining different types of plots, 951-952**
- Connecting to Data, 952-953**
- Copying and Inserting a Sheet, 138-141**
- Copying Data on a Sheet, 247**
- Creating a Complex Border with Multiple Lines, 470-471**
- Creating a Custom Cell Type, 414-418**
- Creating a Custom Filter, 608-613**
- Creating a Custom Group, 627-631**
- Creating a Custom Sheet Model, 582-584**
- Creating a Custom Skin for a Component, 474-477**
- Creating a Custom Skin for a Sheet, 479-481**
- Creating a Header with Multiple Rows or Columns, 213-215**
- Creating a Hierarchical Display Manually, 441-442**
- Creating a Pie Plot, 933-937**
- Creating a Polar Plot, 937-940**
- Creating a Radar Plot, 940-943**
- Creating a Range of Cells, 235**
- Creating a Runtime License, 30**
- Creating a Span in a Header, 215-218**
- Creating a Span of Cells, 280-282**
- Creating a Sunburst Chart, 943-946**
- Creating a Treemap Chart, 946-950**
- Creating a Y Plot, 924-927**
- Creating Alternating Rows, 189-191**
- Creating an XY Plot, 927-929**
- Creating an XYZ Plot, 929-933**
- Creating and Applying a Style for Cells, 483-486**
- Creating and Customizing Cell Borders, 466-470**
- Creating and Using a Custom Function, 658-660**
- Creating and Using a Custom Language Package, 686-688**
- Creating and Using a Custom Name, 657-658**
- Creating and Using a Visual Function, 660-662**

- Creating and Using External Variable, 662-663**
- Creating Camera Shapes, 964-966**
- Creating Charts, 913-914**
- Creating Conditional Formatting with Rules, 486-487**
- Creating Custom Hierarchy Icons, 442**
- Creating Data Type for Custom Objects, 267-272**
- Creating Enhanced Camera Shape, 980-999**
- Creating Plot Types, 924**
- culture, 33**
- Customizable Cell in the Sheet Corner, 224-225**
- Customizing Automatic Completion (Type Ahead), 412-413**
- Customizing Cell Types for Bound Sheets, 429-431**
- Customizing Clipboard Operation Options, 163-170**
- Customizing Column and Field Binding, 426-429**
- Customizing Column Headers for Bound Sheets, 431-433**
- Customizing Data Binding, 425**
- Customizing Drawing, 962**
- Customizing Enhanced Filtering, 613-616**
- Customizing Header Label Text, 201-202**
- Customizing Interaction Based on Events, 551-552**
- Customizing Interaction in Cells, 508**
- Customizing Markers and Points, 708-710**
- Customizing Painting of Parts of the Component, 504-505**
- Customizing Row or Column Interaction, 586**
- Customizing Scroll Bar Tips, 94-95**
- Customizing Simple Filtering, 600**
- Customizing Split Boxes, 100-102**
- Customizing the Appearance of a Cell, 461-462**
- Customizing the Appearance of an Outline (Range Group), 634-636**
- Customizing the Appearance of Headers, 205-206**
- Customizing the Appearance of the Printing, 826**
- Customizing the Default Header Labels, 202-204**
- Customizing the Dimensions of the Component, 453-454**
- Customizing the Display of the Pointer, 496-498**
- Customizing the Filter Bar, 617-619**
- Customizing the Filter List, 602**
- Customizing the Focus Indicator for a Cell, 548-550**
- Customizing the Group Bar, 627**
- Customizing the Header Grid Lines, 208-211**
- Customizing the Individual Sheet Appearance, 454-455**
- Customizing the Input Maps, 774-776**
- Customizing the Number of Rows or Columns, 171-172**
- Customizing the Outline of the Component, 462-463**

- Customizing the Overall Component Appearance, 471-472**
- Customizing the Pop-Up Calculator Control, 411-412**
- Customizing the Pop-Up Date-Time Control, 336-338**
- Customizing the Position in the Display, 93-94**
- Customizing the Print Job Settings, 831-834**
- Customizing the Print Preview Dialog, 845-847**
- Customizing the Printed Page Header or Footer, 835-845**
- Customizing the Printed Page Layout, 834-835**
- Customizing the Renderers, 500-504**
- Customizing the Scroll Bars, 89-93**
- Customizing the Selection Appearance, 515-517**
- Customizing the Sheet Appearance, 453**
- Customizing the Sheet Corner Appearance, 218**
- Customizing the Sheet Name Tabs, 122-126**
- Customizing the Style of Header Cells, 206-208**
- Customizing the User Error Messages, 545**
- Customizing the User Interface Images, 498-499**
- Customizing Undo and Redo Actions, 550-551**
- Customizing User Searching of Data, 636-637**
- Customizing User Selection and Deselection of Data, 511-512**
- Customizing Viewports, 97-100**
- Data Bar Rule, 488-490**
- Data Binding, 419**
 - tutorial, 442-452
- Data Plot Types, 898**
- Deactivating the Default Keyboard Map, 768-769**
- Default Keyboard Maps, 761-762**
- Default Keyboard Navigation, 754-761**
- Default Map for Excel Compatibility, 782-784**
- Default Map for ExtendedSelect and WhenAncestorOfFocused, 767-768**
- Default Map for ExtendedSelect and WhenFocused, 767**
- Default Map for MultiSelect and WhenAncestorOfFocused, 767**
- Default Map for MultiSelect and WhenFocused, 766-767**
- Default Map for Normal and WhenAncestorOfFocused, 762-764**
- Default Map for Normal and WhenFocused, 762**
- Default Map for ReadOnly and WhenAncestorOfFocused, 764 , 764-765**
- Default Map for ReadOnly and WhenFocused, 764**
- Default Map for RowMode and WhenAncestorOfFocused, 765-766**
- Default Map for SingleSelect and WhenAncestorOfFocused, 766**
- Default Map for SingleSelect and WhenFocused, 766**
- Defining the Contents of the Filter Item List, 602-605**
- Defining the Order of the Items in the Filter Item List, 605-606**

Determining Which Header Row Displays the Indicators, 621

Developer's Guide, 0

dialogs

color dialog, 379-383

Display format, 960-961

Displaying a Footer for Columns or Groups, 154-162

Displaying a Print Dialog for the User, 853-854

Displaying an Abort Message for the User, 854

Displaying Cell Data, 272-273

Displaying Dialogs for Users, 853

Displaying Error Icons in Cells or Rows, 545-546

Displaying Formula in Cells, 643-644

Displaying Grid Lines on a Sheet, 463-466

Displaying Spin Buttons, 408-409

Displaying Text Tips in a Cell, 289-290

Doughnut Charts, 889-890

Drawing (Rendering) Style, 227-228

drawings

ink notation, 966-967

Elevation and Rotation, 904-905

End-User License Agreement, 30

Entering Data Actions, 789

Events from User Actions, 785

Excel 互換, 781-782

Excel 互換モード, 782-784

Factors of Keyboard Map Usage, 752-754

File Operations, 792

Fill Effects, 958-960

Filling Cells with Drag and Drop, 522-523

Filling Cells with Drag and Fill, 523-527

Filling Cells with Drag and Move, 527

Finding a Value Using GoalSeek, 655

Finding More Details on the Sheet Models, 573

Finding the Documentation, 35-37

Form Controls, 146-151

Formatted versus Unformatted Data, 49-50

Formatting a Cell Value, 323-324

Formulas in Cells, 640

Funnel Charts, 872-874

Gauge KPI Sparkline, 723-724

General Style of the Sheet Corner, 218-221

Getting More Practice, 35

Getting Started, 22

- Handling Data Using Cell Properties, 245-246**
- Handling Data Using Sheet Methods, 243-245**
- Handling Installation, 22**
- Handling Redistribution, 30-31**
- Handling Right-to-Left Layouts, 504**
- Hbar and Vbar Sparkline, 724-726**
- Header Count Synchronization in the Sheet Corner, 226-227**
- Headers, 197**
- Hiding the Selection When Focus is Lost, 519-520**
- Highlighting Rules, 491-493**
- Histogram Charts, 874-876**
- Histogram Sparkline, 726-730**
- Icon Set Rule, 490-491**
- Image Sparkline, 730-733**
- Implementing a Serializer Class, 806-811**
- Improving Performance by Suspending the Layout, 84-86**
- ink notation, 966-967**
- inking, 966-967**
- Input Data in Rows or Columns, 193-194**
- Inserting Cells, 298-307**
- Installing the Product, 22**
- Interactivity Actions, 790**
- Interoperability of Grouping with Other Features, 631**
- Interoperability of Outlines with Other Features, 636**
- Key Features, 52-54**
- Keyboard Interaction, 751**
- Labels, 908-912**
- language, 33**
- Legends, 912-913**
- Lighting, Shapes, and Borders, 905-908**
- Limiting Values for a Numeric Cell, 409-411**
- Line Charts, 876-878**
- Loading a Skin, 482-483**
- Locating the Pointer Using HitTest, 546-547**
- Locking a Cell, 509-511**
- Managing Cell Range Filtering, 621-623**
- Managing Cell Range Sorting, 593-594**
- Managing Data on a Sheet, 242**
- Managing External Reference, 692-695**
- Managing Filtering of Rows of User Data, 594**
- Managing Grouping of Rows of User Data, 623**
- Managing Outlines (Range Groups) of Rows and Columns, 631-632**

Managing Sorting of Rows of User Data, 586-587
Market Data (High-Low) Charts, 878-879
Migrate .NET Framework Project to .NET 6 Platform, 60-65
modes
 Inking, 966-967
Month and Year Sparkline, 734-737
Moving a Sheet, 135-136
Moving Data on a Sheet, 246-247
Moving Rows or Columns, 187-189
Navigating Sheet Tabs, 126-128
Nesting Functions in a Formula, 653-654
notation
 ink, 966-967
Object Parentage, 78-79
Opening a Custom Text File, 803-804
Opening a Spread COM File, 804-805
Opening a Spread XML File, 801-802
Opening an Excel File, 802-803
Opening Existing Files, 801
Optimizing the Printing, 850
Optimizing the Printing Using Rules, 850-853
Optimizing the Printing Using Size, 853
Pareto Charts, 879
Pareto Sparkline, 737-739
Parsing Formulas in Custom XML Deserialization, 811-812
Pie Charts, 890-891
Pie Plot Types, 889
Pie Sparkline, 739-741
Placing a Formula in Cells, 640-643
Placing and Retrieving Data, 242-243
Placing Child Controls on a Sheet, 153-154
Plot area, 898-899
Plot Types, 864
Point Charts, 879-881
Polar Area Charts, 893-894
Polar Line Charts, 892-893
Polar Plot Types, 891
Polar Point Charts, 891-892
Polar Stripe Charts, 894
Precedents and Dependents, 695-703
Preventing a Cell from Receiving Focus, 547-548
Print Actions, 790-791
Printing, 814

- Printing a Child View of a Hierarchical Display, 817-818**
- Printing a Range of Cells on a Sheet, 820-822**
- Printing a Sheet with Cell Notes, 823-825**
- Printing a Sheet with Shapes, 825-826**
- Printing an Area of the Sheet, 823**
- Printing an Entire Sheet, 814-816**
- Printing in Duplex Mode, 826**
- Printing Particular Pages, 818-820**
- Printing the Portion of the Sheet with Data, 820**
- Printing to PDF, 816-817**
- Product Overview, 45-46**
- Product Requirements, 31-33**
- Protecting a Worksheet, 141-146**
- Providing a Preview of the Printing, 854-856**
- Radar Area Charts, 897**
- Radar Line Charts, 896-897**
- Radar Plot Types, 894-895**
- Radar Point Charts, 895-896**
- Radar Stripe Charts, 897-898**
- Recalculating and Updating Formulas Automatically, 654-655**
- Remove Duplicates from Range, 251-258**
- Removing a Row or Column, 174-175**
- Removing a Sheet, 132-133**
- Removing Data from a Sheet, 250-251**
- Repeatedly Filling a Range of Cells with Copied Cells, 247-249**
- Repeating Rows or Columns on Printed Pages, 847**
- Resetting Parts of the Interface, 79-82**
- Resizing a Cell to Fit the Data, 279-280**
- Resizing a Table, 558-559**
- Resizing the Data to Fit the Cell, 273-274**
- Resizing the Row or Column to Fit the Data, 178-180**
- Returning Information for a Clicked Cell, 547**
- Ribbon Control, 118-119**
- Rich Text Editing, 82-84**
- Rows and Columns, 171**
- Rows or Columns That Have Data, 194-196**
- Save/Load Chart Control, 923-924**
- Saving a Skin, 482**
- Saving and Loading a Skin, 481-482**
- Saving and Loading Map Files, 778-779**
- Saving Data to a File, 792**
- Saving or Loading a Chart, 920-921**

- Saving Spreadsheet Data to Simple XML, 800-801**
- Saving to a Spread XML File, 792-793**
- Saving to a Text File, 795-796**
- Saving to an Excel File, 793-795**
- Saving to an HTML Table, 799-800**
- Saving to an Image File, 796-797**
- Scatter Sparkline, 741-742**
- Searching for Data with Code, 638-639**
- Selecting Actions, 788-789**
- Selecting Multiple Sheets, 136-138**
- Series, 899-900**
- Setting a Background Image for a Sheet, 459-460**
- Setting a Background Image to a Cell, 460-461**
- Setting a Barcode Cell, 367-372**
- Setting a Button Cell, 372-376**
- Setting a Check Box Cell, 376-379**
- Setting a Color Picker Cell, 379-383**
- Setting a Combo Box Cell, 383-386**
- Setting a Currency Cell, 357-358**
- Setting a Date-Time Cell, 334-336**
- Setting a GcCharMask Cell, 338-341**
- Setting a GcDateTime Cell, 341-342**
- Setting a GcMask Cell, 342-345**
- Setting a GcNumber Cell, 345-347**
- Setting a GcTextBox Cell, 347-349**
- Setting a GcTimeSpan Cell, 349-351**
- Setting a General Cell, 332-333**
- Setting a Hyperlink Cell, 388-391**
- Setting a List Box Cell, 394-396**
- Setting a Mask Cell, 358-360**
- Setting a Multiple Option Cell, 399-401**
- Setting a Multiple-Column Combo Box Cell, 396-399**
- Setting a Number Cell, 351-357**
- Setting a Percent Cell, 360-361**
- Setting a Progress Indicator Cell, 401-404**
- Setting a Regular Expression Cell, 361-362**
- Setting a Rich Text Cell, 362-366**
- Setting a Slider Cell, 404-407**
- Setting a Text Cell, 333-334**
- Setting an Image Cell, 391-394**
- setting cell type**
 - barcode, 367-372
 - color picker, 379-383

- Setting Fixed (Frozen) Rows or Columns, 183-187**
- Setting Rich Text in a Cell, 307-310**
- Setting Table Styles, 561-563**
- Setting the Appearance of Filter Indicators, 619**
- Setting the Appearance of Filtered Rows, 597-600**
- Setting the Appearance of Grouped Rows, 625-627**
- Setting the Appearance of Sort Indicators, 591-593**
- Setting the Appearance of the Display of the Filter Item List, 606-608**
- Setting the Background Colors for a Sheet, 455-456**
- Setting the Cell Types of the Register, 41-43**
- Setting the Component to the Original Appearance, 472-473**
- Setting the Header Text, 200-201**
- Setting the Height or Width of Header, 199-200**
- Setting the Properties, 68-69**
- Setting the Row Height or Column Width, 176-178**
- Setting up Conditional Formatting of a Cell, 494-496**
- Setting up Preview Rows, 191-193**
- Setting up the Formula Provider, 682-684**
- Setting up the Name Box, 684-685**
- Setting Up the Rows and Columns of the Register, 39-41**
- Shape Actions, 790**
- Sheet-Level Actions, 789-790**
- Sheets, 120**
- Shortcut Objects, 46-49**
- Showing or Hiding a Row or Column, 175-176**
- Showing or Hiding a Sheet, 133-135**
- Showing or Hiding Filter Indicators, 620-621**
- Showing or Hiding Headers, 198-199**
- Sorting a Table, 559-561**
- Sorting Rows, Columns, or Ranges, 590-591**
- Sparklines, 704**
- Specifying a Cell Reference in a Formula, 644-646**
- Specifying a Sheet Reference in a Formula, 646-647**
- Specifying an External Reference in a Formula, 647**
- Specifying Horizontal and Vertical Axes, 706-708**
- Specifying What the User Can Select, 512-515**
- Specifying What to Print, 814**
- Spread Sparkline, 742-745**
- Spreadsheet Objects, 77**
- Stacked Sparkline, 745-747**
- Starting the Spread Wizard, 33-34**
- Storing Excel Summary and View, 798-799**

- Stripe Charts, 881**
- Swapping Data on a Sheet, 249-250**
- Table Display in the Sheet Corner, 222-224**
- Tables, 554**
- tablet PC inking, 966-967**
- Text Display in the Sheet Corner, 221-222**
- Text Rendering with GDI, 505-506**
- Text to Columns, 258-267**
- Top, Bottom, or Average Rules, 493-494**
- Touch Support with the Component, 1003**
- Tutorial: Binding to a Corporate Database (Visual Studio 2013 or later), 442-452**
- Tutorial: Creating a Checkbook Register, 38**
- tutorials**
 - data binding, 442-452
- Underlying Keystroke Processing, 751-752**
- Underlying Models, 49**
- Understanding Additional Features of Cell Types, 407**
- Understanding Cell Type Basics, 325-326**
- Understanding Charts, 858-859**
- Understanding Edit Mode in a Cell, 508-509**
- Understanding Headers, 197-198**
- Understanding How Cell Type Affects Model Data, 330-331**
- Understanding How Cell Types Display and Format Data, 327-330**
- Understanding How Cell Types Work, 326-327**
- Understanding Parts of the Component, 77-78**
- Understanding Simple Row Filtering, 601-602**
- Understanding Structured Reference Syntax Rules, 570-571**
- Understanding Structured References, 569**
- Understanding the Axis Model, 577-578**
- Understanding the Data Model, 575-577**
- Understanding the Optional Interfaces, 584-585**
- Understanding the Printing Options, 826-831**
- Understanding the Product, 45**
- Understanding the Selection Model, 578-579**
- Understanding the Sheet Model Classes and Interfaces, 573-575**
- Understanding the Span Model, 579**
- Understanding the Spread Wizard, 33**
- Understanding the Style Model, 579-582**
- Understanding the Underlying Models, 573**
- Understanding Touch Messages, 1003**
- Using a Bound Data Source, 953-954**
- Using a Cell Comparison Validator, 529-530**
- Using a Character Format Validator, 530-531**

- Using a Circular Reference in a Formula, 647-649**
- Using a Pair Validator, 535-536**
- Using a Regular Expression Validator, 537-538**
- Using a Required Field Validator, 538-539**
- Using a Required Type Validator, 539-540**
- Using a String Comparison Validator, 531-532**
- Using a Surrogate Character Validator, 540**
- Using a Text Length Validator, 540-541**
- Using a Touch Keyboard, 1003-1004**
- Using a Value Comparison Validator, 532-533**
- Using an Encoding Validator, 533-534**
- Using an Outline (Range Group) of Rows or Columns, 632-634**
- Using an UnBound Data Source, 954-955**
- Using Auto Row Height, 182-183**
- Using Automatic Sorting, 589-590**
- Using Conditional Formatting of Cells, 486**
- Using Custom Filter Indicator Images, 619-620**
- Using DataTable Formula, 649-653**
- Using Drag Operations to Fill Cells, 522**
- Using Edit Mode and Focus, 508**
- Using Excel-compatible Keyboard Shortcuts, 779-781**
- Using External Variables with Text Box Control, 663-665**
- Using Grouping, 624-625**
- Using Input Maps with Action Maps, 770-774**
- Using Language Package, 685**
- Using Operators and Special Items, 569-570**
- Using Satellite Assemblies for Languages, 33**
- Using Serialization, 805-806**
- Using Smart Tags Drop-Down, 71-73**
- Using Structured References, 571-572**
- Using Table Filters, 555-558**
- Using the Array Formula, 665-666**
- Using the Chart Control, 921-923**
- Using the Chart Control on sheet, 914-919**
- Using the Excel Compatibility Input Maps, 781-782**
- Using the Exclude List Validator, 534**
- Using the Include List Validator, 534-535**
- Using the Range Validator, 536-537**
- Using the Spread Wizard, 34-35**
- Using the Touch Menu Bar, 1004-1008**
- Using Touch Support, 1008-1009**
- Using Touch Support when Moving Columns or Rows, 1019-1020**

- Using Touch Support when Resizing Columns or Rows, 1020-1021**
- Using Touch Support when Sorting, 1025**
- Using Touch Support with AutoFit, 1009**
- Using Touch Support with Cell Notes, 1009**
- Using Touch Support with Charts, 1009-1010**
- Using Touch Support with Clipboard Operations, 1010**
- Using Touch Support with Drag and Fill, 1010-1012**
- Using Touch Support with Drop-Down Elements, 1012-1013**
- Using Touch Support with Editable Cells, 1013-1014**
- Using Touch Support with Filtering, 1016-1017**
- Using Touch Support with Grouping, 1018-1019**
- Using Touch Support with InputMan Cells, 1014-1016**
- Using Touch Support with Range Grouping, 1017-1018**
- Using Touch Support with Scrolling, 1021-1023**
- Using Touch Support with Selections, 1023-1024**
- Using Touch Support with Shapes, 1024**
- Using Touch Support with the Tab Strip, 1025-1026**
- Using Touch Support with Viewports, 1025**
- Using Touch Support with Zooming, 1026-1027**
- Using Validation, 527-528**
- Using Validation in Cells, 528-529**
- Using Verbs in the Properties Window, 69-71**
- Using Visible Indicators in the Cell, 546**
- Using Windows Regional Settings or Options, 33**
- Using XP Themes with the Component, 499-500**
- Validating User Input, 541-545**
- Vari Sparkline, 747-750**
- Walls, 900-901**
- Waterfall Charts, 881-883**
- Ways to Improve Performance, 86-87**
- Working with 1-Based Indexing, 162**
- Working with a SubEditor, 413-414**
- Working with BuiltIn Dialogs, 108-117**
- Working with Cell Format Strings, 235-242**
- Working with Collection Editors, 73-74**
- Working with Deselections, 520-522**
- Working With Dynamic Array Formulas, 666-679**
- Working with Editable Cell Types, 331-332**
- Working with Graphical Cell Types, 366**
- Working with Hierarchical Data Display, 436-441**
- Working with Images, 999-1002**
- Working with Multiple Sheets, 121-122**
- Working with Pattern and Gradient Fill Effects, 297-298**

Working with Scroll Bars, 89
Working with Selections, 517-519
Working With Shapes (Enhanced Shape Engine), 967-980
Working with Shapes in Code, 962-964
Working With Slicers, 102-108
Working with Sparklines, 710-713
Working with the Active Cell, 230-231
Working with the Active Sheet, 120-121
Working with the Component, 55
Working with the Formula Text Box, 679-682
Wrapping the Header Text, 204-205
XY Bubble Charts, 883-884
XY Line Charts, 884-885
XY Plot Types, 883
XY Point Charts, 884
XY Stripe Charts, 885
XYZ Line Charts, 887-888
XYZ Plot Types, 885-886
XYZ Point Charts, 886-887
XYZ Stripe Charts, 889
XYZ Surface Charts, 888-889
Y Plot Types, 864-865
エラー
 アイコン, 545-546
 行, 545-546
シェイプ, カメラ, 964-966
スパークライン, グルーピング, 710-713
スパークライン, グルーピング解除, 710-713
スパークライン, スイッチ, 710-713
スパークライン, 横軸, 706-708
スパークライン, 縦軸, 706-708
スパークライン, 追加, 704-706
データバー, 488-490
ポイント, 708-710
マーカー, 708-710
リサイズ, 273-274
ルール
 データバー, 488-490
 トップ10, 493-494
 平均, 493-494
入力マップ, 782-784, 781-782
 xml, 778-779

保存, 778-779

読込, 778-779

縮小して全体を表示, 273-274