## 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.

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

For sample information, refer to **Getting Started**.

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)**.

# 1 Table of Contents

## Getting Started

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

- **Handling Installation**
- **Working with the Component**
- **Understanding the Spread Wizard**
- **Getting More Practice**
- **Tutorial: Creating a Checkbook Register**

For more in-depth explanation of the product, refer to **Understanding the Product**.

## Handling Installation

The following tasks involve installing and redistributing the product:

- **Installing the Product**
- **Licensing a Trial Project after Installation**
- **End-User License Agreement**
- **Creating a Build 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 **Programs** -> **GrapeCity**-> **Spread Studio 10** -> **Spread Windows Forms**-> **SpreadWindowsFormsReadMe**. Select the Read Me under the GrapeCity 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 \GrapeCity\Spread Studio 10\Docs\Windows Forms under the program files directory and then double-click the readme.chm file.

You can also access the Read Me on the web site.

## Licensing a Trial Project after Installation

To license Windows Forms projects made with the trial version do the following:

1. Ensure that Spread is licensed on the machine by following the installation steps in the ReadMe on the web site.
2. Open the project in Microsoft Visual Studio.
3. Open the Visual Studio **Build** menu and select **Rebuild Solution**.
4. The executable application is now licensed and no nag screens or evaluation banners appear when you run it. You can distribute the application to unlicensed machines and no nag screens or evaluation banners appear.

If you have installed a trial version of the product, you can license the product using the product splash screen in the Microsoft Visual Studio project.

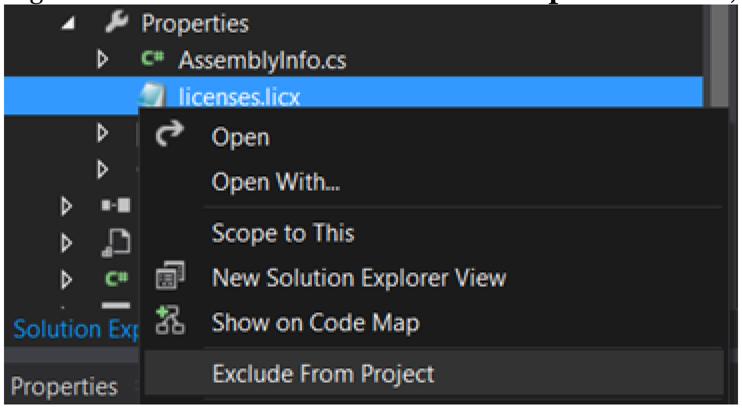
## End-User License Agreement

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

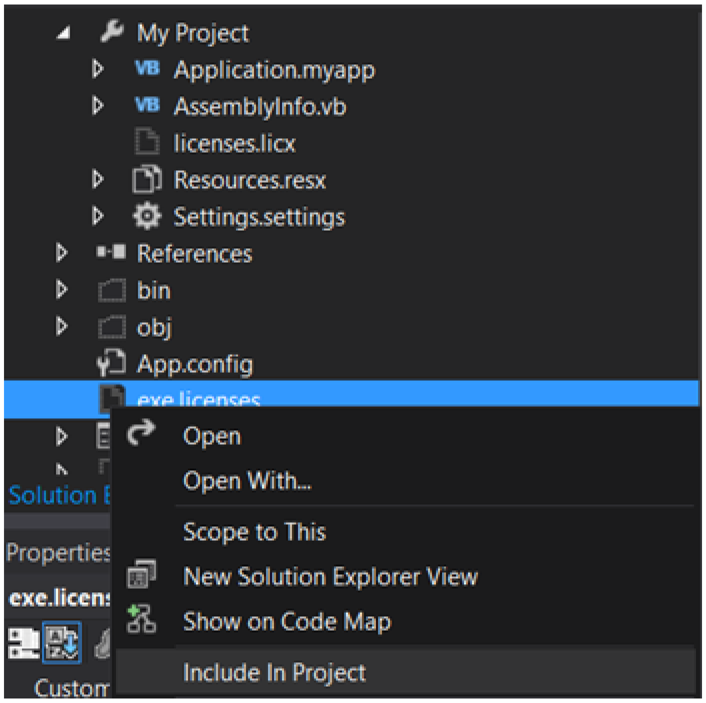
## Creating a Build License

You can create a build license to use on a build machine.

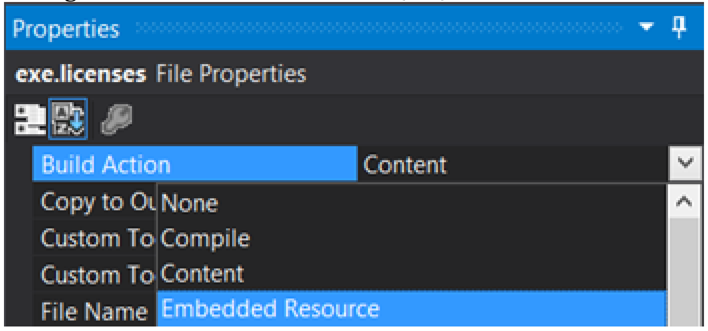
Licenses are built using the license compiler tool (lc.exe) to produce a special resource file with the .licenses file extension. Visual Studio VB.NET and C# projects automatically handle compiling the licenses.licx in the project to produce the .licenses resource file, which is linked into the target executable. The components' run-time license keys in that licenses resource file are loaded and verified when the first instance of each component with the LicenseProvider attribute is created in the application. You can remove the licenses.licx from your Visual Studio project and add the .licenses resource in its place using the following steps:

1. Build the project using the licensed components on a developer machine which is licensed for development with all the components referenced in the project (this creates the .licenses resource).
2. Find the licenses.licx in the Solution Explorer window. You can use the **Show All Files** toolbar button to see it or expand the **Properties** folder.
3. Right-click the licenses.licx in the **Solution Explorer** window, and then select **Exclude From Project.**


4. Use Windows Explorer (outside Visual Studio) to find the .licenses file in the obj\{configuration} folder (obj\Debug or obj\Release). The file should have the name {target}.{ext}.licenses (for example: project1.exe.licenses).
5. Copy that file to the project folder and rename it to remove the target name (rename it from {target}.{ext}.licenses to {ext}.licenses). For example: project1.exe.licenses to exe.licenses.
6. In the Visual Studio Solution Explorer window, find the {ext}.licenses (you might need to refresh the window), then right-click the file and select **Include In Project**.



7. Change the **Build Action** for the {ext}.licenses from **Content** to **Embedded Resource**.



8. The project can now be built without requiring a developer license on the machine, since the license has already been built and linked into the project.

Note the following restrictions:

- The licenses resource contains the name of the target module encoded in its contents, so that licenses resource is specific to that particular project.
- The steps described above will not bypass any part of the design-time license enforcement. The developer license is still required to open forms containing instances of the licensed controls.
- If the licensed components in the project change, then special care should be taken. The licenses.licx should be added back to the project first, so that it does not get recreated (empty) by Visual Studio and cause type references (and embedded licenses in the resource) to be lost. After the new licensed components are added or changed in the licx, the above steps should be repeated.
- The above steps only apply for .NET managed code applications which use the standard .NET Framework component licensing model (ActiveX control licensing in managed .NET applications does not use this mechanism).

## 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 98
- Microsoft Windows 98 SE
- Microsoft Windows ME
- Microsoft Windows 2000 (SP3)
- Microsoft Windows Server 2003
- Microsoft Windows Server 2008
- Microsoft Windows Server 2012 R2
- Microsoft Windows XP (SP2)
- Microsoft Windows Vista
- Microsoft Windows 7
- Microsoft Windows 8
- Microsoft Windows 8.1
- Microsoft Windows 10

*Software*
You must have the Microsoft .NET Framework installed.

**Files to Distribute**

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

- The following assemblies that come with Spread Windows Forms:
  - FarPoint.CalcEngine.dll
  - FarPoint.Excel.dll
  - FarPoint.PluginCalendar.WinForms.dll
  - FarPoint.Win.dll
  - FarPoint.Win.Spread.dll
  - FarPoint.Localization.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. This feature is only available with the .NET 2.0 Framework.
- 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 4.0 version of Spread Windows Forms**

For developing applications with the .NET 4.0 version of Spread Windows Forms, you must have the following system and software specifications:

***Operating System***
One of the following:

Microsoft Windows 2000 Professional (SP4)

Microsoft Windows 2000 Server

Microsoft Windows 2003 Server (SP1)

Microsoft Windows Server 2012 R2

Microsoft Windows 2008

Microsoft Windows XP (SP2)

Microsoft Windows Vista

Microsoft Windows 7

Microsoft Windows 8

Microsoft Windows 8.1

Microsoft Windows 10

***Software***
The release version of the Microsoft .NET 4.0 Framework or later.

These are 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).

## Working with the Component

The tasks involved with using the Spread component on a Windows Form are:

- **Adding a Component to a Visual Studio 2015 or 2017 Project**
- **Adding a Component to a Visual Studio 2013 Project**
- **Adding a Component to a Project**
- **Understanding Parts of the Component**
- **Using Smart Tags Drop-Down**
- **Using Verbs in the Properties Window**
- **Working with Collection Editors**
- **Adding Support for High DPI Settings**

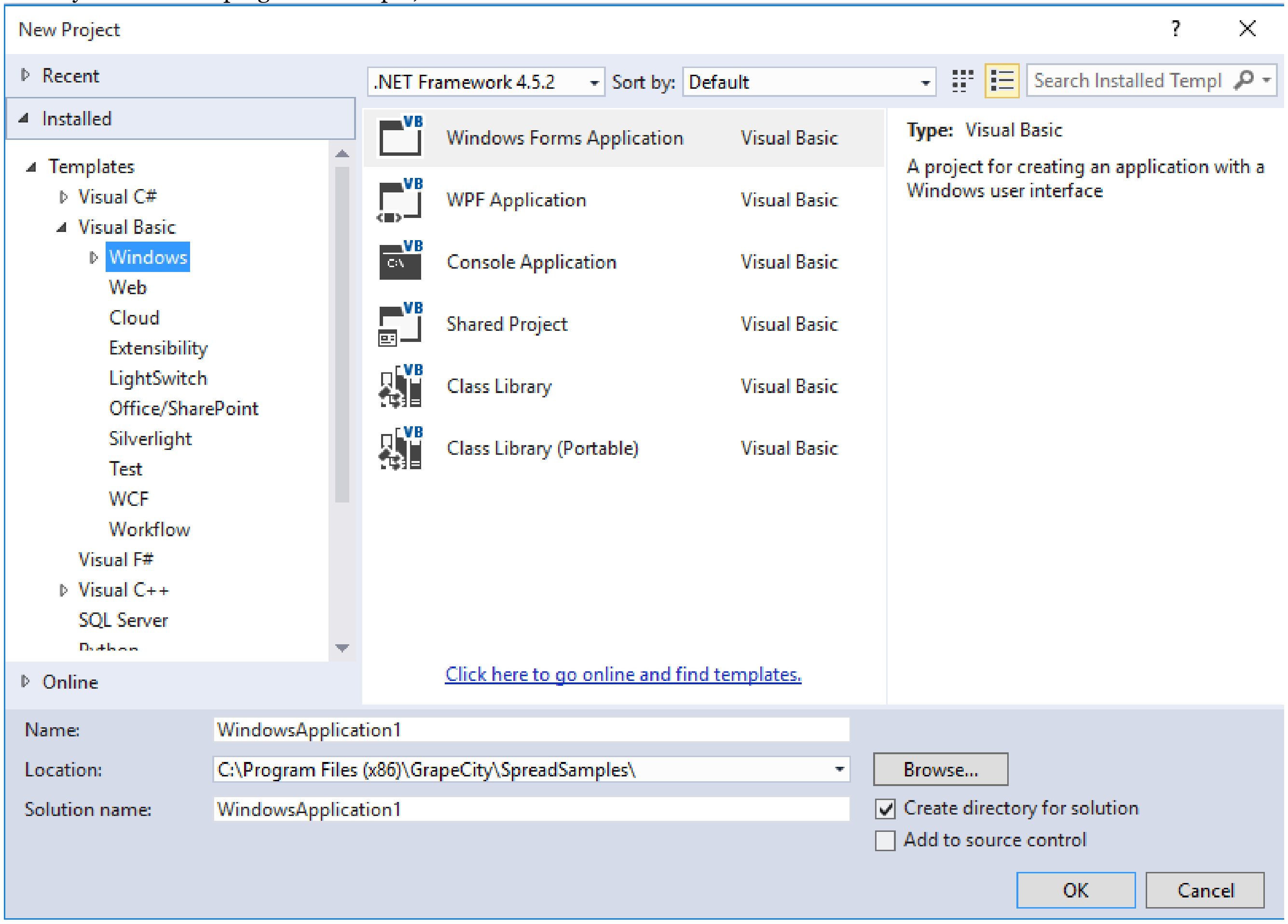
## Adding a Component to a Visual Studio 2015 or 2017 Project

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**.



Type: Visual Basic

A project for creating an application with a Windows user interface

Name: WindowsApplication1

Location: C:\Program Files (x86)\GrapeCity\SpreadSamples\

Solution name: WindowsApplication1

[✓] Create directory for solution

[ ] Add to source control

   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 **GrapeCity 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.

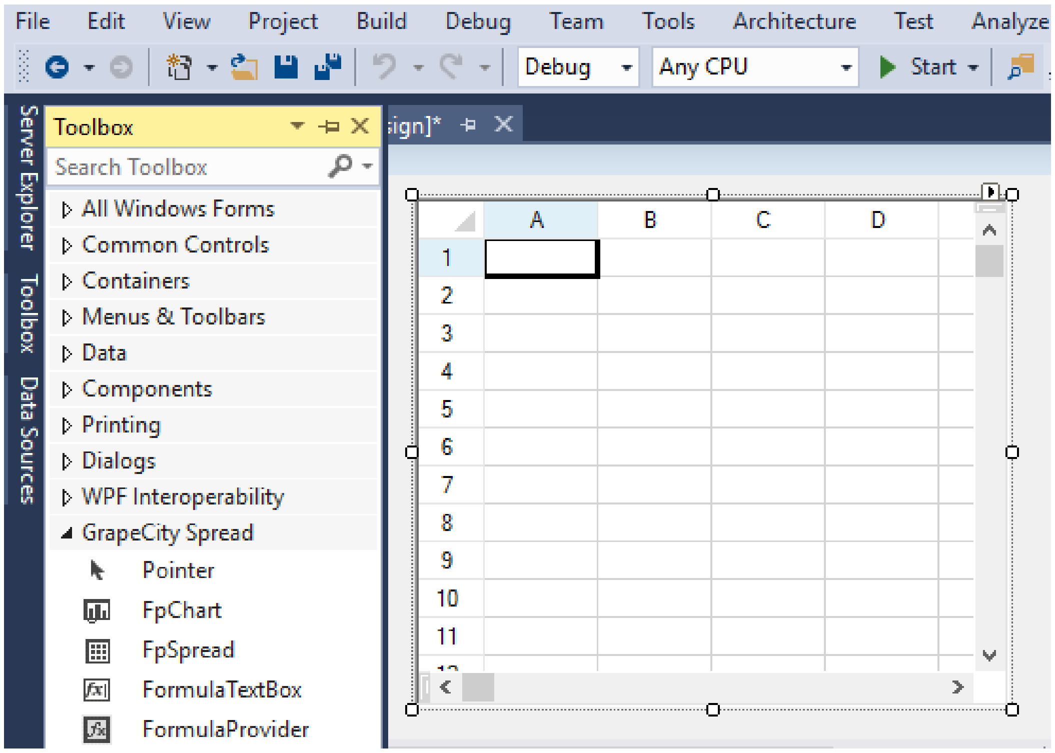
If the Spread component is not displayed in the list of components, click **Browse** and browse to the installation path for the Spread Windows Forms component. Once there, select the FarPoint.Win.Spread.dll and click **Open**. The Spread component is now displayed in the list of components. Select it 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 **GrapeCity 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.



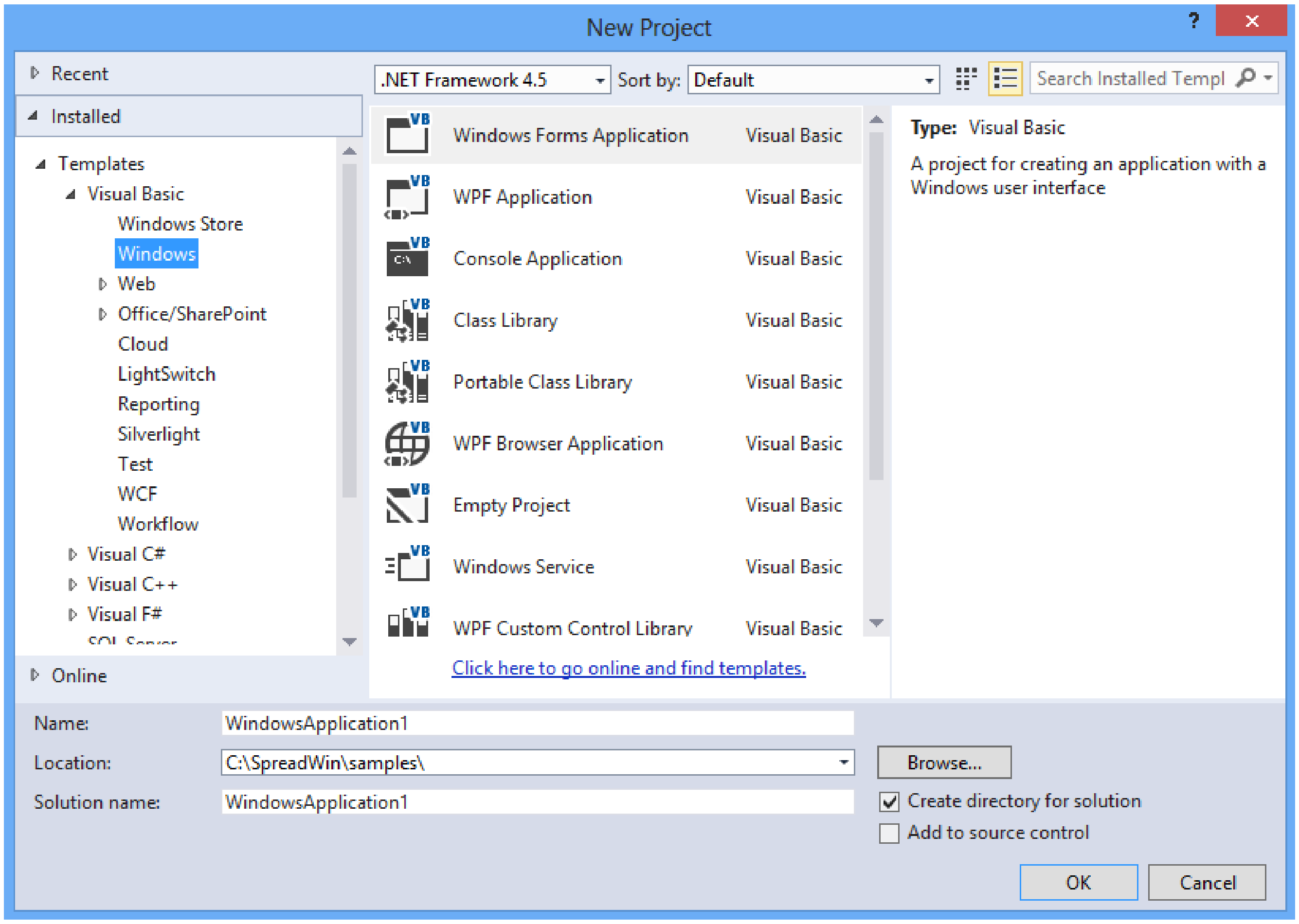
You have added the Spread component to the project.

## Adding a Component to a Visual Studio 2013 Project

If you are new to the .NET platform, you might be unfamiliar with how to start a new project using a component. To use the Spread Windows Forms product, you need 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** or select **New Project**... under **Start**.
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**.


New Project
Recent
Installed
Templates
Visual Basic
Windows Store
Windows
Web
Office/SharePoint
Cloud
LightSwitch
Reporting
Silverlight
Test
WCF
Workflow
Visual C#
Visual C++
Visual F#
Online
.NET Framework 4.5
Sort by: Default
Search Installed Templ
Windows Forms Application Visual Basic
WPF Application Visual Basic
Console Application Visual Basic
Class Library Visual Basic
Portable Class Library Visual Basic
WPF Browser Application Visual Basic
Empty Project Visual Basic
Windows Service Visual Basic
WPF Custom Control Library Visual Basic
Click here to go online and find templates.
Type: Visual Basic
A project for creating an application with a Windows user interface
Browse...
OK
Cancel
Name: WindowsApplication1
Location: C:\SpreadWin\samples\
Solution name: WindowsApplication1
[✓] Create directory for solution
[ ] Add to source control

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.

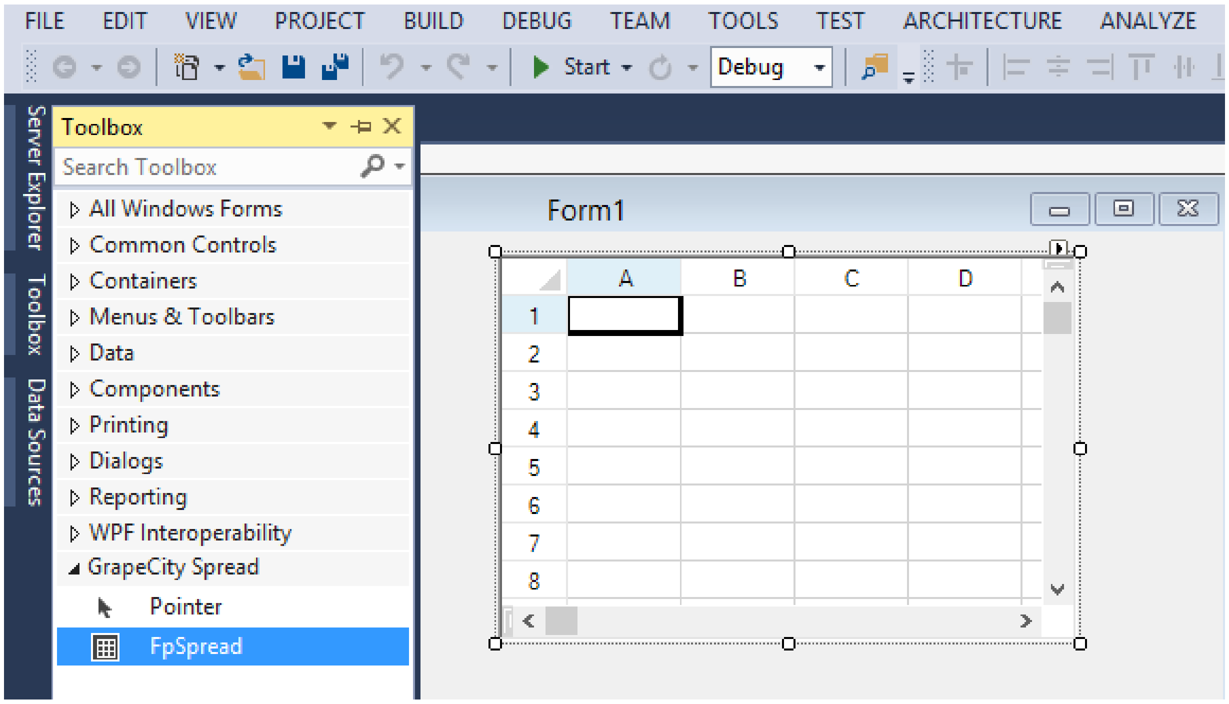
The next step is to add the Spread component to the toolbox. This only has to be done once.

1. If the Toolbox is not displayed, from the **View** menu choose **Toolbox**.
2. Once the Toolbox is displayed, look in the **GrapeCity 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 **GrapeCity 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 now look similar to the picture shown here.

You have added the Spread component to the project.

## Adding a Component to a Project

If you are new to the .NET platform, you might be unfamiliar with how to start a new project using a component. To use the Spread Windows Forms product, you need to add the component to a project in Visual Studio .NET.

The following general steps are for Microsoft Visual Studio 2012 or earlier.

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 **Project Type** area, select a project type depending on the language environment in which you are developing. For example, in the **Project Types** list, choose **Visual C# Projects**.
4. In the **New Project** dialog,
   a. In the **Project Types** list, choose **Visual C# Projects** or **Visual Basic Projects** depending on the language you are using.
   b. In the **Templates** list, choose **Windows Application**.
   c. In the **Name** box, type the name of the new project. The default is WindowsApplication1 for the first Windows Forms application.

d. In the **Location** box, leave the location path as the designated path, or click **Browse** to change the path to a new directory.
e. Click **OK**.

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

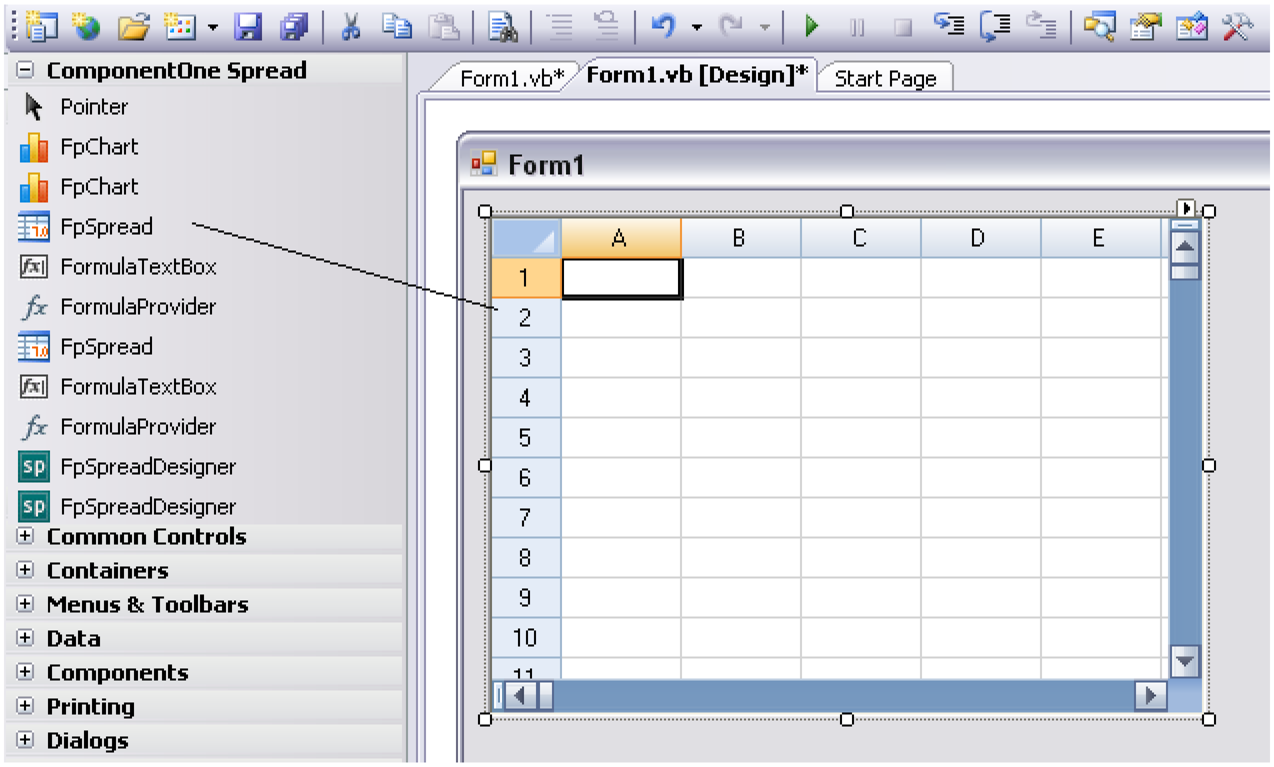
5. In the Solution Explorer, right-click on the form name, Form1. Choose **Rename** from the pop-up menu, then type the new form name you prefer for the new form name.

The next step is to add the Spread component to the Toolbox. This only has to be done once.

1. If the Toolbox is not displayed, from the **View** menu choose **Toolbox**.
2. Once the Toolbox is displayed, look in the **GrapeCity 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, right-click in the Toolbox, and from the pop-up menu choose **Customize Toolbox**, **Add/Remove Items**, or **Choose Items** (depending on the version of Visual Studio).
4. In the **Customize Toolbox** dialog, click the **.NET Framework Components** tab.
5. 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 FpSpread 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.
6. 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 Windows Forms (or whatever category to which you added it), select the Spread 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.

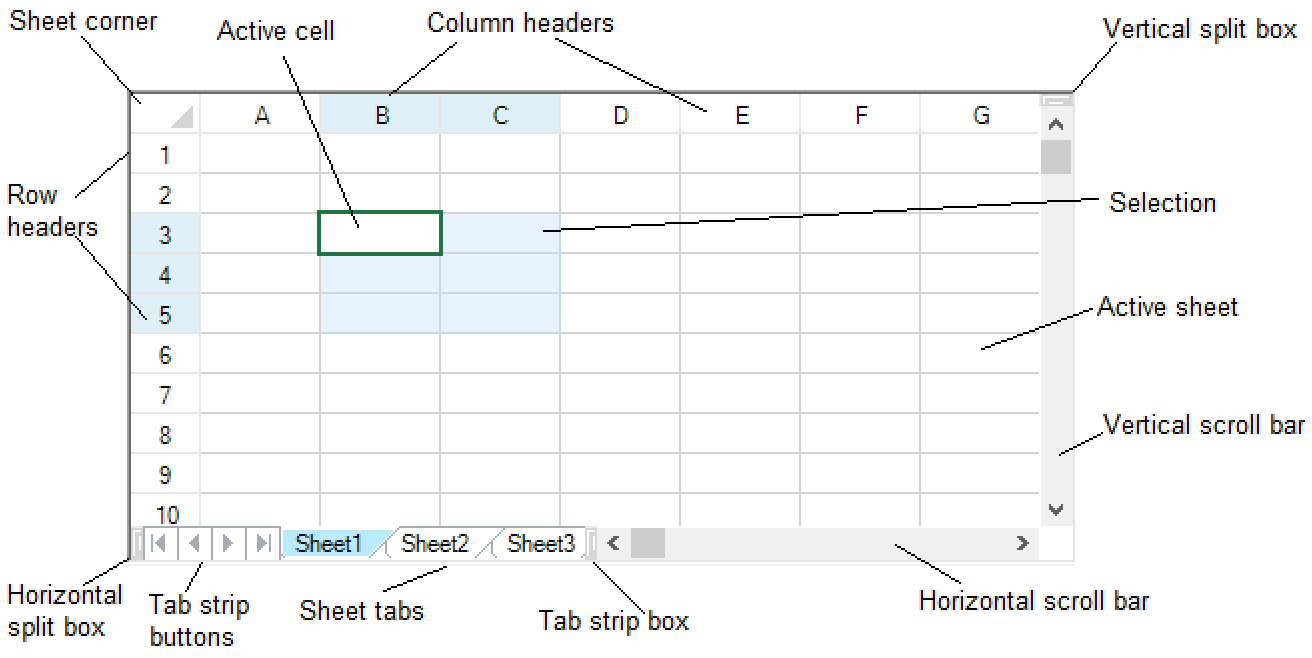


Your project should now look similar to the picture shown here.

You have added the Spread component to the project.

## Understanding Parts of the Component

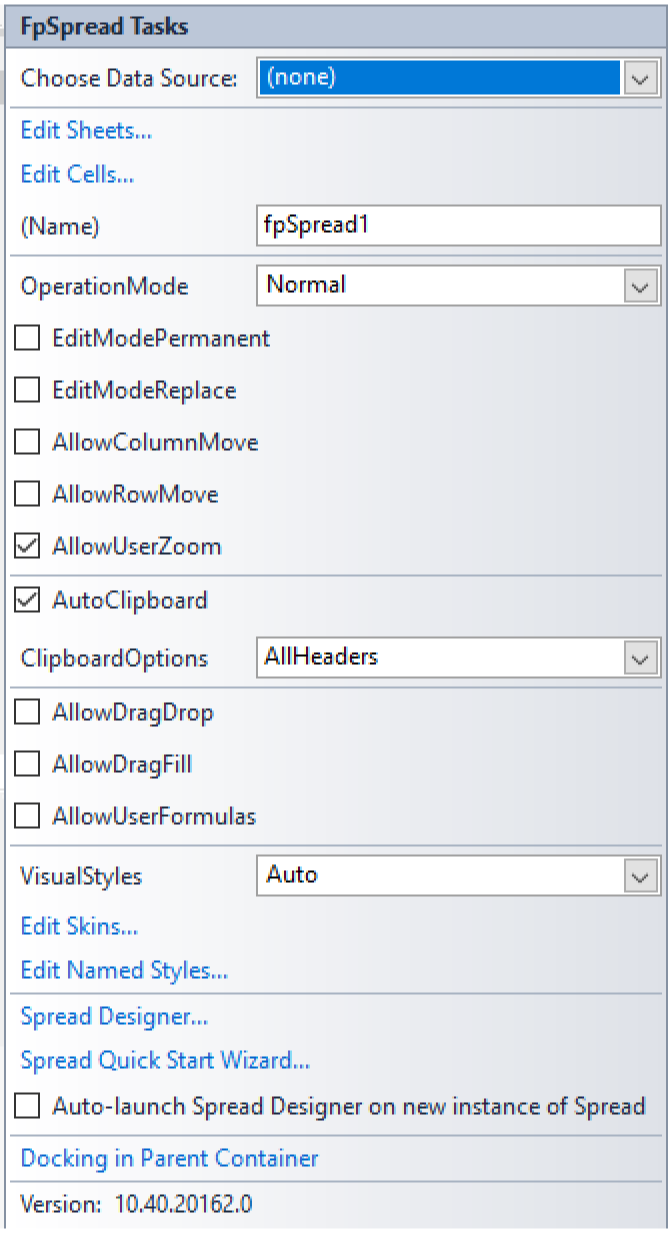
The Spread component is made up of the spreadsheet that displays the data along with scroll bars and, if multiple sheets, sheet tabs in a tab strip. The figure below shows the major parts of the Spread component. Several of these can be hidden, but this shows the default display.



| For more information on ... | Refer to ... |
|---|---|
| sheet corner | **Customizing the Sheet Corner Appearance** |
| sheet tabs | **Customizing the Sheet Name Tabs of the Component** |
| scroll bars | **Customizing the Scroll Bars of the Component** |
| row and column headers | **Customizing the Appearance of Headers** |
| focus indicator (of active cell) | **Customizing the Focus Indicator for a Cell** |
| selections | **Customizing User Selection of Data** |
| active sheet | **Working with the Active Sheet** |

## Using Smart Tags Drop-Down

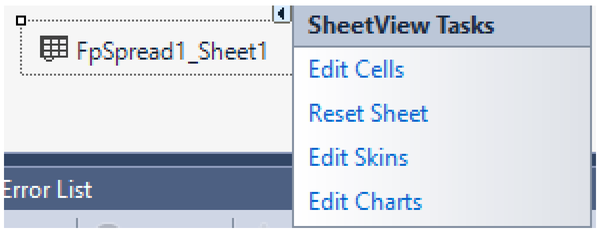
You can perform any of several tasks, launch various editors, and set various properties from the smart tags drop-down available from the Spread component on a Form in Visual Studio .NET. The smart tag is the arrow icon at the top, right edge of the control. The Spread tasks available in the smart tags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



| Task | Explanation or Reference |
|---|---|
| Choose Data Source | Refer to **Managing Data Binding**. |
| Edit Sheets | Refer to **Customizing the Individual Sheet Appearance** and **Customizing Sheet Interaction**. |
| 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<br>• EditModePermanent | Refer to **Allowing the User to Move Rows or Columns**, **Allowing the User to Zoom the Display of the Component**, **Filling Cells with Drag and Drop**, |

| | |
|---|---|
| • EditModeReplace<br>• AllowColumnMove<br>• AllowRowMove<br>• AllowUserZoom<br>• AllowDragDrop<br>• AllowDragFill<br>• AllowUserFormulas | **Filling Cells with Drag and Fill**, **Allowing the User to Enter Formulas**, and **Understanding Edit Mode in a Cell**. |
| 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.

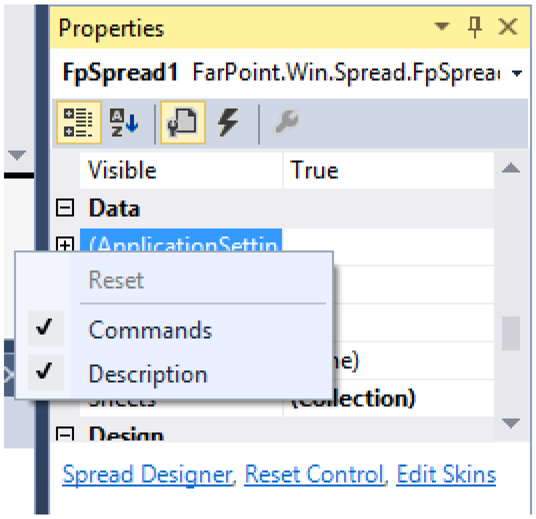


| Tasks | Description |
|---|---|
| Edit Cells | This opens the Cell Editor and allows you to edit various properties of the selected cell or cells of a sheet. For more information on setting cell properties, refer to **Customizing the Appearance of a Cell**. |
| Reset Sheet | This restores all the settings 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)**. |

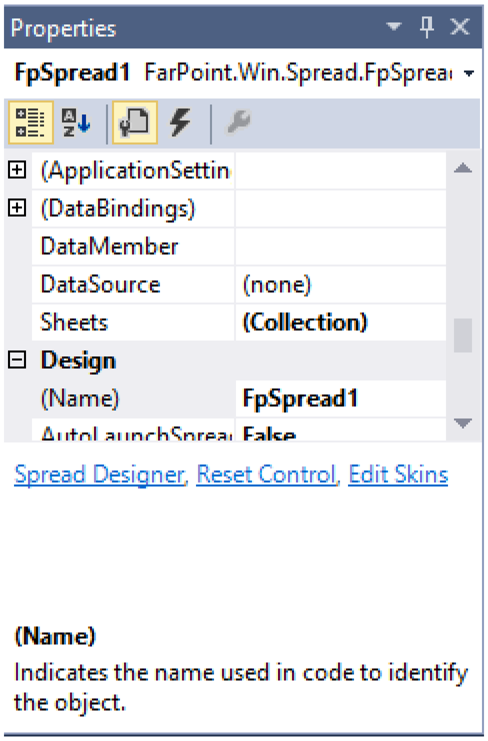
## Using Verbs in the Properties Window

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.

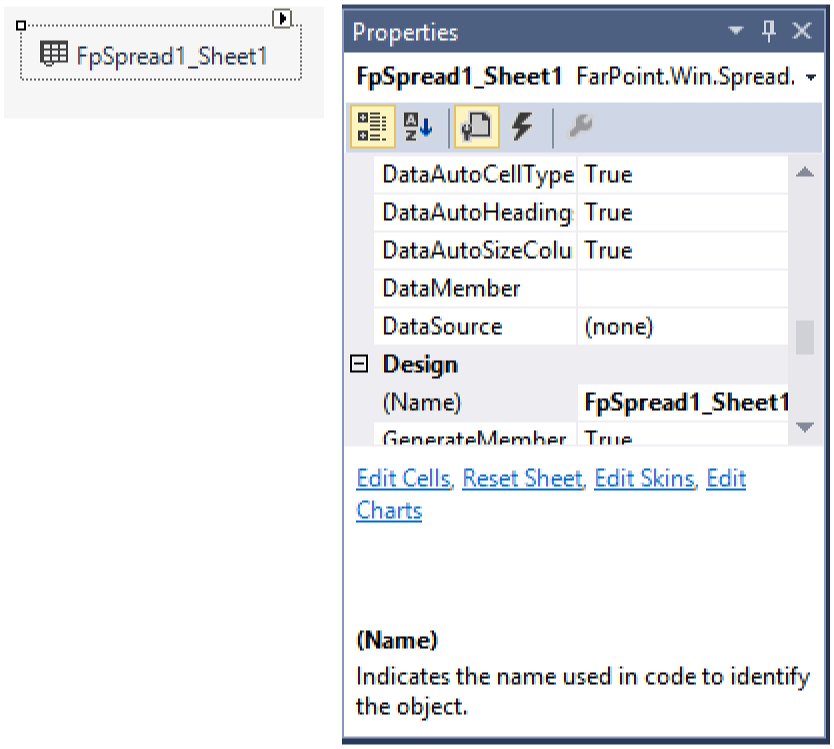


The following table summarizes the Spread component verbs:

| 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.



| | |
|---|---|
| DataAutoCellType | True |
| DataAutoHeading | True |
| DataAutoSizeColu | True |
| DataMember | |
| DataSource | (none) |
| **Design** | |
| (Name) | **FpSpread1_Sheet1** |
| GenerateMember | True |

Edit Cells, Reset Sheet, Edit Skins, Edit Charts

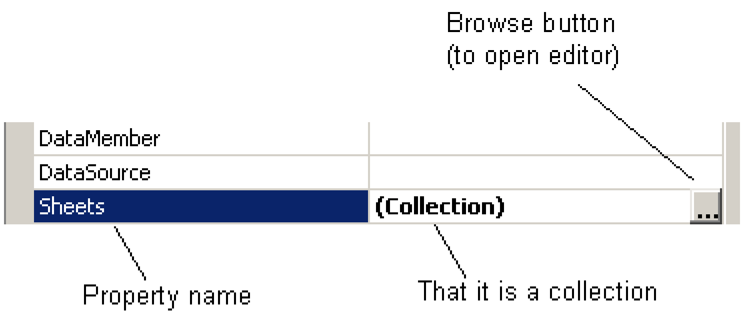
**(Name)**
Indicates the name used in code to identify the object.

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)**. |

## 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.

You can refer to the following steps for a general example of how to enable DPI support in .NET 4.5.2.

1. Add code such as the following to the application manifest.

**Code**

```
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0"
xmlns:asmv3="urn:schemas-microsoft-com:asm.v3" >
  <asmv3:application>
    <asmv3:windowsSettings
xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSettings">
      <dpiAware>true</dpiAware>
    </asmv3:windowsSettings>
  </asmv3:application>
</assembly>
```

2. 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.

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>
```

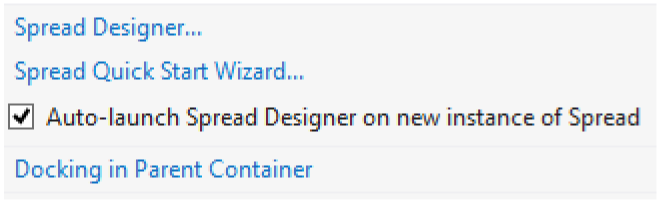
## Understanding the Spread Wizard

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

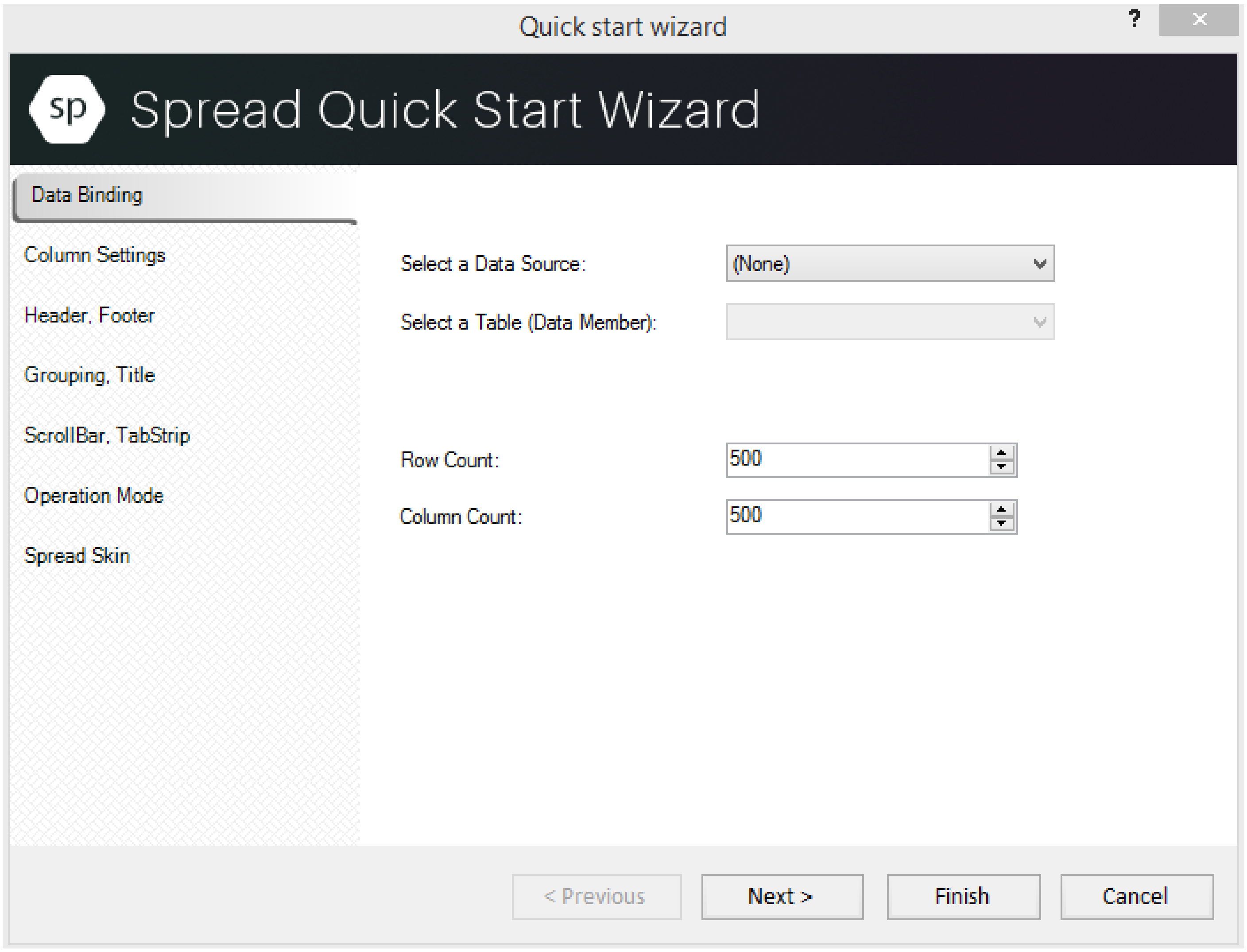
You can launch the Spread Wizard from the Smart Tags on the FpSpread component on the form in Visual Studio as shown in this figure. The smart tag is the arrow icon at the top, right edge of the control.


Spread Designer...
Spread Quick Start Wizard...
Auto-launch Spread Designer on new instance of Spread
Docking in Parent Container

## 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**.



Select a Data Source: (None)
Select a Table (Data Member):
Row Count: 500
Column Count: 500

## 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**

## 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**->**GrapeCity**->...->**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 of the Component**
- **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**
- **Customizing the Sheet Name Tabs of the Component**
- **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**

## Getting Technical Support

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

- Help and other documentation files installed with the product.
  For instructions for accessing the help and other documentation, see **Finding the Documentation**.
- Product forum at http://sphelp.grapecity.com/forums/forum/spreadsheets/

If you cannot find the answer using these sources, please contact Technical Support using one of these methods:

| | |
|---|---|
| Web site: | http://sphelp.grapecity.com/ |
| E-mail: | spread.support@grapecity.com |
| Fax: | (412) 681-4384 |
| Phone | (412) 681-4738 |

Technical Support is available between the hours of 9:00 a.m. and 5:00 p.m. Eastern time, Monday through Friday.

## 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"
```

4. 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**

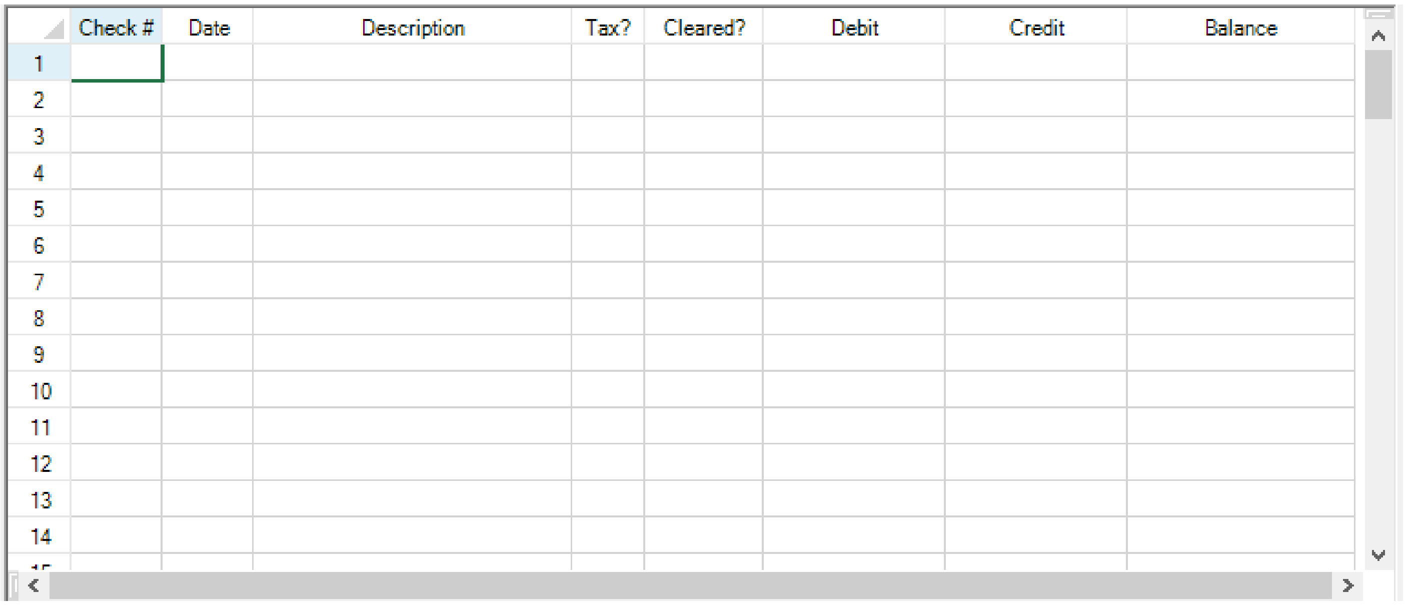
**C#**

```
// 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
```

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



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
```

3. 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
```

4. 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
```

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

**Example**

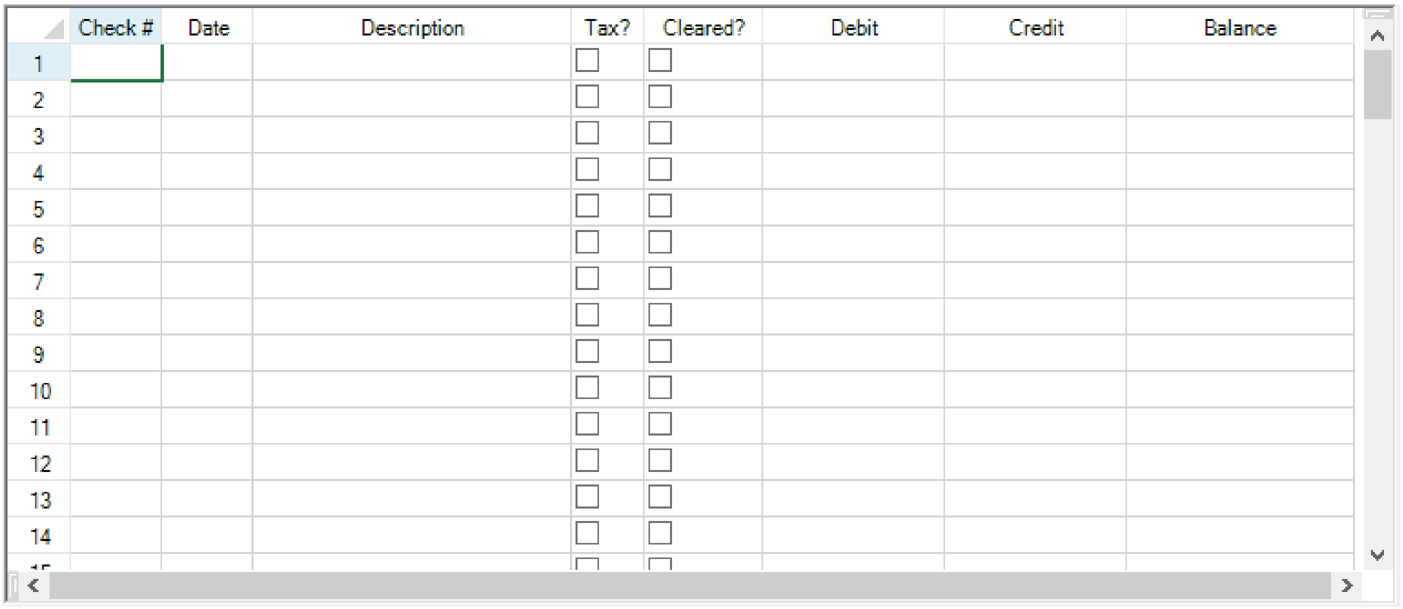
**C#**

```
// 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
```

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



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**

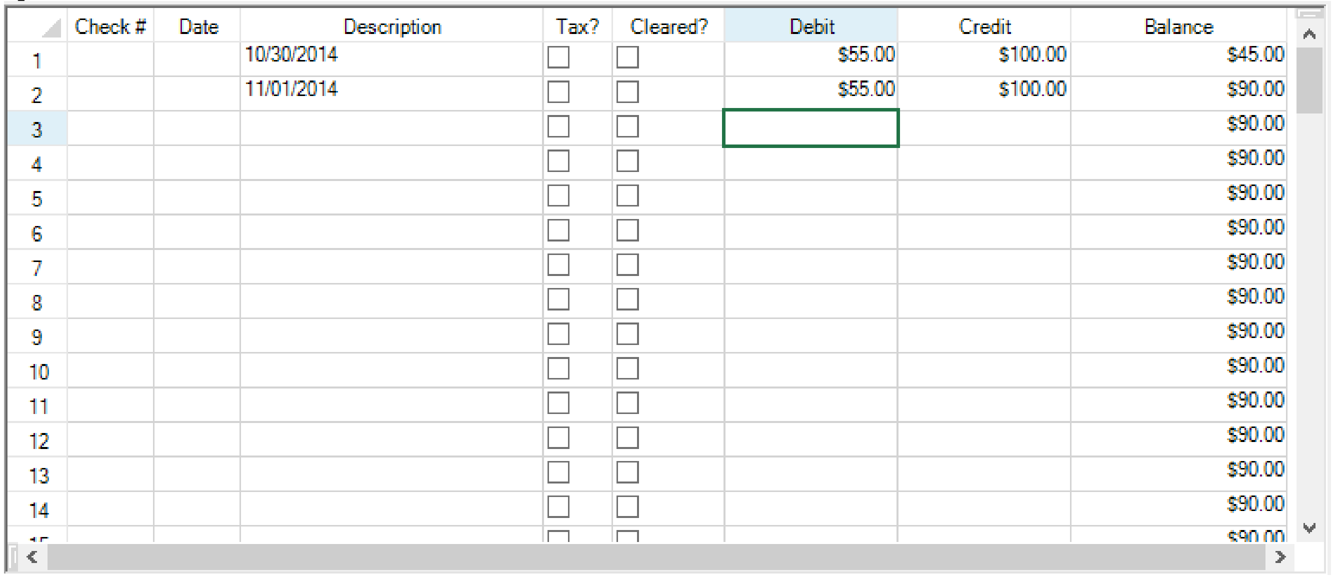
   **C#**

```
// 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 | ☐ | ☐ | $55.00 | $100.00 | $45.00 |
| 2 | | | 11/01/2014 | ☐ | ☐ | $55.00 | $100.00 | $90.00 |
| 3 | | | | ☐ | ☐ | | | $90.00 |
| 4 | | | | ☐ | ☐ | | | $90.00 |
| 5 | | | | ☐ | ☐ | | | $90.00 |
| 6 | | | | ☐ | ☐ | | | $90.00 |
| 7 | | | | ☐ | ☐ | | | $90.00 |
| 8 | | | | ☐ | ☐ | | | $90.00 |
| 9 | | | | ☐ | ☐ | | | $90.00 |
| 10 | | | | ☐ | ☐ | | | $90.00 |
| 11 | | | | ☐ | ☐ | | | $90.00 |
| 12 | | | | ☐ | ☐ | | | $90.00 |
| 13 | | | | ☐ | ☐ | | | $90.00 |
| 14 | | | | ☐ | ☐ | | | $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

Spread Windows Forms provides a customizable, extendable, and object-oriented spreadsheet component for use in the Microsoft NET framework. Spread Windows Forms also provides Excel support and customization down to the cell level, with a capable user-interface design and back-end calculation efficiency. The following sections explain some of the underlying concepts for this unique and powerful product.

- **Product Overview**
- **Feature Overview**
- **Namespaces Overview**
- **Concepts Overview**

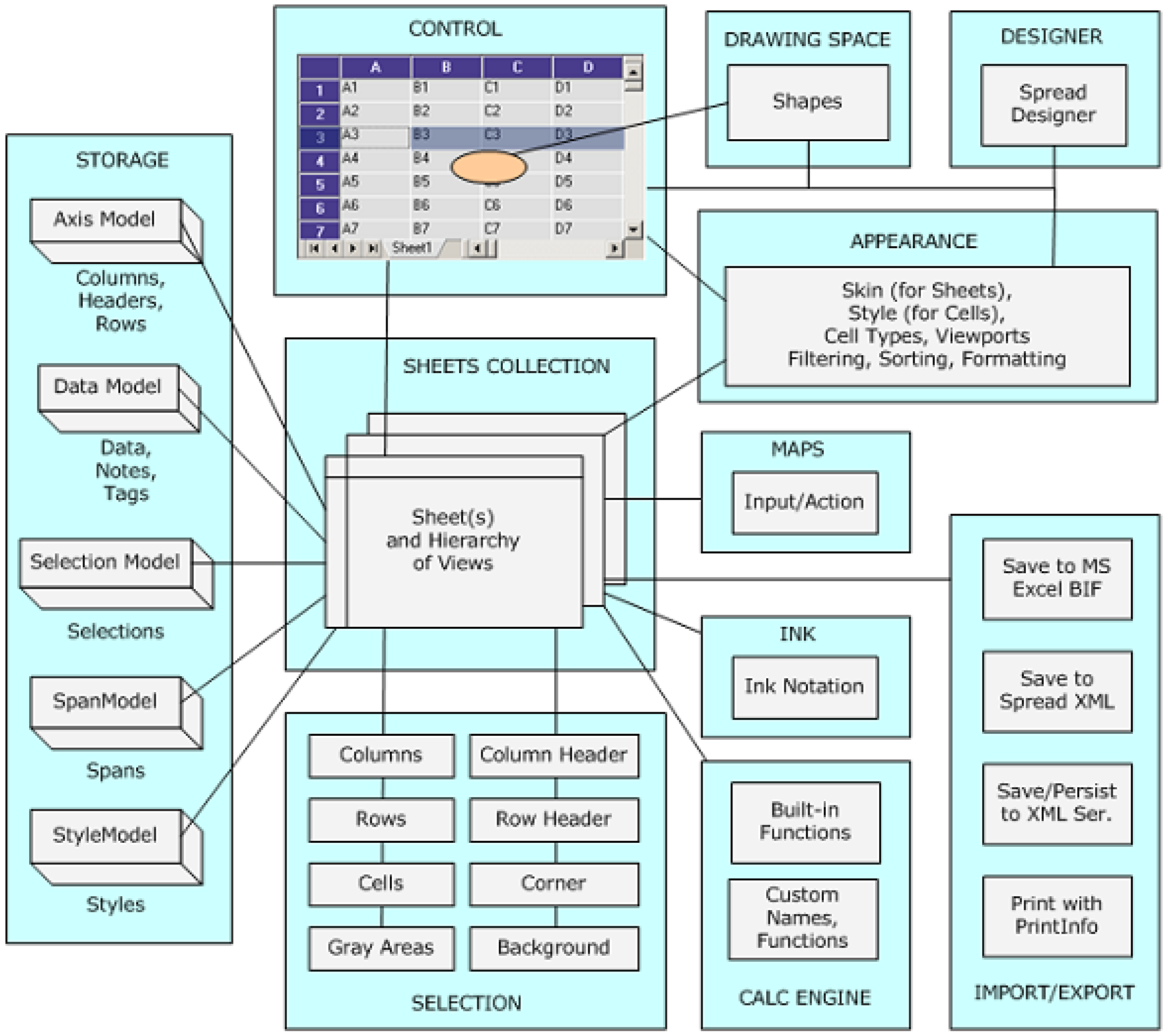
## Product Overview

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. A single Spread component supports many sheets, rows, and columns. Cross-sheet referencing allows calculations to make use of data and formulas on a variety of sheets. Spread Windows Forms uses dot notation for object-oriented coding in .NET.

The Spread component may be dropped on a Windows Form and customized for a range of applications. You can customize the appearance and the user interaction in a variety of ways. With a built-in Designer, you can quickly create a prototype or customize your finished design. With most of the Spread's appearance and functionality based on underlying models, the advanced developer has complete control over the component. For details on the Spread Designer, refer to **Spread Designer Guide (on-line documentation)**.

Import and export capabilities provide another source of flexibility when developing and exchanging designs. 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. For more information on exporting to (and importing from) a file, refer to the **Import and Export Reference (on-line documentation)**.

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



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

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. For more information on styles for cells, refer to **Creating and Applying a Style for Cells**. For more information on skins for sheets, refer to **Creating a Custom Skin for a Sheet**.

For more information about shapes, refer to **Customizing Drawing**.

For more information on the formulas and functions that can be entered in a cell, refer to the **Managing Formulas in Cells**.

For more information on the parts of the interface, refer to **Understanding Parts of the Component**.

For a list of many of the features, see **Feature Overview**.

## Feature Overview

Spread Windows Forms introduces some powerful features, as described in the following topics. Each topic refers to other topics in the documentation that provide more information.

- **Camera Shapes**
- **Cell Types for Cell Functionality**

- **Chart Controls on a Sheet**
- **Child Controls on a Sheet**
- **Column Footers and Group Footers**
- **Conditional Formatting**
- **Data Binding**
- **Excel Support with Import and Export Capabilities**
- **Filtering Data on a Sheet**
- **Formula Text Box (Formula Bar)**
- **Formula Provider Control**
- **Functions and Formulas**
- **Gradients for Button Cells, Headers, and More**
- **Grouping Rows in the Display**
- **Grouping Rows or Columns in an Outline**
- **Headers with Multiple Columns and Rows**
- **Hierarchical Display**
- **HitTest for Locating the Cursor**
- **Indicators and Icons in the Interface**
- **Ink Notation Support**
- **Keyboard Action Mapping**
- **Multiple Sheets**
- **Name Box Control**
- **Notes for Cells**
- **Panes or Viewports**
- **Printing and PDF**
- **Quick Start Wizard**
- **Right-To-Left Layouts**
- **Row Preview**
- **Searching and Search Dialog**
- **Shapes, Drawings, and Annotations (Freehand Drawing)**
- **Skins and Styles for Customized Appearance**
- **Sorting Rows or Columns**
- **Spannable Cells**
- **Sparklines**
- **Spread Designer**
- **Status Bar (on-line documentation)**
- **Tables**
- **Tab Strip and Sheet Name Tabs**
- **Text Rendering with GDI**
- **Title and SubTitle**
- **Touch Support**
- **Undo and Redo Actions**
- **Visual Styles for XP Themes**

## Camera Shapes

You can add camera shapes to the sheet. Camera shapes are created by taking a snapshot of the content in a range of cells. Changes in the cell range will update the content of the camera shape.

For more information, refer to **Creating Camera Shapes**.

## Cell Types for Cell Functionality

You can use the feature-rich set of cell types or extend the functionality by defining your own. Use any of the more than 20 pre-defined cell types or create your own to determine what kind of data can be entered into a cell, avoiding unnecessary checks and validations by the developer, and providing a natural way for your user to enter data.

You can set barcode cell types for displaying barcodes, fractions in cells using the properties added to the number cell type, and allow your users to select a color from a color picker cell.

GcDateTime, GcNumber, and GcTextBox cells are now available in addition to the standard Spread date-time, number, and text cells.

For more information about cell types, refer to **Customizing Interaction with Cell Types**.

## Chart Controls on a Sheet

You can add chart controls to a sheet. There are many different types and views of charts that you can create. You can use the Chart Designer, Spread Designer, or code to add a chart control.

For more information, refer to **Using the Chart Control**.

## Child Controls on a Sheet

You can host child controls on a sheet to provide more interaction with the user.

For more information, refer to **Placing Child Controls on a Sheet**.

## Column Footers and Group Footers

You can add column and group footers to the sheet. You can put information in the footer such as formulas or text.

For more information, refer to **Displaying a Footer for Columns or Groups**.

## Conditional Formatting

You can set up conditional formats within cells that determine the formatting of the cell based on the outcome of a conditional operation. You can use rules or comparison operators in the conditional format.

For more information, refer to **Using Conditional Formatting of Cells**.

## Data Binding

With Spread Windows Forms, you have many databinding options.

You can bind the spreadsheet to a data set to display and allow your users to edit information. Spread can automatically update the data set with the changes.

You can also allow part of your spreadsheet to be unbound. You can also bind to a range of cells.

For more information, refer to **Managing Data Binding**.

## Excel Support with Import and Export Capabilities

You can import data from and export data (and formatting) to Microsoft Excel, both in individual spreadsheets and entire workbooks. You can import and export entire spreadsheet(s) with data and formatting to and from XML. Several versions of Excel are supported and several file types, including XLS, XLSX, CSV, and TXT.

For more details about what happens during importing or exporting, refer to the **Import and Export Reference (on-line documentation)**.

For more information about saving and loading files, refer to **Saving Data to a File** and **Opening Existing Files**.

## Filtering Data on a Sheet

You can customize the user experience for filtering data on a sheet. With row filtering, you can allow the user to filter the data in columns on a sheet and display only the rows of data which meet criteria from a drop-down list or change the appearance of rows based on that filtering.

For more information, refer to **Managing Filtering of Rows of User Data**.

## Formula Text Box (Formula Bar)

You can add a formula text box for editing formulas. The formula text box is similar to the formula editor available to the developer and has the appearance of a text box. The formula bar provides a list of calculation functions and provides a visual method of selecting cell ranges for the formula.

For more information, refer to **Using the Additional Spread Controls**.

## Formula Provider Control

You can add a formula provider control to the form that will provide formulas for other controls on the form.

For more information, refer to **Using the Additional Spread Controls**.

## Functions and Formulas

You can use built-in functions and operators to develop formulas and perform calculations. Add calculations quickly to your applications by using any of over 300 pre-defined functions or add your own custom functions. Choose from any of these types of functions:

- Database
- Date and Time
- Engineering
- Financial
- Information
- Logic
- Lookup and Reference
- Math and Trigonometry
- Statistics
- Text
- Volatile

Spread also offers a floating formula bar that you can provide to your end users to allow them to pick functions.

You can also create custom formulas or custom names to use in formulas. Refer to **Creating and Using a Custom Function** and **Creating and Using a Custom Name** for more information.

For more information on entering formulas using the Spread Designer, refer to the **Spread Designer Guide (on-line documentation)**.

For more information on formulas in general, refer to **Managing Formulas in Cells** and to the Formula Reference.

## Gradients for Button Cells, Headers, and More

You can set gradients for button cells and headers.

For more information see, **Adding a Gradient to Header Cells**.

For more information see, **Setting a Button Cell**.

## Grouping Rows in the Display

You can set up the spreadsheet to give users the ability to group rows of data based on a particular column name by dragging the column header to the grouping bar. This is useful for displaying large amounts of data in organized groups and organizing rows based on the category of a column. This is sometimes referred to as Outlook-style grouping.

For more information about outlines see, **Managing Grouping of Rows of User Data**.

## Grouping Rows or Columns in an Outline

You can set up the spreadsheet to give users the ability to form a range of expandable and collapsible rows or columns. This is useful for handling rows or columns of data that do not need to be visible all the time but are related to adjacent rows or columns. This is sometimes referred to as range grouping or Excel-like grouping or outlines.

For more information about grouping see, **Managing Outlines (Range Groups) of Rows and Columns**.

## Headers with Multiple Columns and Rows

You can have multiple column headers and row headers. You can also span header cells. Use headers with multiple columns or rows to organize your column and row information.

For more information about headers, refer to **Creating a Header with Multiple Rows or Columns**.

## Hierarchical Display

You can create a sheet within a row to display relational data hierarchically, with parent rows and child views of related data.

For more information about hierarchical display of data, refer to **Working with Hierarchical Data Display**.

## HitTest for Locating the Cursor

You can use the **HitTest** method for finding the location of the cursor (pointer) on the spreadsheet component to help with development of applications where accessibility issues are concerned.

For more information about the **HitTest** method of the spreadsheet component, refer to **Locating the Pointer Using HitTest**.

## Indicators and Icons in the Interface

You can create custom row filtering indicators and custom sorting indicators to display in the column header. You can also create custom images for the hierarchy display icons for expanding and collapsing the hierarchy.

You can also make a marquee (animated) focus indicator as well as other enhanced focus indicators.

For more information, refer to **Customizing the User Interface Images**.

## Ink Notation Support

You can use the ink notation for writing or drawing on Spread with your Tablet PC. With a simple method, you can turn on support for this inking feature.

For more details about ink notation and support for the Tablet PC, refer to **Allowing the User to Draw with a Tablet PC**.

## Keyboard Action Mapping

Spread Windows Forms provides multiple ways to customize navigation within the component, including keyboard navigation and action keys. With input maps and action maps, you can determine actions in the Spread component that occur when the user presses keys. A default mapping of keys and related actions is provided; however, you can customize these maps that define the mapping of keys to actions. For example, you can map the Enter key so that the component moves the focus to the cell below the current cell.

You can load an Excel compatibility input action map or other map file. You can also save an input action map to a file.

For more information about navigation keys, input maps, and action maps, refer to the descriptions in **Managing Keyboard Interaction**.

## Multiple Sheets

Spread Windows Forms supports multiple sheets in a single component each uniquely named. Use multiple sheets to categorize your information, similar to using worksheets in Microsoft Excel.

Sheets can have many rows and columns as well. You can define styles for sheets and apply those styles across multiple sheets.

For more information about sheets, refer to **Customizing the Sheet Appearance** and **Customizing Sheet Interaction**.

## Name Box Control

The name box control can be used to create or display custom names at run time.

For more information, refer to **Setting up the Name Box**.

## Notes for Cells

Spread Windows Forms allows cells to have notes attached to provide additional information to users. The cell note has the following functionality:

- automatically sizes cell notes based on contents
- prints cell notes
- provides customizable locations for the cell notes
- lets you customize the note indicator color
- provides sticky notes that stay where they are placed

For more information, refer to **Adding a Note to a Cell**.

## Panes or Viewports

You can allow more than one pane or viewport in the spreadsheet to allow you to view data from different parts of the spreadsheet in one display. Display data in multiple viewports and allow your user to customize their own viewport view by providing split boxes along with scroll bars.

For more information about scrollable viewports and split bars, refer to **Customizing Viewports**.

## Printing and PDF

You can customize the printing of your spreadsheets with many different printing options. For example, you can have colors and images in headers and footers. For more information, see **Customizing the Appearance of the Printing**.

You can print parts of the sheet as needed. You can also print a file to a Portable Document Format (PDF) file. For more information, see **Specifying What to Print**.

There are many events related to printing including the **PrintPreviewShowing ('PrintPreviewShowing Event' in the on-line documentation)** event. For more information, refer to **Providing a Preview of the Printing**. You can also set your own PrintPreviewDialog with the **SetPrintPreview ('SetPrintPreview Method' in the on-line documentation)** method.

## Quick Start Wizard

You can use the wizard to quickly set up data binding and other areas of the control.

For more information, refer to **Understanding the Spread Wizard**.

## Right-To-Left Layouts

You can handle support for right-to-left layouts in the Spread component.

For more information, refer to **Handling Right-to-Left Layouts**.

## Row Preview

You can add a preview row that contains extra information about a row. The preview row is displayed below the row it provides information for. You can specify colors and other formatting for the preview row as well.

For more information, refer to **Setting up Preview Rows**.

## Searching and Search Dialog

You can perform searches in code or provide a search dialog with various options to your user.

For more information, refer to **Customizing User Searching of Data**.

## Shapes, Drawings, and Annotations (Freehand Drawing)

You can draw shapes and customize drawing objects on top of a sheet to highlight parts of the sheet or point the user through the use of a form.

You can allow the user to draw in freehand on the sheet with annotations. The drawing is done on the same layer as shapes. This ability is called annotation mode.

For more information about shapes, refer to the **Designing Shapes (on-line documentation)** and **Customizing Drawing**.

## Sorting Rows or Columns

You can programmatically sort rows or columns or a range of cells. You can specify the order of sorting and the comparison method for sorting. You can allow your users to sort rows automatically simply by clicking on the column header. Use methods at the sheet level to perform these kinds of sorting:

- automatically sorting by column
- manually sorting columns or rows
- sorting data in a range of cells

For more information about sorting, refer to **Managing Sorting of Rows of User Data**.

## Spannable Cells

You can span cells. Create cell spans to join cells together, allowing one cell to span across multiple cells to include, for example, your company logo. You can span data cells or headers. You can also have Spread automatically merge cells that are the same value.

For more information about spanning cells, refer to **Creating a Span of Cells**. For more information, refer to **Allowing Cells to Merge Automatically**.

## Sparklines

You can add sparklines to a cell. A sparkline is a small graph that fits inside a cell and uses data from a range of cells.

For more information, refer to **Using Sparklines**.

## Spread Designer

You can use the Spread Designer to design your component and to create a prototype quickly. Use the Spread Designer to reduce development time by allowing you to customize the look and feel of the Spread component at design time using an intuitive, easy-to-use interface.

You can also show the Spread Designer in your application at run time.

For more information, refer to the **Spread Designer Guide (on-line documentation)**.

## Skins and Styles for Customized Appearance

Easily and quickly configure the appearance of Spread using predefined skins or create and save your own custom skins that define many of the appearance settings for a sheet. Custom skins can be shared with everyone in your development team, allowing a consistent look of the component across applications. You can also create specific styles that contain many of the appearance settings for individual cells. Named styles help you quickly customize the appearance of a cell or range of cells.

For information on managing skins (that apply to styles and renderers of the component), refer to **Applying a Skin to the Component** and **Creating a Custom Skin for a Component**.

For information about skins for sheets (using the older way of setting styles for individual sheets), refer to **Applying a**

**Skin to a Sheet** and **Creating a Custom Skin for a Sheet**.

For information on saving the skin to a file, refer to **Saving and Loading a Skin**.

For more information on styles for individual cells, refer to **Creating and Applying a Style for Cells**.

For more information on Visual Styles (and XP Themes), refer to **Using XP Themes with the Component**.

## Tables

You can add tables to the sheet. You can also sort or filter the table data.

For more information, refer to **Creating Tables**.

## Tab Strip and Sheet Name Tabs

You can customize the painting of various parts of the interface including the sheet name tabs and the tab strip that contains them.

For more information about the tab strip, refer to **Customizing the Sheet Name Tabs of the Component** and **Customizing Painting of Parts of the Component**.

## 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, How to Draw Text with GDI.

For more information in the Microsoft .NET Framework documentation, refer to Microsoft .NET TextRenderer Class.

## Title and SubTitle

You can add a specially formatted area at the top of the spreadsheet that includes a title and subtitle.

For more information, refer to **Adding a Title and Subtitle to a Sheet**.

## Touch Support

Spread supports touch gestures in many areas of the control.

For more information, refer to **Using Touch Support with the Component**.

## Undo and Redo Actions

You can allow the end user to undo and redo various types of user actions. This includes cell editing, copying, and pasting from the Clipboard, and other actions.

For more information, refer to **Customizing Undo and Redo Actions**.

## Visual Styles for XP Themes

You can assign visual styles to the Spread component to achieve the look and feel of XP themes.

For more information, refer to **Using XP Themes with the Component**.

## Namespaces Overview

In Spread Windows Forms, namespaces are organized to contain objects according to how they are used in the component and the features they provide. The objects in the Spread Windows Forms component fall into three categories:

- objects that represent parts of the spreadsheet, like column, rows, and cells
- objects that represent cell types and the formatting of data in those cells
- objects that are conceptual representations of underlying aspects of the spreadsheet

For each of these there is a specific namespace. The namespaces are organized as follows:

| Namespace | Description |
|---|---|
| **FarPoint.Win.Spread ('FarPoint.Win.Spread Namespace' in the on-line documentation)** | Provides the base classes, interfaces, enumerations, and delegates for the parts of the spreadsheet in Spread. |
| **FarPoint.Win.Spread.CellType ('FarPoint.Win.Spread.CellType Namespace' in the on-line documentation)** | Provides the base classes, interfaces, and enumerations for the cell types. |
| **FarPoint.Win.Spread.DrawingSpace ('FarPoint.Win.Spread.DrawingSpace Namespace' in the on-line documentation)** | Provides the base classes, interfaces, and enumerations for the various shapes and objects that can be drawn in the drawing space. |
| **FarPoint.Win.Spread.Model ('FarPoint.Win.Spread.Model Namespace' in the on-line documentation)** | Provides the base classes, interfaces, and enumerations for the models in Spread. |

The spreadsheet and cell type objects call the model objects. If you are new to working with Spread, or are new to developing in an object-oriented environment, you might want to use the spreadsheet and cell type objects at first, as you become familiar with features of Spread. However, intensive use of these objects can degrade your application's performance.

If you are an experienced programmer, you might want to use the model objects directly, instead of accessing them through the shortcut objects. If you want to extend Spread Windows Forms, you must use the model objects to do so.

The spreadsheet objects and event arguments are in classes in the main FarPoint Spread namespace. For a discussion of how to work with these objects, refer to **Shortcut Objects**.

The cell type objects provide ways for you to set up different types of cells to help the user or limit the types of input. They are separated into their own namespace, the CellType namespace mainly to allow you to see all the cell type information in one place, separate from the spreadsheet objects. For details on the different cell types, refer to **Customizing Interaction with Cell Types**.

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 component. These are found in the Model namespace. In Spread, you can use the default models or extend them through inheritance. Refer to **Underlying Models** for more information on models.

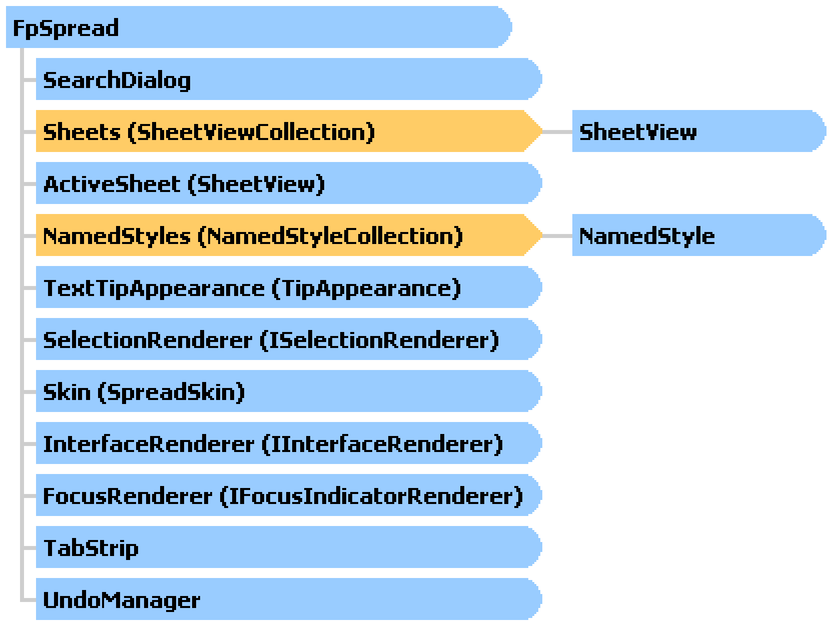
## Concepts Overview

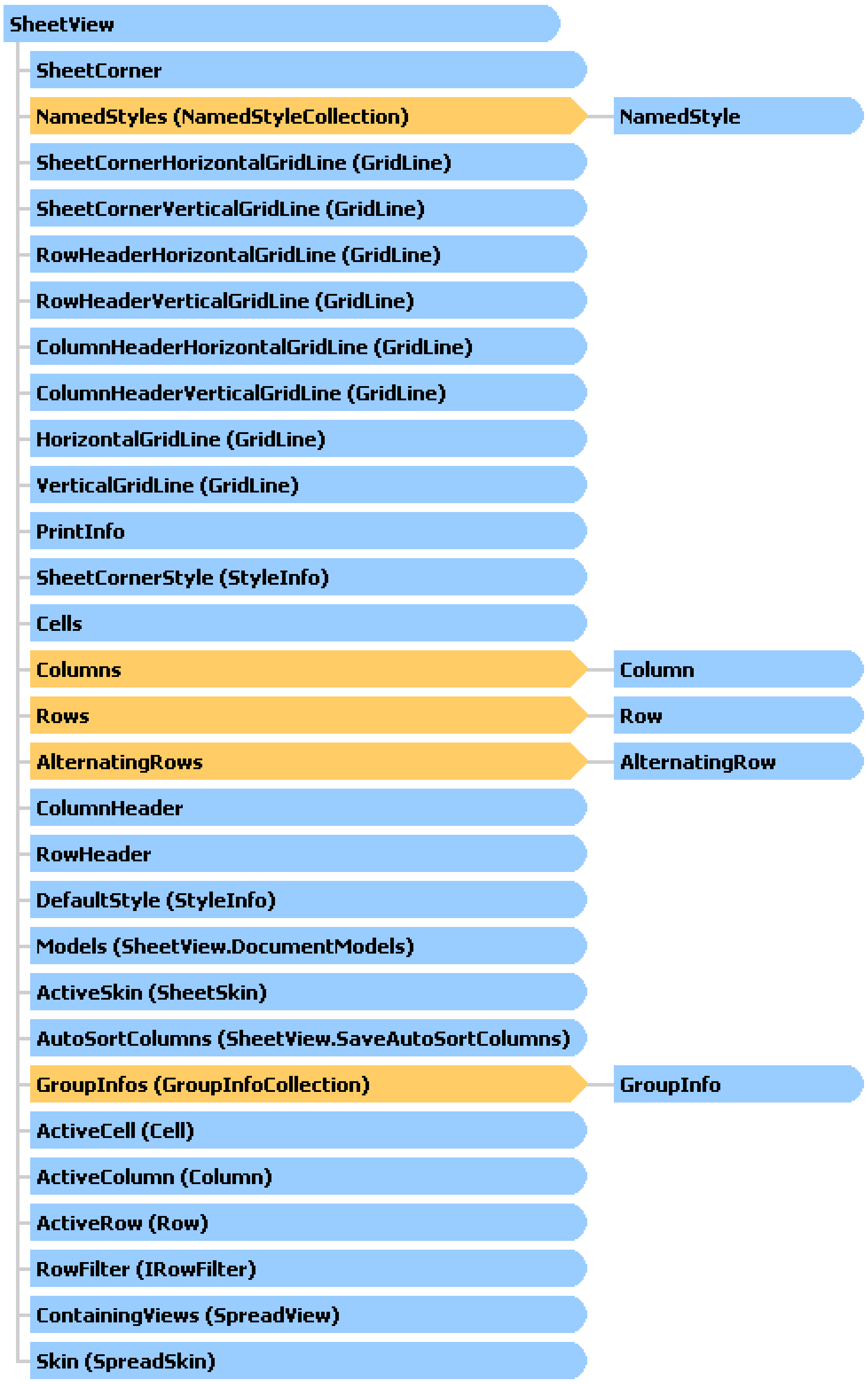
Users who are new to using Spread products might want to review the following topics regarding object orientation.

- **Shortcut Objects**
- **Object Parentage**
- **Formatted versus Unformatted Data**
- **Cell Types**
- **Underlying Models**

## 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.


FpSpread
SearchDialog
Sheets (SheetViewCollection)
SheetView
ActiveSheet (SheetView)
NamedStyles (NamedStyleCollection)
NamedStyle
TextTipAppearance (TipAppearance)
SelectionRenderer (ISelectionRenderer)
Skin (SpreadSkin)
InterfaceRenderer (IInterfaceRenderer)
FocusRenderer (IFocusIndicatorRenderer)
TabStrip
UndoManager


SheetView
SheetCorner
NamedStyles (NamedStyleCollection)
NamedStyle
SheetCornerHorizontalGridLine (GridLine)
SheetCornerVerticalGridLine (GridLine)
RowHeaderHorizontalGridLine (GridLine)
RowHeaderVerticalGridLine (GridLine)
ColumnHeaderHorizontalGridLine (GridLine)
ColumnHeaderVerticalGridLine (GridLine)
HorizontalGridLine (GridLine)
VerticalGridLine (GridLine)
PrintInfo
SheetCornerStyle (StyleInfo)
Cells
Columns
Column
Rows
Row
AlternatingRows
AlternatingRow
ColumnHeader
RowHeader
DefaultStyle (StyleInfo)
Models (SheetView.DocumentModels)
ActiveSkin (SheetSkin)
AutoSortColumns (SheetView.SaveAutoSortColumns)
GroupInfos (GroupInfoCollection)
GroupInfo
ActiveCell (Cell)
ActiveColumn (Column)
ActiveRow (Row)
RowFilter (IRowFilter)
ContainingViews (SpreadView)
Skin (SpreadSkin)

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.

## 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

For more information on the setting of properties of an object and how to use the Parent property of an object, refer to **Customizing the Appearance of a Cell** and **Customizing Interaction in Cells**. For information on cell types, which is set in a different way than inheriting from a parent, refer to **Customizing Interaction with Cell Types**.

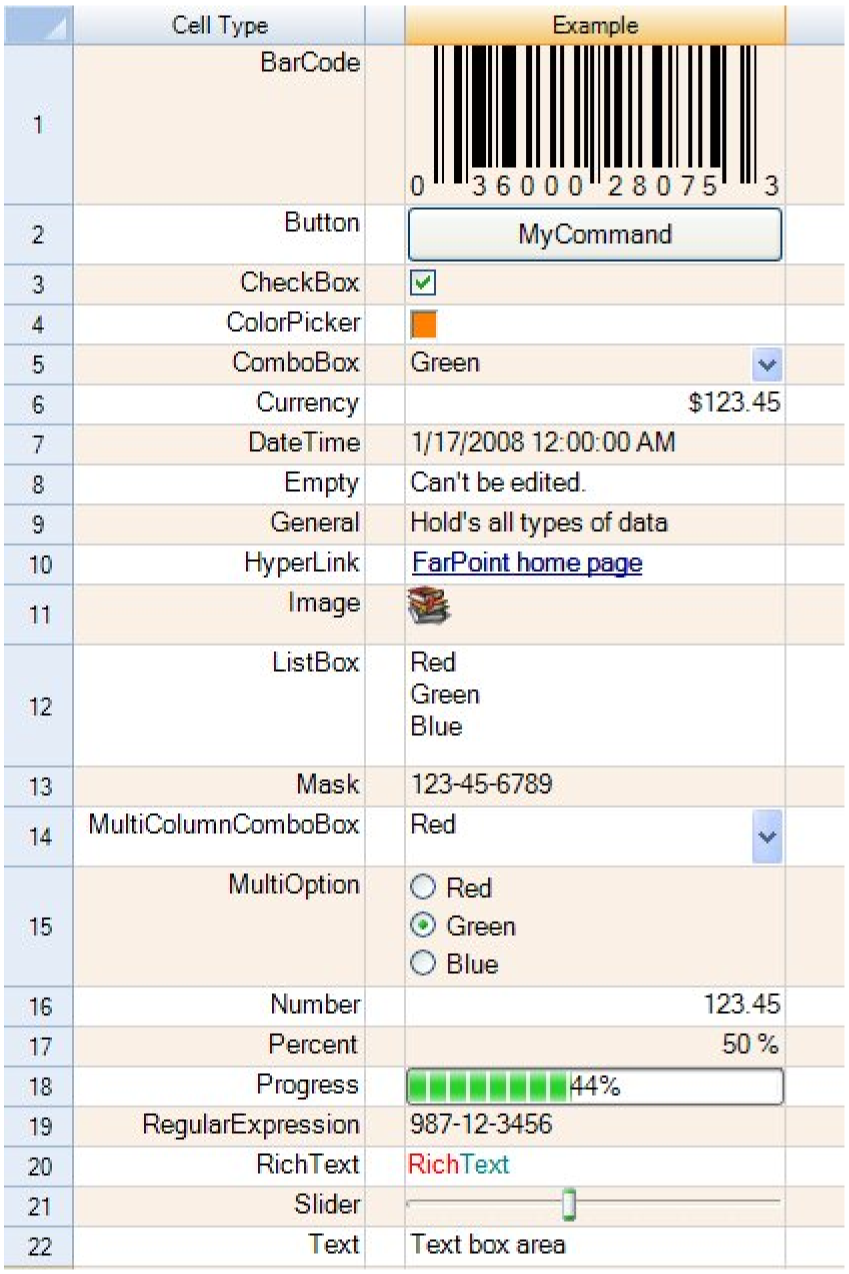
## Formatted versus Unformatted Data

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 | 0 36000 28075 3 |
| 2 | Button | MyCommand |
| 3 | 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<br>Green<br>Blue |
| 13 | Mask | 123-45-6789 |
| 14 | MultiColumnComboBox | Red |
| 15 | MultiOption | ○ Red<br>● Green<br>○ 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 |

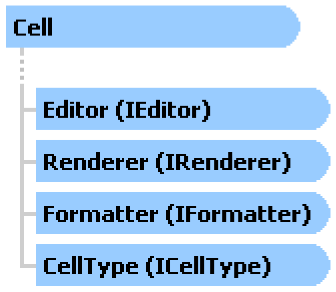
In general the cell types are grouped into two broad categories: editable cell types (such as text, currency, and number) and graphical or control cell types (such as button, progress, and slider). For a list of cell types and details about using them, refer to **Customizing Interaction with Cell Types**.

**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 **Customizing Interaction with Cell Types**.

## 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**.

## Understanding the 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.

The tasks that relate to setting the appearance of objects in the Spread component include:

- **Working with Sheets**
- **Working with the Rows and Columns**
- **Working with Headers**
- **Working with Cells**

For information on customizing the appearance using the Spread Designer, refer to the **Spread Designer Guide (on-line documentation)**.

## Working with 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**
- **Adding a Sheet**
- **Copying and Inserting a Sheet**
- **Moving a Sheet**
- **Removing a Sheet**
- **Showing or Hiding a Sheet**

When you work with sheets, you can work with the objects using the shortcut objects in code, (**SheetView ('SheetView Class' in the on-line documentation)** and **SheetViewCollection ('SheetViewCollection 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. 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 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.

## Working with the Active Sheet

The active sheet is the sheet that currently receives any user interaction. 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.

**Using a Shortcut**

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

**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:

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

You can specify properties for an individual sheet or for several sheets at a time. Use the SheetView class or the ActiveSheet shortcut in code. For more information, refer to **Customizing the Individual Sheet Appearance**.

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 in the **SheetView ('SheetView Class' in the on-line documentation)** class to name the sheet programmatically.

To add a new sheet, refer to **Adding a Sheet**. To remove a sheet, refer to **Removing a Sheet**.

You can use code such as the following to copy a sheet to another Spread component.

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

Since the two sheets use the same SheetView object, any changes made in one sheet are automatically reflected in the other.

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

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

The FpSpread.**Sheets ('Sheets Property' in the on-line documentation)** property can be used when you wish to use an integer for the sheet value. This method has a **SheetViewCollection ('SheetViewCollection Class' in the on-line documentation)** object as its value and takes an integer as the indexer instead of a string for a sheet name.

For information about the display of the sheet names in the sheet name buttons, refer to **Customizing the Sheet Name Tabs of the Component**.

## 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.

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 of the Component**.

To add a sheet to the component, 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. 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.

**Using a Shortcut**

1. Create a new **SheetView ('SheetView Class' in the on-line documentation)** object.
2. If you want to do so, set properties for the sheet, such as its name.
3. Call the Sheets shortcut object **Add ('Add Method' in the on-line documentation)** method to add the new sheet to the **SheetViewCollection ('SheetViewCollection Class' in the on-line documentation)** for the component.

**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);
```

**VB**

```
' 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)
```

## Copying and Inserting a Sheet

You can copy and insert a sheet to the same Spread component or another Spread component on the form. Spread does not provide a way to copy the sheet, but with the code given below you can easily create your own CopySheet method. To copy a sheet and insert it in the component, simply create a new method, called CopySheet, as shown here and then use the **Add ('Add Method' in the on-line documentation)** or **Insert ('Insert Method' in the on-line documentation)** methods in the **SheetViewCollection ('SheetViewCollection Class' in the on-line documentation)** class.

The CopySheet method also copies all shapes on that sheet.

Copying a sheet using the CopySheet method also copies the NamedStyleCollection in the sheet, and creates separate copies of any NamedStyle 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 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 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.

The **SpreadActions ('SpreadActions Class' in the on-line documentation)** class has options for the clipboard copy, cut, and paste of a sheet.

**Using Code**

1. Create a new CopySheet method to be used for copying sheets.
2. Handle the named styles for that sheet as described above.
3. Call the Sheets shortcut object **Add ('Add Method' in the on-line documentation)** method to add the new sheet or the Insert method to insert the sheet to the **SheetViewCollection ('SheetViewCollection Class' in the on-line documentation)** for the component.

**Example**

This is the code for the CopySheet method.

**C#**

```
private void Form1_Load(object sender, EventArgs e)
{
  fpSpread1.Sheets.Count = 3;
}

private void button1_Click(object sender, EventArgs e)
```

```
{
  FarPoint.Win.Spread.SheetView s = new FarPoint.Win.Spread.SheetView();
  s.Cells[0, 0].Text = "test";
  FarPoint.Win.Spread.DrawingSpace.FourWayArrowShape sh = new
FarPoint.Win.Spread.DrawingSpace.FourWayArrowShape();
  sh.Name = "Arrow";
  s.AddShape(sh);
  fpSpread1.Sheets.Add(CopySheet(s));
}

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

**VB**

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
  FpSpread1.Sheets.Count = 3
End Sub

Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
  Dim s As New FarPoint.Win.Spread.SheetView()
  s.Cells(0, 0).Text = "test"
  Dim sh As New FarPoint.Win.Spread.DrawingSpace.FourWayArrowShape()
  sh.Name = "Arrow"
  s.AddShape(sh)
  FpSpread1.Sheets.Add(CopySheet(s))
End Sub

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
```

## 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.

**Using a Shortcut**

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)
```

## 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.

**Using a Shortcut**

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. For more information on removing the sheet completely, refer to **Removing a Sheet**.

For programming details, refer to the **Visible ('Visible Property' in the on-line documentation)** property of the **SheetView ('SheetView 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 to hide.
5. Select the **Visible** property in the property list, and then select false.
6. Click **OK** to close the editor.

**Using a Shortcut**

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 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.

## Working with the 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**

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. For more information, refer to **Allowing User Interaction with 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
```

### Using Code

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)** object.

### 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 methods in the **SheetView ('SheetView Class' in the on-line documentation)** class or the methods in the **DefaultSheetDataModel ('DefaultSheetDataModel Class' in the on-line documentation)** class.

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 column 6.

**C#**

```
fpSpread1.Sheets[0].AddColumns(6,2);
```

**VB**

```
FpSpread1.Sheets(0).AddColumns(6,2)
```

**Using Code**

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 column 6.

**C#**

```
FarPoint.Win.Spread.SheetView Sheet0;
Sheet0 = fpSpread1.Sheets[0];
Sheet0.AddColumns(6,2);
```

**VB**

```
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 methods in the **SheetView ('SheetView Class' in the on-line documentation)** class or the methods in the **DefaultSheetDataModel ('DefaultSheetDataModel Class' in the on-line documentation)** class.

For more details refer to the SheetView.**RemoveRows ('RemoveRows Method' in the on-line documentation)** method or SheetView.**RemoveColumns ('RemoveColumns Method' in the on-line documentation)** method.

If you simply want to hide the row or column from the end user, but not remove it from the sheet, 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 column 6.

**C#**

```
fpSpread1.Sheets[0].RemoveColumns(6,2);
```

**VB**

```
FpSpread1.Sheets(0).RemoveColumns(6,2)
```

**Using Code**

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 column 6.

**C#**

```
FarPoint.Win.Spread.SheetView Sheet0;
Sheet0 = fpSpread1.Sheets[0];
Sheet0.RemoveColumns(6,2);
```

**VB**

```
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

By default, rows and columns are visible on a sheet. You can hide rows or columns in a sheet so that they are not displayed. You can also hide row headers and column headers. You hide a row or column by setting the Row.**Visible ('Visible Property' in the on-line documentation)** property or Column.**Visible ('Visible Property' in the on-line documentation)** property to false. The hidden rows or columns are not displayed at run time, but during design time, they are still visible.

When you hide a row or column, the size of the row or column is remembered by the Spread component. If you redisplay the row or column, it is displayed at the size it was before it was hidden. For example, if the width of a column is 100 pixels and you hide the column, when you redisplay it, the width is 100.

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 methods of the sheet. For example, to find out if a column appears, use the **GetViewportLeftColumn ('GetViewportLeftColumn Method' in the on-line documentation)** and **GetViewportRightColumn ('GetViewportRightColumn Method' in the on-line documentation)** methods to determine the left-most and right-most columns in the viewport and then determine if the column falls within those indexes.

For information on hiding rows or columns in headers, refer to **Showing or Hiding Headers**.

For more information on hiding rows or columns using the Spread Designer, refer to the **Spread Designer Guide (on-line documentation)**.

**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 or a row or column to select it and then select an option from the drop-down combo list for the **Visible** property.

**Using Code**

Use the **SetColumnVisible ('SetColumnVisible Method' in the on-line documentation)** or **SetRowVisible ('SetRowVisible Method' in the on-line documentation)** method for the sheet.

**Example**

**C#**

```
fpSpread1.Sheets[0].SetColumnVisible(0,true);
fpSpread1.Sheets[0].SetRowVisible(0,false);
//Another option is to use the Visible property.
fpSpread1.Sheets[0].Columns[1].Visible = false;
fpSpread1.Sheets[0].Rows[1].Visible = true;
```

**VB**

```
fpSpread1.Sheets(0).SetColumnVisible(0, False)
fpSpread1.Sheets(0).SetRowVisible(0, True)
'Another option is to use the Visible property.
FpSpread1.Sheets(0).Columns(1).Visible = True
FpSpread1.Sheets(0).Rows(1).Visible = False
```

**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.

## Working with 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**
- **Creating a Header with Multiple Rows or Columns**
- **Showing or Hiding Headers**

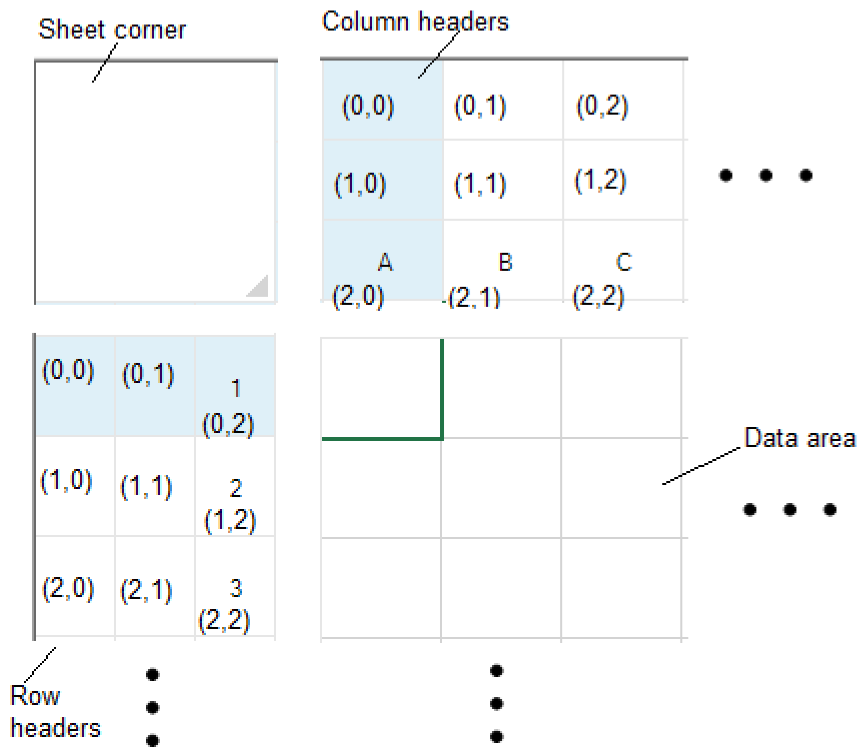
You can also customize header appearance in other ways.

- 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.

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

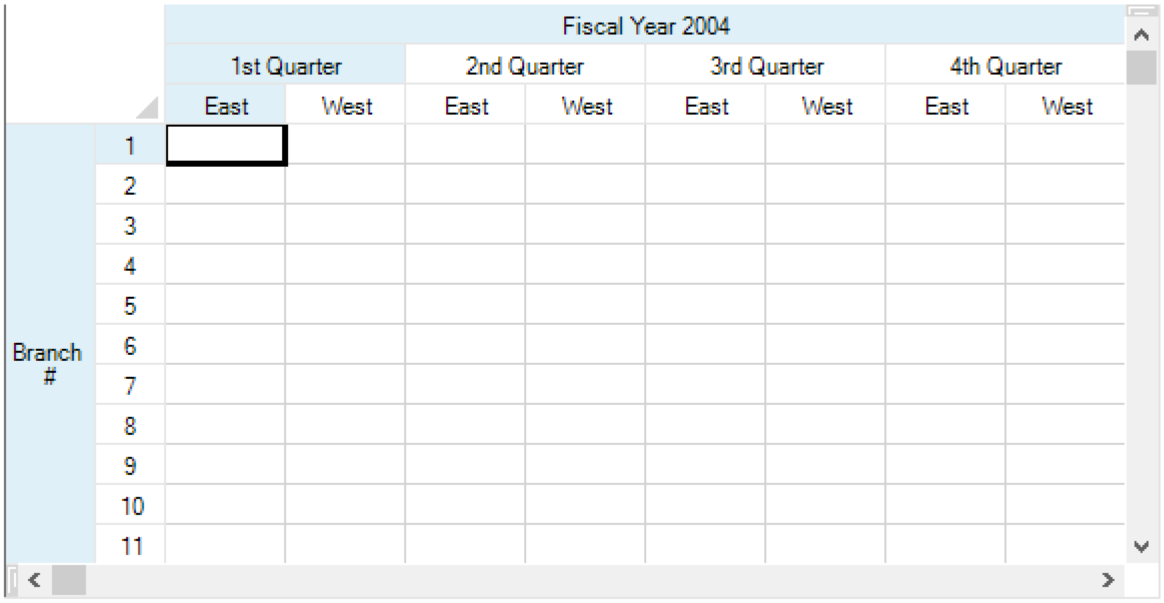
The following figure shows the sheet corner, headers, and cells and illustrates the cell coordinates in headers with multiple rows and columns. The coordinates are shown in parentheses.



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.

## 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.



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 or 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 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 #";
```

**VB**

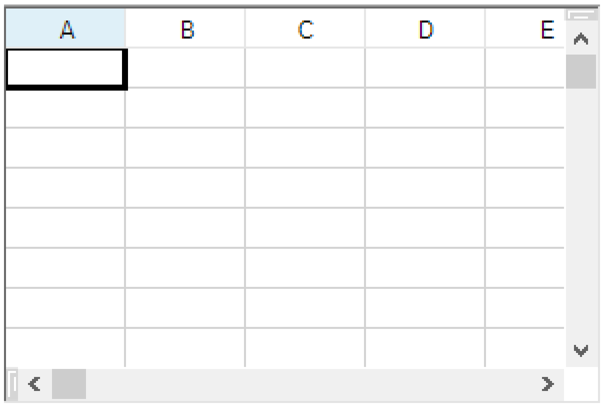
```
' Set the number or rows and columns in the headers.
FpSpread1.Sheets(0).ColumnHeaderRowCount = 3
FpSpread1.Sheets(0).RowHeaderColumnCount = 2
' Span the header cells as needed.
FpSpread1.Sheets(0).AddColumnHeaderSpanCell(1, 0, 1, 2)
FpSpread1.Sheets(0).AddColumnHeaderSpanCell(1, 2, 1, 2)
FpSpread1.Sheets(0).AddColumnHeaderSpanCell(1, 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 or 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.

## 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:

- **RowHeader ('RowHeader Class' in the on-line documentation)** class **Visible ('Visible Property' in the on-line documentation)** property
- **ColumnHeader ('ColumnHeader Class' in the on-line documentation)** class **Visible ('Visible Property' in the on-line documentation)** property
- **SheetView ('SheetView Class' in the on-line documentation)** class **RowHeaderVisible ('RowHeaderVisible Property' in the on-line documentation)** property
- **SheetView ('SheetView Class' in the on-line documentation)** class **ColumnHeaderVisible ('ColumnHeaderVisible Property' in the on-line documentation)** property

Alternatively, you can customize the headers by providing custom text or headers with multiple columns or rows, as explained in **Customizing the Default Header Labels** and **Creating a Header with Multiple Rows or 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. 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 Code**

1. Create a new **SheetView ('SheetView Class' in the on-line documentation)** object.
2. Set the **SheetView ('SheetView Class' in the on-line documentation)** object **ColumnHeaderVisible ('ColumnHeaderVisible Property' in the on-line documentation)** or **RowHeaderVisible ('RowHeaderVisible Property' in the on-line documentation)** property to false.
3. Set the sheet equal to the **SheetView ('SheetView Class' in the on-line documentation)** object you just created.

**Example**

This example code sets the first sheet to not display column headers.

**C#**

```
// Create a new sheet.
FarPoint.Win.Spread.SheetView newsheet = new FarPoint.Win.Spread.SheetView();
newsheet.ColumnHeaderVisible = false;
// Set first sheet equal to SheetView object.
fpSpread1.Sheets[0] = newsheet;
```

**VB**

```
' Create a new sheet.
Dim newsheet As New FarPoint.Win.Spread.SheetView()
newsheet.ColumnHeaderVisible = False
' Set first sheet equal to SheetView object.
FpSpread1.Sheets(0) = newsheet
```

**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.

## Working with 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**
- **Creating a Range of Cells**

**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 **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 information on customizing the appearance of cells using the Spread Designer, refer to 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 **SelectionStyle ('SelectionStyle 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 **SelectionStyle** 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)
```

## 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.

## 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.

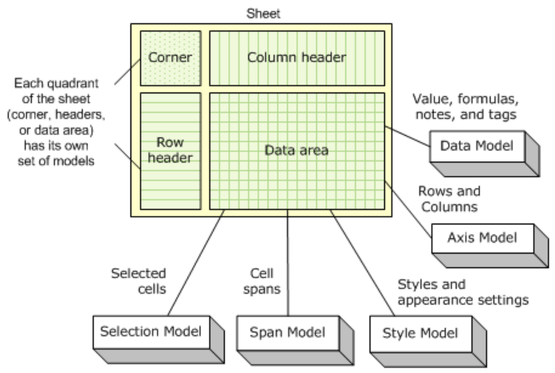
Consult the following topics for lists of the Spread models and more information about the model classes.

These topics can help you customize the component using models:

- **Understanding the Types of Sheet Models**
- **Understanding the Sheet Model Classes and Interfaces**
- **Finding More Details on the Sheet Models**
- **Creating a Custom Sheet Model**
- **Understanding the Optional Interfaces**

### Understanding the Types of Sheet Models

The Spread component models are illustrated conceptually in the following diagram:


Sheet
Corner
Column header
Each quadrant of the sheet (corner, headers, or data area) has its own set of models
Row header
Data area
Value, formulas, notes, and tags
Data Model
Rows and Columns
Axis Model
Selected cells
Cell spans
Styles and appearance settings
Selection Model
Span Model
Style Model

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 implement 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 | Classes and Interface | Description |
|---|---|---|

## Model

| | | |
|---|---|---|
| Axis model | **BaseSheetAxisModel ('BaseSheetAxisModel Class' in the on-line documentation)**<br>**DefaultSheetAxisModel ('DefaultSheetAxisModel Class' in the on-line documentation)**<br>**ISheetAxisModel ('ISheetAxisModel Interface' in the on-line documentation)** | 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 | **BaseSheetDataModel ('BaseSheetDataModel Class' in the on-line documentation)**<br>**DefaultSheetDataModel ('DefaultSheetDataModel Class' in the on-line documentation)**<br>**ISheetDataModel ('ISheetDataModel Interface' in the on-line documentation)** | Basis for the manipulation of data in the cells in the sheet. For more information, see **Understanding the Data Model**. |
| Selection model | **BaseSheetSelectionModel ('BaseSheetSelectionModel Class' in the on-line documentation)**<br>**DefaultSheetSelectionModel ('DefaultSheetSelectionModel Class' in the on-line documentation)**<br>**ISheetSelectionModel ('ISheetSelectionModel Interface' in the on-line documentation)** | Basis for the behavior of and interaction of selected cells in the sheet. For more information, see **Understanding the Selection Model**. |
| Span model | **BaseSheetSpanModel ('BaseSheetSpanModel Class' in the on-line documentation)**<br>**DefaultSheetSpanModel ('DefaultSheetSpanModel Class' in the on-line documentation)**<br>**ISheetSpanModel ('ISheetSpanModel Interface' in the on-line documentation)** | Basis for how cells in the sheet are spanned. For more information, see **Understanding the Span Model**. |
| Style model | **BaseSheetStyleModel ('BaseSheetStyleModel Class' in the on-line documentation)**<br>**DefaultSheetStyleModel ('DefaultSheetStyleModel Class' in the on-line documentation)**<br>**ISheetStyleModel ('ISheetStyleModel Interface' in the on-line documentation)** | Basis for the appearance of the cells in the sheet. For more information, see **Understanding the Style Model**. |

The sheet (SheetView object) is a composite of the five underlying models (Axis, Data, Selection, Span, and Style). 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). 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. The Selection model handles any cell range selections that are made, and Span models handles any spanned cells. The Style model handles the appearance settings for the cells (for example, the background color, the font, and the cell type).

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**.

## Finding More Details on the Sheet Models

The following topics list additional information about the models of a sheet.

- **Understanding the Data Model**
- **Understanding the Axis Model**
- **Understanding the Selection Model**
- **Understanding the Span Model**
- **Understanding the Style Model**

For general information about the types of sheet models, refer to **Understanding the Sheet Model Classes and Interfaces**.

## 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
- Balance of Speed and Performance

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 that is created when the sheet is created.

The default data model, **DefaultSheetDataModel ('DefaultSheetDataModel Class' in the on-line documentation)**, 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:

```
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.

To see the difference between the default data model and the objects on the sheet, review the following code snippets. These code snippets bind the sheet to a data source called MyData.

```
FpSpread1.Sheets(0).DataSource = MyData.Tables(0)
```

and

```
Dim model As FarPoint.Win.Spread.Model.DefaultSheetDataModel = New FarPoint.Win.Spread.Model.DefaultSheetDataModel(MyData, strTable)
FpSpread1.Sheets(0).Models.Data = model
```

In the first code snippet, the existing data model is used and resized to the data source; in the second snippet, the data model is replaced with a new one and the old one discarded. The outcome is the same in both examples, but the first example results in the old data model getting garbage collected. Generally you might not want to replace the data model unless you are creating your own data model class. There is generally no need to replace the data model with another DefaultSheetDataModel since there is already one there to use.

**Balance of Speed and Performance**

If you derive from **DefaultSheetDataModel ('DefaultSheetDataModel Class' in the on-line documentation)** and use that implementation of **GetValue ('GetValue Method' in the on-line documentation)** and **SetValue ('SetValue Method' in the on-line documentation)** to store the data, then it will use the Spread implementation of sparse arrays and matrices to balance the memory usage with the access speed. This implementation is designed to make it very fast to create a very large model (that is, 2 billion rows by 2 billion columns) and keep it reasonably fast to get and set values into it, until the number of values gets very large (in which case you will start to run out of memory anyway).

In cases where the model is very large and/or sparse (that is, more than two-thirds empty), access speed is slower (a binary search is required) and memory usage is lower. In cases where the model is not very large (less than 32K rows and/or columns) and not sparse (more than one-third full), then the access speed is faster (no binary search required) and memory usage is higher.

You can run some simple tests by creating a test project with Spread on a form, and setting the **ColumnCount ('ColumnCount Property' in the on-line documentation)** and **RowCount ('RowCount Property' in the on-line documentation)** for the sheet to very large numbers, and you should not see any delay at all because the memory allocated is based on the actual number of data items. If you start to fill the sheet with lots and lots of data, then you will notice delays after a while, especially when memory gets low and the system starts using the page file to swap virtual memory (it will take a very large quantity of data for that to happen though).

## 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, Columns
- Row, Rows
- AlternatingRow, AlternatingRows

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.

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 **Customizing Interaction with 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:

- **BackColor ('BackColor Property' in the on-line documentation)**
- **Border ('Border Property' in the on-line documentation)**
- **CellType ('CellType Property' in the on-line documentation)**
- **Editor ('Editor Property' in the on-line documentation)**
- **Font ('Font Property' in the on-line documentation)**
- **ForeColor ('ForeColor Property' in the on-line documentation)**
- **Formatter ('Formatter Property' in the on-line documentation)**
- **HorizontalAlignment ('HorizontalAlignment Property' in the on-line documentation)**
- **Locked ('Locked 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**

```
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"
```

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**.

All of the optional interfaces are implemented by **DefaultSheetDataModel ('DefaultSheetDataModel Class' in the on-line documentation)**; therefore, if you want any of the optional interfaces implemented in your data model, it might be easier to simply subclass **DefaultSheetDataModel ('DefaultSheetDataModel Class' in the on-line documentation)**.

**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 be consume a lot of time and memory.

Instead, you might want to create your own custom data model, as shown in the following example.

**Example**

**C#**

```
for (r = 0; r < 1000000; r++)
for ( c = 0; c < 10; c++)
spread.Sheets[0].Cells[r,c].Value = r + c;
```

**Example**

**C#**

```
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;
    }
}
```

## 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 | Customizations Allowed |
|---|---|
| **IArraySupport ('IArraySupport Interface' in the on-line documentation)** | Allows customization of support for getting and setting arrays of values in a range of cells |
| **IDataSourceSupport ('IDataSourceSupport Interface' in the on-line documentation)** | Allows customization of data binding on a sheet |
| **IChildModelSupport ('IChildModelSupport Interface' in the on-line documentation)** | Allows customization of hierarchical data models for hierarchies on a sheet; used in conjunction with **IDataSourceSupport ('IDataSourceSupport Interface' in the on-line documentation)** |
| **ICalculationSupport ('ICalculationSupport Interface' in the on-line documentation), ICustomFunctionSupport ('ICustomFunctionSupport Interface' in the on-line documentation), ICustomNameSupport ('ICustomNameSupport Interface' in the on-line documentation), IExpressionSupport ('IExpressionSupport Interface' in the on-line documentation), IExpressionSupport2 ('IExpressionSupport2 Interface' in the on-line documentation), IIterationSupport ('IIterationSupport Interface' in the on-line documentation)** | Allows customization of formulas on a sheet; **ICustomFunctionSupport ('ICustomFunctionSupport Interface' in the on-line documentation), ICustomNameSupport ('ICustomNameSupport Interface' in the on-line documentation)**, and **IIterationSupport ('IIterationSupport Interface' in the on-line documentation)** are not useful without **IExpressionSupport ('IExpressionSupport Interface' in the on-line documentation)** |

| | |
|---|---|
| **IDisjointSelections ('IDisjointSelections Interface' in the on-line documentation)**, **IQuerySelection ('IQuerySelection Interface' in the on-line documentation)** | Allows customization of an ordered array of cell ranges containing the selected cells with the 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 |
| **IMovable ('IMovable Interface' in the on-line documentation)**, **IRangeSupport ('IRangeSupport Interface' in the on-line documentation)** | Allows customization of moving, inserting, and deleting rows and columns support for a range of cells on a sheet; also covers clear, copy, move, and swap support |
| **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 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 Overall Component Appearance**
- **Customizing the Individual Sheet Appearance**
- **Customizing the Sheet Corner Appearance**

For information on customizing the interaction with parts of the Spread component, refer to **Customizing Sheet Interaction**.

For information on customizing the appearance using the Spread Designer, refer to the **Spread Designer Guide (on-line documentation)**.

## Customizing the Overall Component Appearance

You can set several aspects that determine the appearance of the overall Spread component. 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**
- **Customizing the Renderers**
- **Customizing the Dimensions of the Component**
- **Customizing the Outline of the Component**
- **Customizing the Display of the Pointer**
- **Customizing Painting of Parts of the Component**
- **Using XP Themes with the Component**
- **Handling Right-to-Left Layouts**

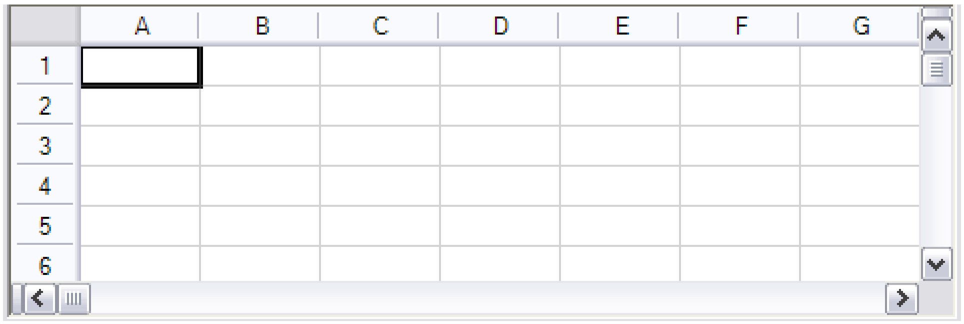
Other topics related to overall appearance include:

- **Setting the Background Colors for a Sheet**
- **Displaying Grid Lines on a Sheet**

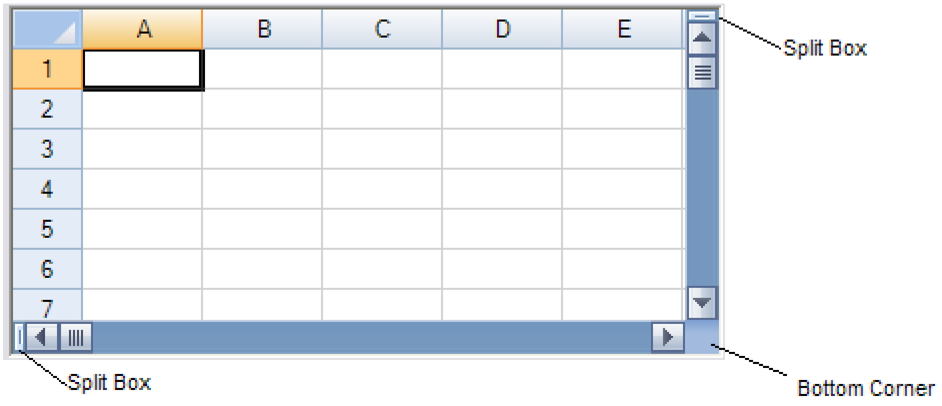
For information on other aspects of the overall component, refer to **Customizing Interaction in the Overall Component**.

## Setting the Component to the Original Appearance

You can set the appearance of the spreadsheet component to the original default look of version 3. This involves setting the renderers for the overall component, column header, row header, scroll bars, sheet corner, and focus indicator. You can set a skin along with the renderers for the scroll bars to get the look from version 2.5 (`FarPoint.Win.Spread.DefaultSkins.Default.Apply(FpSpread1)`).



The interface renderer for the component effects the following areas as well as the **GrayAreaBackColor ('GrayAreaBackColor Property' in the on-line documentation)** property of the sheet (area between the last column or row and the scroll bars).



To set the appearance to the default, use the following properties.

**Using Code**

To set the appearance to the look of version 3, set the renderers including the ColumnHeaderRenderer, the RowHeaderRenderer, and the CornerRenderer, as shown in the following example.

**Example**

This example shows how to set the appearance of the component to the original version 3 look.

**C#**

```
fpSpread1.InterfaceRenderer = null;
// set the column header renderer to the default
fpSpread1.ActiveSheet.ColumnHeader.DefaultStyle.Renderer = new
FarPoint.Win.Spread.CellType.ColumnHeaderRenderer();
// set the row header renderer to the default
fpSpread1.ActiveSheet.RowHeader.DefaultStyle.Renderer = new
FarPoint.Win.Spread.CellType.RowHeaderRenderer();
// set the sheet corner renderer to the default
fpSpread1.ActiveSheet.SheetCorner.DefaultStyle.Renderer = new
FarPoint.Win.Spread.CellType.CornerRenderer();
// set the scroll bar renderers to the default
fpSpread1.HorizontalScrollBar.Renderer = null;
fpSpread1.VerticalScrollBar.Renderer = null;
```

```
// The focus indicator can be set to the version 3 look as well
//fpSpread1.FocusRenderer = new FarPoint.Win.Spread.DefaultFocusIndicatorRenderer;
```

**VB**

```
FpSpread1.InterfaceRenderer = Nothing
' set the column header renderer to the default
FpSpread1.ActiveSheet.ColumnHeader.DefaultStyle.Renderer = New
FarPoint.Win.Spread.CellType.ColumnHeaderRenderer
' set the row header renderer to the default
FpSpread1.ActiveSheet.RowHeader.DefaultStyle.Renderer = New
FarPoint.Win.Spread.CellType.RowHeaderRenderer
' set the sheet corner renderer to the default
FpSpread1.ActiveSheet.SheetCorner.DefaultStyle.Renderer = New
FarPoint.Win.Spread.CellType.CornerRenderer
FpSpread1.HorizontalScrollBar.Renderer = Nothing
' set the scroll bar renderers to the default
FpSpread1.VerticalScrollBar.Renderer = Nothing
' The focus indicator can be set to the version 3 look as well
'FpSpread1.FocusRenderer = New FarPoint.Win.Spread.DefaultFocusIndicatorRenderer
```

## Applying a Skin to the Component

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.

| For more information and instructions about | See |
|---|---|
| Creating and applying your own skins | **Creating a Custom Skin for a Component** |
| Creating and applying your own cell-level styles | **Creating and Applying a Style for Cells** |
| Saving the skin to a file or stream | **Saving and Loading a Skin** |
| Customizing a skin in the Spread Designer | **SpreadSkin Editor (on-line documentation)** |
| Spread skins | **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 | See |
|---|---|
| Applying the built-in skins | **Applying a Skin to the Component** |
| Creating and applying your own cell-level styles | **Creating and Applying a Style for Cells** |
| Saving the skin to a file or stream | **Saving and Loading a Skin** |
| Underlying model for skins | **Understanding the Style Model** |
| Customizing a skin in the Spread Designer | **SpreadSkin Editor (on-line documentation)** |
| Spread skins | **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 int1 = new
FarPoint.Win.Spread.EnhancedInterfaceRenderer();
int1.ArrowColorDisabled = Color.Green;
int1.ArrowColorEnabled = Color.LightSeaGreen;
int1.ScrollBoxBackgroundColor = Color.Aqua;
int1.TabShape =
FarPoint.Win.Spread.EnhancedInterfaceRenderer.SheetTabShape.RoundedRectangle;
int1.TabStripButtonStyle =
FarPoint.Win.Spread.EnhancedInterfaceRenderer.ButtonStyles.Enhanced;
int1.TabStripButtonFlatStyle = FlatStyle.Popup;
int1.SheetTabBorderColor = Color.Aquamarine;
int1.SheetTabLowerActiveColor = Color.DarkSeaGreen;
int1.SheetTabLowerNormalColor = Color.DarkOliveGreen;
int1.SheetTabUpperActiveColor = Color.ForestGreen;
int1.SheetTabUpperNormalColor = Color.LightSeaGreen;
int1.SplitBarBackgroundColor = Color.Aquamarine;
int1.SplitBarDarkColor = Color.DarkGreen;
int1.SplitBarLightColor = Color.LightGreen;
int1.SplitBoxBackgroundColor = Color.Green;
int1.SplitBoxBorderColor = Color.LimeGreen;
int1.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",
int1, 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)
```

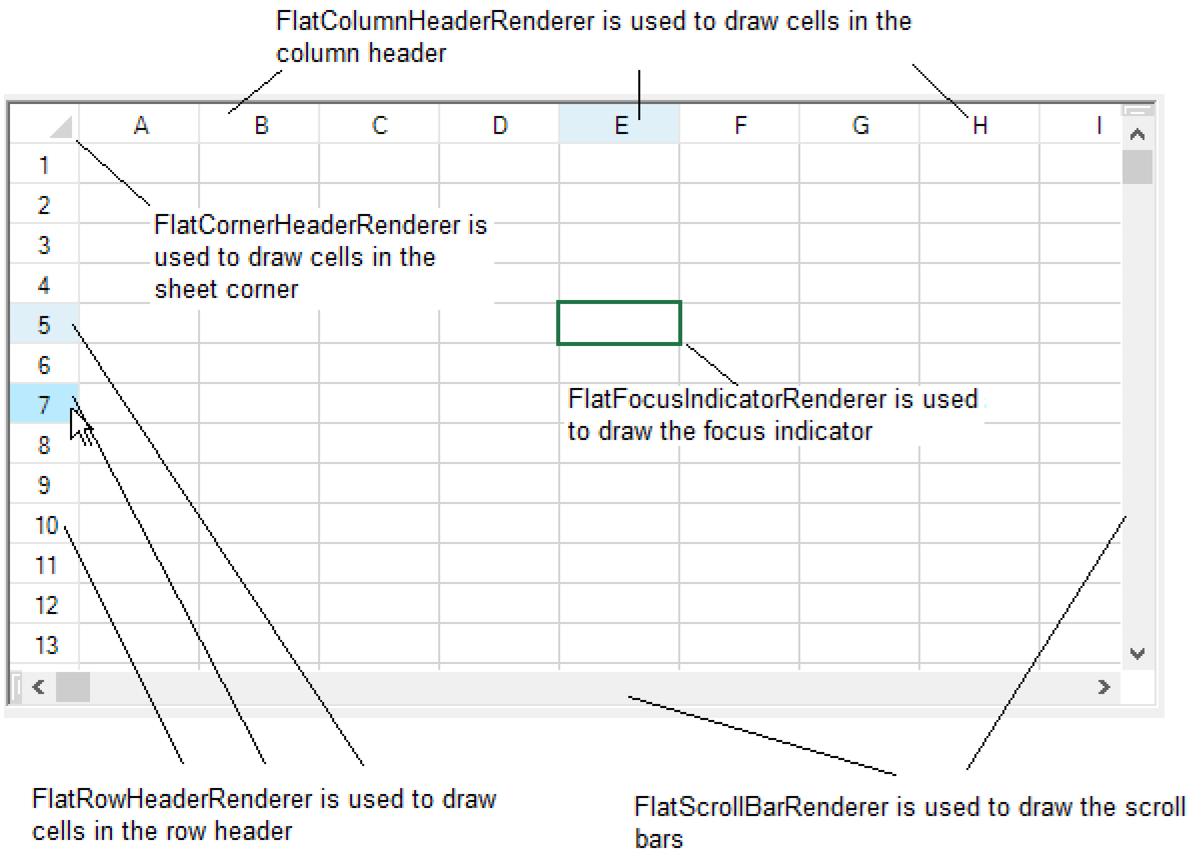
**Using the Spread Designer**

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.

## Customizing the Renderers

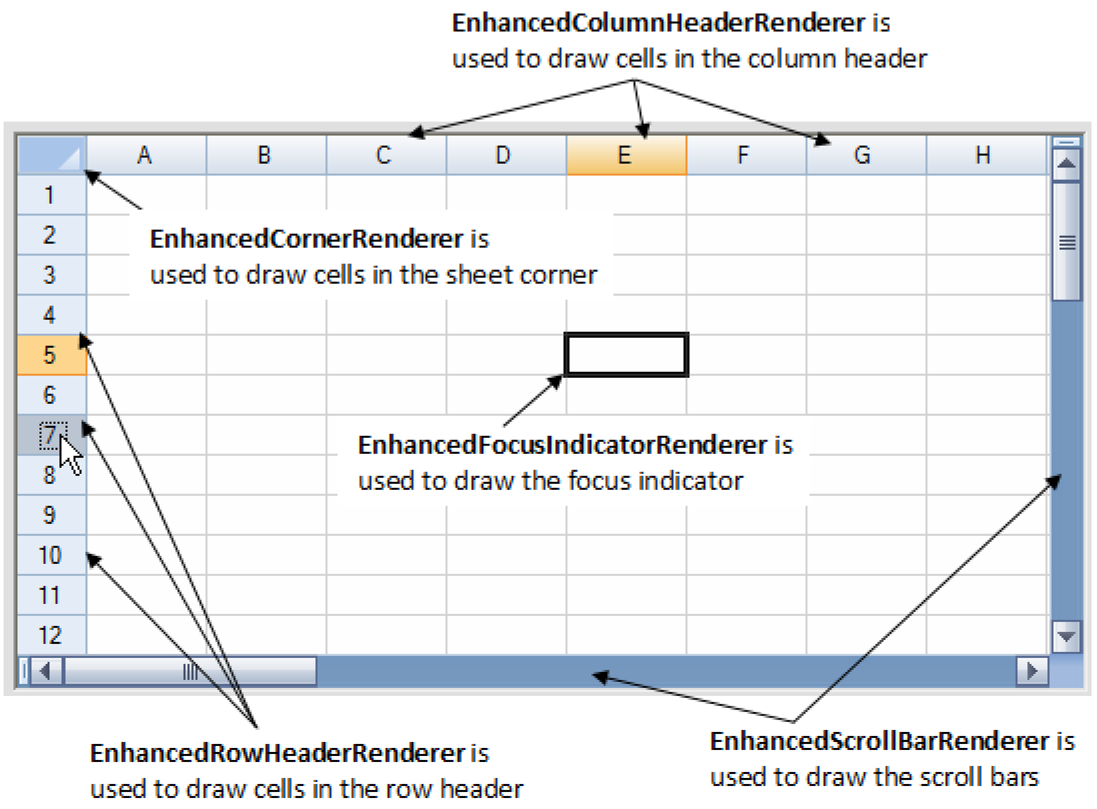
You can customize the renderers used to create the default styles.

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.


FlatColumnHeaderRenderer is used to draw cells in the column header
A
B
C
D
E
F
G
H
I
1
2
3
4
5
6
7
8
9
10
11
12
13
FlatCornerHeaderRenderer is used to draw cells in the sheet corner
FlatFocusIndicatorRenderer is used to draw the focus indicator
FlatRowHeaderRenderer is used to draw cells in the row header
FlatScrollBarRenderer is used to draw the scroll bars

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.


EnhancedColumnHeaderRenderer is used to draw cells in the column header
EnhancedCornerRenderer is used to draw cells in the sheet corner
EnhancedFocusIndicatorRenderer is used to draw the focus indicator
EnhancedRowHeaderRenderer is used to draw cells in the row header
EnhancedScrollBarRenderer is used to draw the scroll bars

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
```

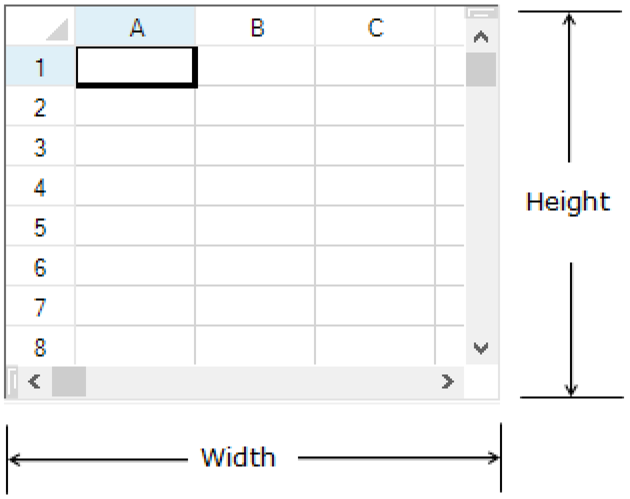
```
fpSpread1.ActiveSheet.RowHeader.ColumnCount = 3;
FarPoint.Win.Spread.CellType.FlatRowHeaderRenderer flatrowheader = new
FarPoint.Win.Spread.CellType.FlatRowHeaderRenderer();
fpSpread1.ActiveSheet.RowHeader.DefaultStyle.Renderer = flatrowheader;

//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
```

## 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 x 20) + (10 x 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#**

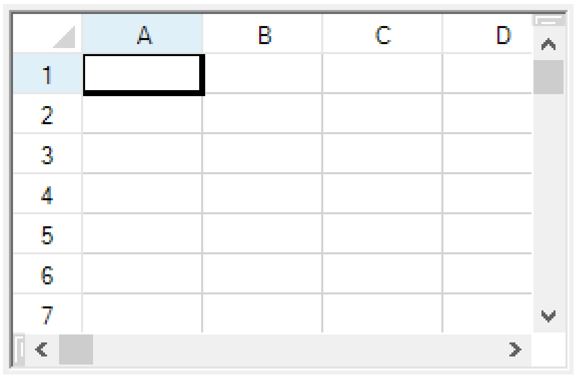
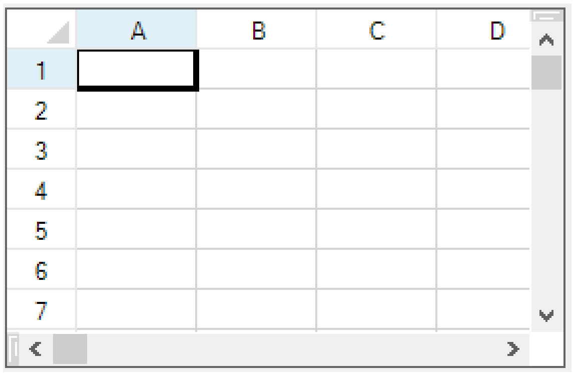
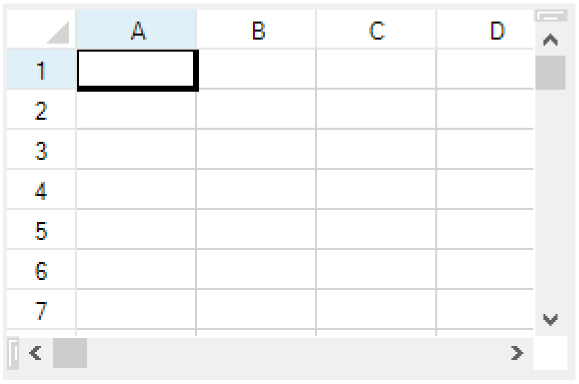
```
fpSpread1.Height = 250;
fpSpread1.Width = 300;
```

**VB**

```
FpSpread1.Height = 250
FpSpread1.Width = 300
```

## Customizing the 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 | Example |
|---|---|
| Fixed, three-dimensional (default) |  |
| 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**

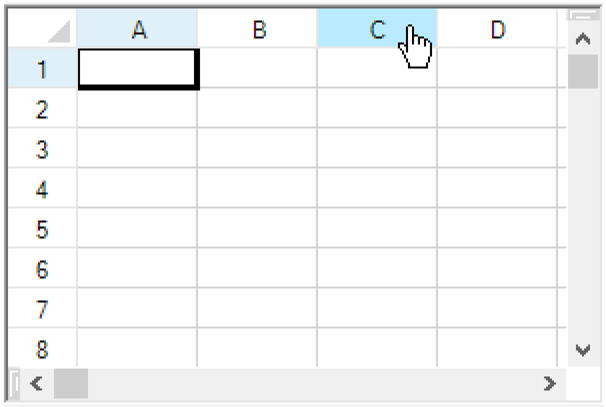
```
FpSpread1.BorderStyle = BorderStyle.FixedSingle
```

**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.

## 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 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 |

## 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.

```
FpSpread1.VisualStyles = FarPoint.Win.VisualStyles.Off
```

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.VisualStyles = FarPoint.Win.VisualStyles.On;
```

**VB**

```
FpSpread1.VisualStyles = FarPoint.Win.VisualStyles.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.

## 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 | Description |
|---|---|
| **RightToLeft ('RightToLeft Property' in the on-line documentation)** | Gets or sets whether the object should paint right to left |
| **OnRightToLeftChanged ('OnRightToLeftChanged Method' in the on-line documentation)** | 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 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**
- **Setting a Background Image for a Sheet**
- **Displaying Grid Lines on a Sheet**
- **Adding a Title and Subtitle to a Sheet**
- **Displaying a Footer for Columns or Groups**
- **Applying a Skin to a Sheet**
- **Creating a Custom Skin for a Sheet**

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 of the Component**
- **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 tasks that relate to setting the user interaction at the sheet level, refer to **Customizing Interaction with a Sheet**.

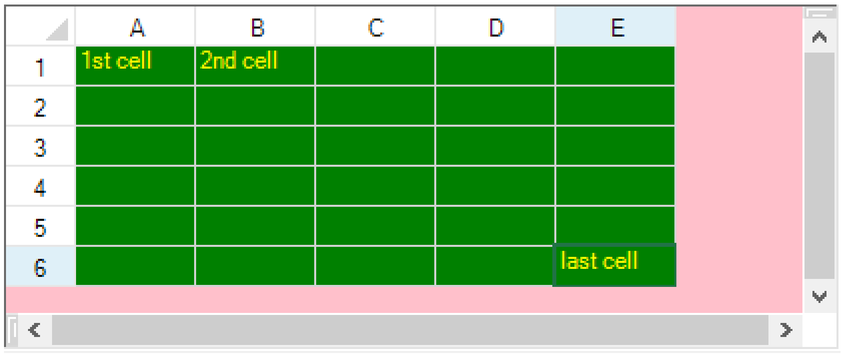
For more details on the objects involved, refer to the **SheetView ('SheetView Class' in the on-line documentation)** class and **SheetViewCollection ('SheetViewCollection Class' in the on-line documentation)** class.

## 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**

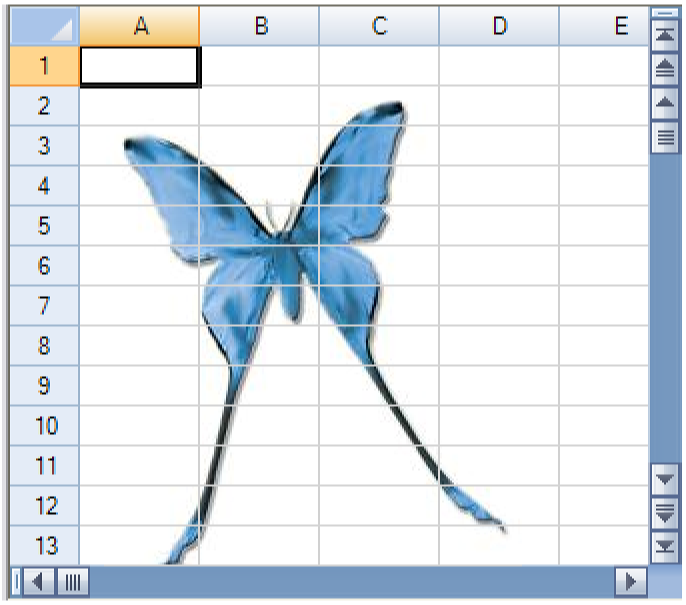
```
' Create a new SheetView object.
Dim newsheet As New FarPoint.Win.Spread.SheetView()
FpSpread1.InterfaceRenderer = Nothing
' Set the SheetView object's background color to light yellow.
newsheet.GrayAreaBackColor = Color.LightYellow
' Assign the SheetView object to the first sheet in the component.
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.

## 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.
2. Set the backcolor of the default style.

**Example**

This example code sets the background image of the sheet.

**C#**

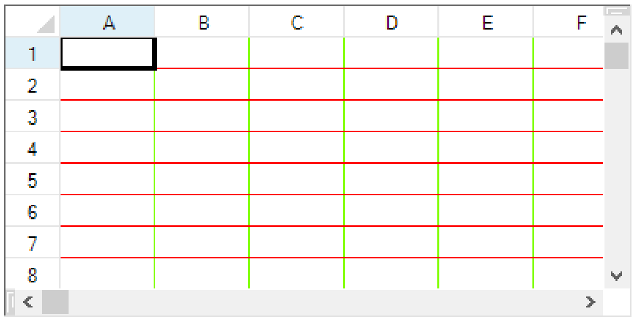
```
private void Form1_Load(object sender, System.EventArgs e)
{
   //Specify background images.
   fpSpread1.BackgroundImage = Image.FromFile("D:\\images\\butterfly.gif");
   //Set "Transparent" to the default background color of the sheet.
   fpSpread1.ActiveSheet.DefaultStyle.BackColor = Color.Transparent;
}
```

**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")
   'Set "Transparent" to the default background color of the sheet.
   FpSpread1.ActiveSheet.DefaultStyle.BackColor = Color.Transparent
End Sub
```

## 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. You can customize the following grid line characteristics:

- **color** - the color of the grid line
- **highlight color** - the highlight color for three-dimensional grid lines
- **shadow color** - the shadow color for three-dimensional grid lines
- **type** - the type of grid line
- **width** - the width in pixels of the grid line

The color is for flat lines; the highlight and shadow color are for the other types of lines.

You can hide the grid line in a particular direction by setting the type of grid line to None. For example, the following code hides the horizontal grid lines.

```
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.

**Note:** You can display lines around individual cells by setting cell borders. For more information, refer to **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**

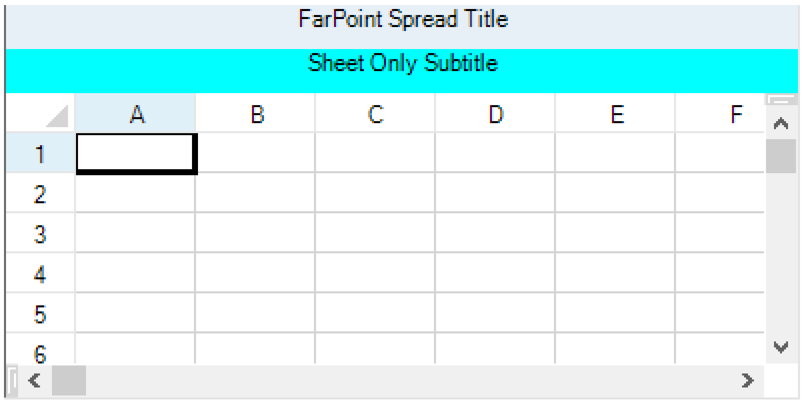
```
Dim HGridLine As New FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Flat, Color.Red)
Dim VGridLine As New FarPoint.Win.Spread.GridLine(FarPoint.Win.Spread.GridLineType.Flat, Color.Chartreuse)
FpSpread1.Sheets(0).HorizontalGridLine = HGridLine
FpSpread1.Sheets(0).VerticalGridLine = 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.
4. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

## 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 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.

**Using a Shortcut**

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**

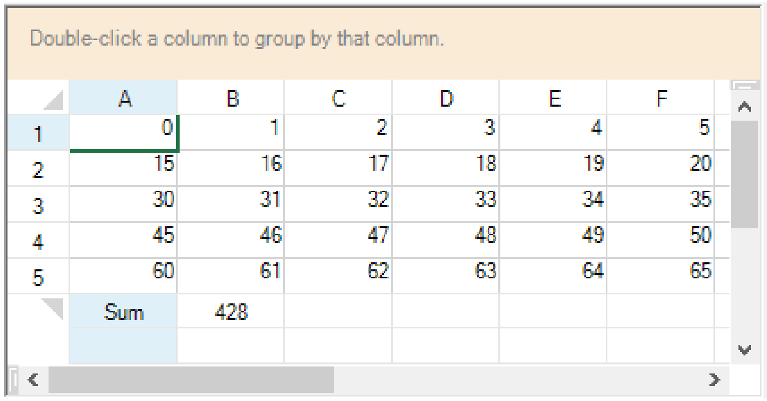
1. From the **Settings** menu, select the **Titles** icon in the **Spread Settings** section.
2. Specify the text and whether to display the title and subtitle.
3. Select **OK** to close the dialog.
4. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

## 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**.

For more information about setting the group appearance, refer to **Setting the Appearance of Grouped Rows**.

For more information on column appearance, refer to **Customizing the Row or Column Appearance**.

**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
```

**Using a Shortcut**

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
```

## 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 | See |
|---|---|
| Creating and applying your own sheet skins | **Creating a Custom Skin for a Sheet** |
| Creating and applying your own cell-level styles | **Creating and Applying a Style for Cells** |
| Saving the sheet skin to a file or stream | **Saving and Loading a Skin** |
| Customizing a skin in the Spread Designer | **SheetSkin Editor (on-line documentation)** in the Spread Designer Guide |
| Sheet skins | **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 | See |
|---|---|
| Applying the built-in sheet skins | **Applying a Skin to a Sheet** |
| Creating and applying your own cell-level styles | **Creating and Applying a Style for Cells** |
| Saving the sheet skin to a file or stream | **Saving and Loading a Skin** |
| Underlying model for skins | **Understanding the Style Model** |
| Customizing a skin in the Spread Designer | **SheetSkin Editor (on-line documentation)** in the Spread Designer Guide |
| Sheet skins | **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.

## 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.

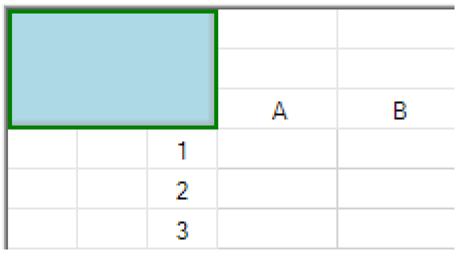
The underlying models of the sheet corner are exposed through the sheet model property and you can change this model. The following classes are involved in the sheet corner.

- 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.



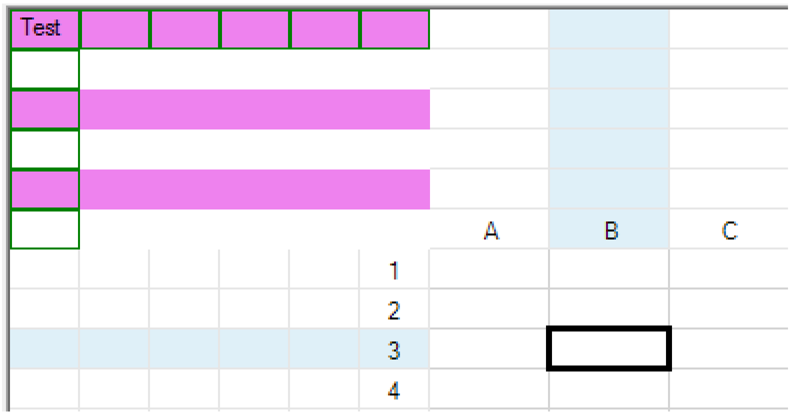
The sheet corner supports right-to-left orientation. When you set Right-to-Left mode in Spread for the entire spreadsheet, the sheet corner also displays with right-to-left orientation as well.

**General Customization**

Several of the properties of a StyleInfo object can be set for the sheet corner cell. These properties include:

- **background color** - the background color of the cell
- **border** - the border around the cell
- **cell type** - the type of cell
- **font** - the font settings of the cell
- **text color** - the color of text color in the cell
- **alignment** - the alignment of text in the cell (horizontal and vertical)

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.VisualStyles =
FarPoint.Win.VisualStyles.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.VisualStyles =
FarPoint.Win.VisualStyles.Off
```

**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#**

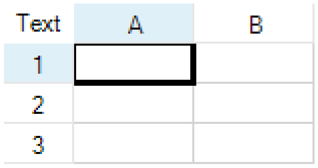
```
FarPoint.Win.Spread.CellType.GeneralCellType gencell = new
FarPoint.Win.Spread.CellType.GeneralCellType();
FarPoint.Win.Picture cornerimage = new
FarPoint.Win.Picture(Image.FromFile("D:\\images\\logocorner.jpg"));
gencell.BackgroundImage = cornerimage;
fpSpread1.ActiveSheet.SheetCornerStyle.CellType = 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 **SheetCorner** properties or overriding the **PaintCell** method. The figure shows the results of the example code in this topic.



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**

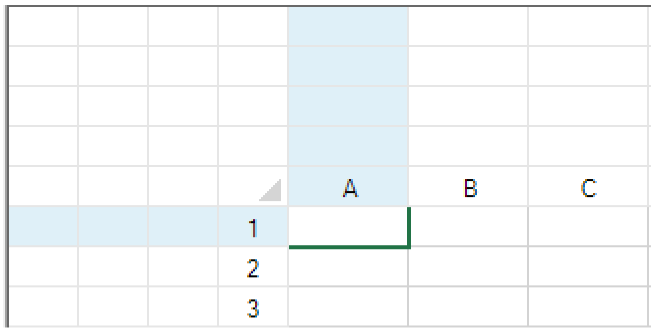
```
Public Class SheetCorner
Inherits FarPoint.Win.Spread.CellType.GeneralCellType
   Public Overrides Sub PaintCell(ByVal g As System.Drawing.Graphics, ByVal r As
System.Drawing.Rectangle, ByVal appearance As FarPoint.Win.Spread.Appearance, ByVal
value As Object, ByVal isSelected As Boolean, ByVal isLocked 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

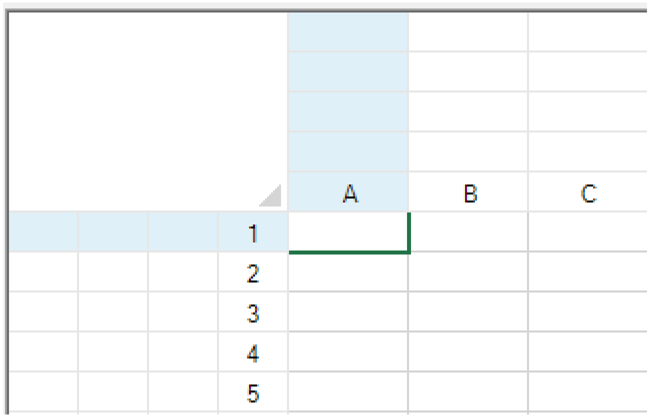
The sheet corner can have a table or grid display with columns and rows of cells, though cells are not editable. You can set the sheet corner to a table (range) display or a single cell display with the **AllowTableCorner ('AllowTableCorner Property' in the on-line documentation)** property of the sheet (**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:



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:



**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
```

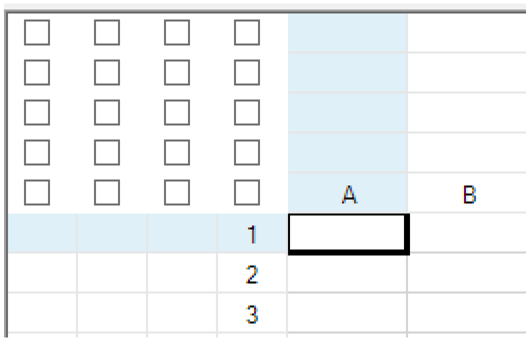
## Customizable Cell in the Sheet Corner

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).

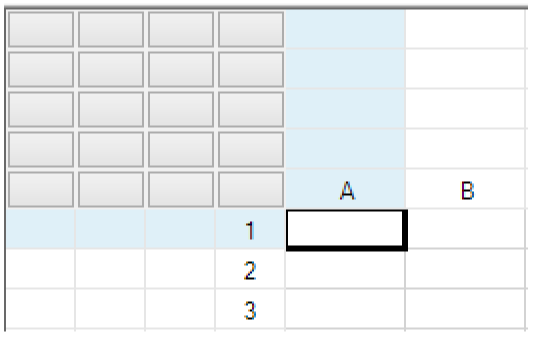
**C#**

```
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#**

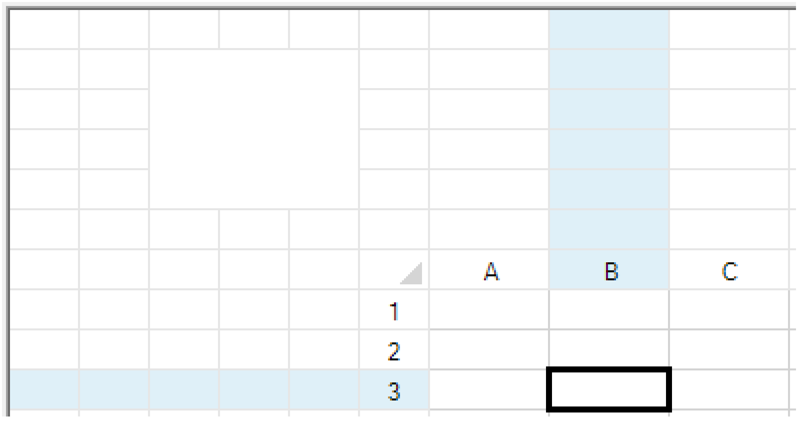
```
fpSpread1.ActiveSheet.AllowTableCorner = true;
fpSpread1.ActiveSheet.SheetCorner.ColumnCount = 4;
fpSpread1.ActiveSheet.SheetCorner.RowCount = 5;
fpSpread1.ActiveSheet.SheetCorner.Cells[2,3].CellType = new
FarPoint.Win.Spread.CellType.ButtonCellType();
```

**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();
```

## Customizing Row, Column, and Cell Appearance

You can customize the appearance of various parts of the Spread component at the row, column, and cell level.

The tasks that relate to setting the appearance of objects in the Spread component include:

- **Customizing the Row or Column Appearance**
- **Customizing the Appearance of Headers**
- **Customizing the Appearance of a Cell**

For information on customizing the interaction with parts of the Spread component, refer to **Customizing Sheet Interaction**.

For information on customizing the appearance using the Spread Designer, refer to the **Spread Designer Guide (on-line documentation)**.

## Customizing the Row or Column Appearance

These tasks relate to setting the appearance of columns or rows in the sheet:

- **Setting the Row Height or Column Width**
- **Resizing the Row or Column to Fit the Data**
- **Finding Rows or Columns That Have Data**
- **Creating Alternating Rows**

When you work with rows and columns, you can work with the objects using the shortcuts in code (**Row ('Row Class' in the on-line documentation)**, **Rows ('Rows Class' in the on-line documentation)**, **Column ('Column Class' in the on-line documentation)**, **Columns ('Columns Class' in the on-line documentation)**, **AlternatingRow ('AlternatingRow Class' in the on-line documentation)**, **AlternatingRows ('AlternatingRows Class' in the on-line documentation)**) 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. 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 (on-line documentation)**.

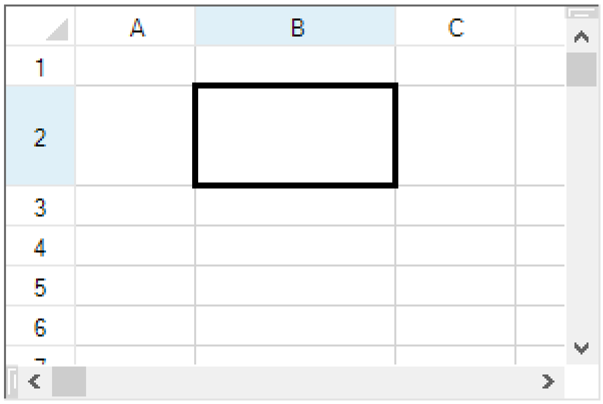
As with most spreadsheet and grid products, Spread does not allow in-cell editing of the cells in the row and column headers.

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. For more information, refer to **Object Parentage**.

- For more information about the appearance of rows as a result of filtering, refer to **Setting the Appearance of Filtered Rows**.
- For more information about the appearance of cells, refer to **Creating and Applying a Style for Cells**.
- For information on the underlying model responsible for rows and columns, refer to the **Understanding the Axis Model**.
- 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.

## 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.

### 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 a Shortcut

Set the Columns shortcut object **Width ('Width Property' in the on-line documentation)** property. You can use -1 for all columns (Columns[-1]).

### 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
```

### Using Code

Set the **Width ('Width Property' in the on-line documentation)** property for a **Column ('Column Class' in the on-line documentation)** object.

**Example**

This example code sets the second column's width to 100 pixels.

**C#**

```
FarPoint.Win.Spread.Column Col1;
Col1 = fpSpread1.Sheets[0].Columns[1];
Col1.Width = 100;
```

**VB**

```
Dim Col1 As FarPoint.Win.Spread.Column
Col1 = FpSpread1.Sheets(0).Columns(1)
Col1.Width = 100
```

**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

You can 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 Row class **GetPreferredHeight ('GetPreferredHeight Method' in the on-line documentation)** method and Column class **GetPreferredWidth ('GetPreferredWidth Method' in the on-line documentation)** method always include the header cells. The **SheetView ('SheetView Class' in the on-line documentation)** class overloaded **GetPreferredColumnWidth ('GetPreferredColumnWidth Method' in the on-line documentation)** method 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.

```
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**.

**Using Code**

Use the listed methods to set the width and height of columns or rows.

**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();
sizecol = col.GetPreferredWidth();
row.Height = sizerow;
col.Width = sizecol;
```

**VB**

```
Dim row As FarPoint.Win.Spread.Row
Dim col As FarPoint.Win.Spread.Column
Dim sizerow As Single
Dim sizecol 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()
sizecol = col.GetPreferredWidth()
row.Height = sizerow
col.Width = sizecol
```

## Finding 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:

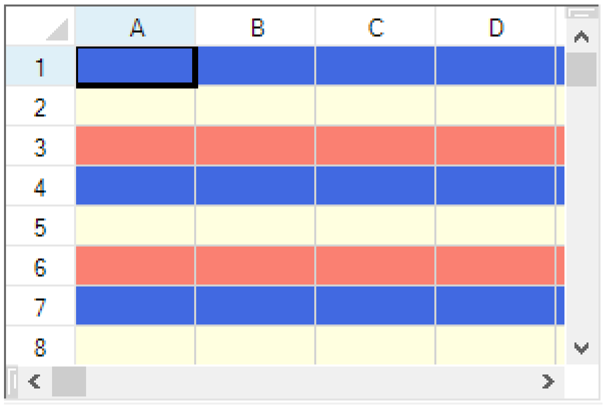
- **NonEmptyColumnCount ('NonEmptyColumnCount Property' in the on-line documentation)** property
- **NonEmptyColumnCount ('NonEmptyColumnCount Property' in the on-line documentation)** property

## 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.



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**.

### Using a Shortcut

1. Set the **Count** property for the **AlternatingRows** shortcut object.
2. Set the various appearance and other properties of the **AlternatingRows** shortcut object, such as the **BackColor** and **ForeColor** properties.
3. Create additional alternating row appearances by setting properties for additional **AlternatingRows** shortcut objects, increasing the index for each appearance you create.

### 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#

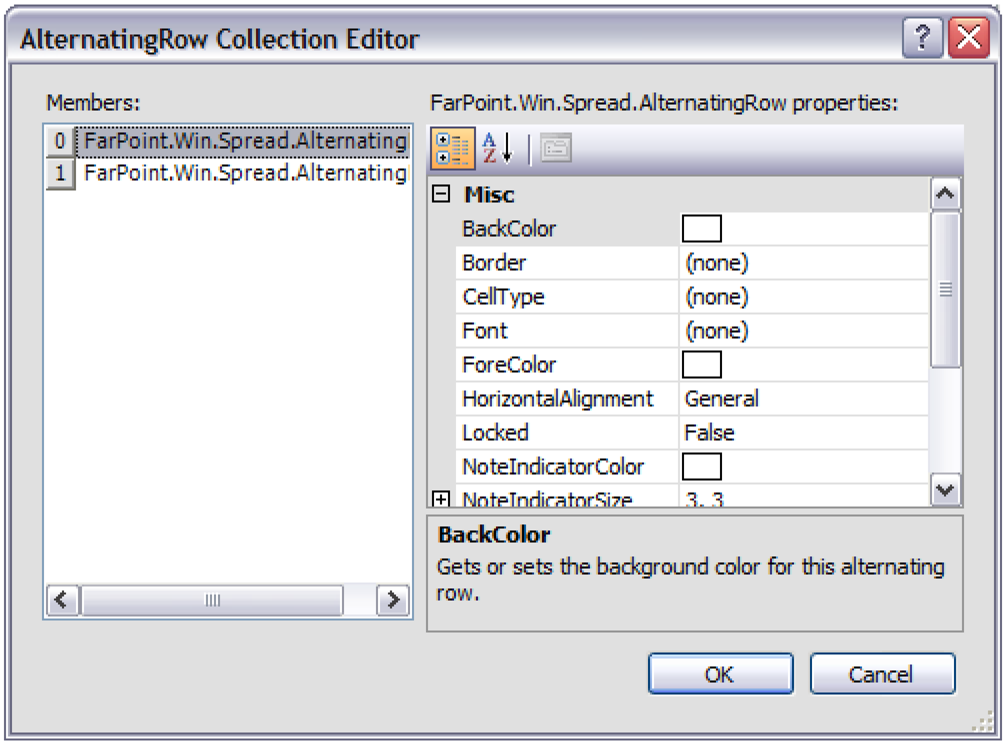
```
fpSpread1.Sheets[0].AlternatingRows.Count = 3;
fpSpread1.Sheets[0].AlternatingRows[0].BackColor = Color.RoyalBlue;
fpSpread1.Sheets[0].AlternatingRows[0].ForeColor = Color.Navy;
fpSpread1.Sheets[0].AlternatingRows[1].BackColor = Color.LightYellow;
fpSpread1.Sheets[0].AlternatingRows[1].ForeColor = 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.
4. Click the **AlternatingRows** button to display the **AlternatingRow Collection Editor** as shown in the following figure.


AlternatingRow Collection Editor

Members:

0 FarPoint.Win.Spread.Alternating
1 FarPoint.Win.Spread.Alternating

FarPoint.Win.Spread.AlternatingRow properties:

| | |
|---|---|
| **Misc** | |
| BackColor | ⬜ |
| Border | (none) |
| CellType | (none) |
| Font | (none) |
| ForeColor | ⬜ |
| HorizontalAlignment | General |
| Locked | False |
| NoteIndicatorColor | ⬜ |
| NoteIndicatorSize | 3, 3 |

**BackColor**
Gets or sets the background color for this alternating row.

5. Select alternating row pattern for which to set properties.
6. Set properties for the selected pattern using the property list.
7. Click **OK** to close the **AlternatingRow Collection Editor**.
8. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

## Customizing the Appearance of Headers

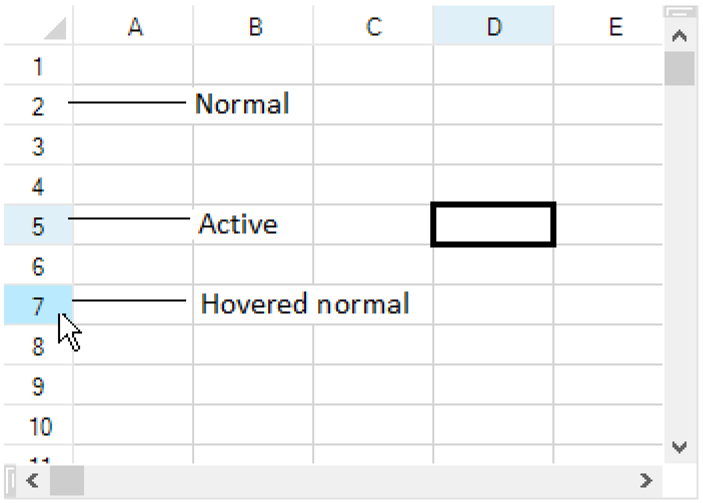
You can customize the appearance of header cells. These tasks relate to setting the appearance of headers for rows or columns in the sheet:

- **Customizing the Default Header Labels**
- **Customizing Header Label Text**
- **Customizing the Style of Header Cells**
- **Adding a Gradient to Header Cells**
- **Customizing the Header Grid Lines**
- **Setting the Height or Width of Header Cells**
- **Creating a Span in a Header**

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.

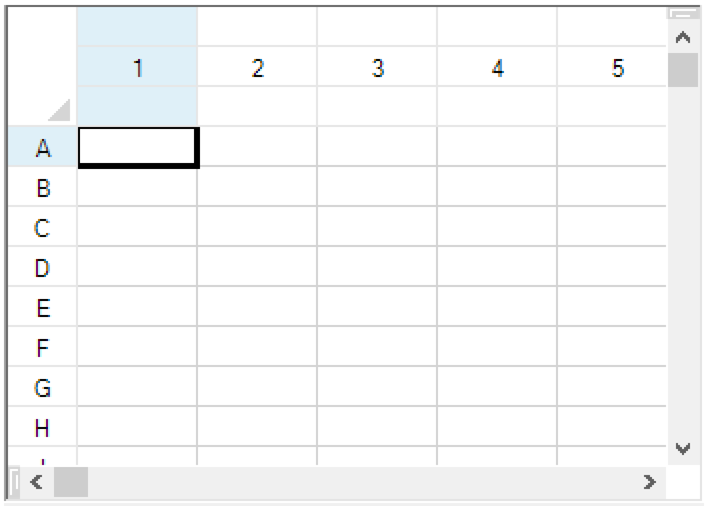


- For information on the appearance of the sheet corner, refer to **Customizing the Sheet Corner Appearance**
- For more information on cell-level properties of header cells, refer to the **Cell ('Cell Class' in the on-line documentation)** and **Cells ('Cells Class' in the on-line documentation)** objects.
- 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 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.

**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.

For more information, refer to these API members:

- **RowHeader ('RowHeader Class' in the on-line documentation)** class **AutoText ('AutoText Property' in the on-line documentation)** and **AutoTextIndex ('AutoTextIndex Property' in the on-line documentation)** properties
- **ColumnHeader ('ColumnHeader Class' in the on-line documentation)** class **AutoText ('AutoText Property' in the on-line documentation)** and **AutoTextIndex ('AutoTextIndex Property' in the on-line documentation)** properties
- **SheetView ('SheetView Class' in the on-line documentation)** class **RowHeaderAutoText ('RowHeaderAutoText Property' in the on-line documentation)** and **RowHeaderAutoTextIndex ('RowHeaderAutoTextIndex Property' in the on-line documentation)** properties
- **SheetView ('SheetView Class' in the on-line documentation)** class **ColumnHeaderAutoText ('ColumnHeaderAutoText Property' in the on-line documentation)** and **ColumnHeaderAutoTextIndex ('ColumnHeaderAutoTextIndex Property' in the on-line documentation)** properties
- **HeaderAutoText ('HeaderAutoText Enumeration' in the on-line documentation)** enumeration

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**.

### 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.

### Using a Shortcut

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**

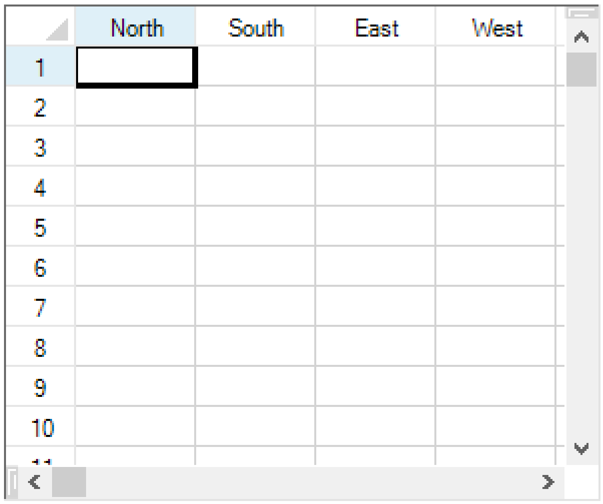
```
' Set the column header to display numbers instead of letters.
FpSpread1.Sheets(0).ColumnHeader.RowCount = 3
FpSpread1.Sheets(0).ColumnHeaderAutoTextIndex = 1
FpSpread1.Sheets(0).ColumnHeaderAutoText = FarPoint.Win.Spread.HeaderAutoText.Numbers
FpSpread1.Sheets(0).RowHeaderAutoText = FarPoint.Win.Spread.HeaderAutoText.Letters
```

**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.

## 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.



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**.

**Using a Shortcut**

- If you want to change the text in a header cell or display text in a cell, set the **Text** property for the ColumnHeader's Cell object to the custom text you want to display. If you want to set the text for multiple header cells, call the ColumnHeader's **SetClip ('SetClip Method' in the on-line documentation)** or **SetClipValue ('SetClipValue Method' in the on-line documentation)** methods.
- If you want to change the labels displayed, set the **Label** property for the ColumnHeader's Column object to the custom text you want to display.

**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.

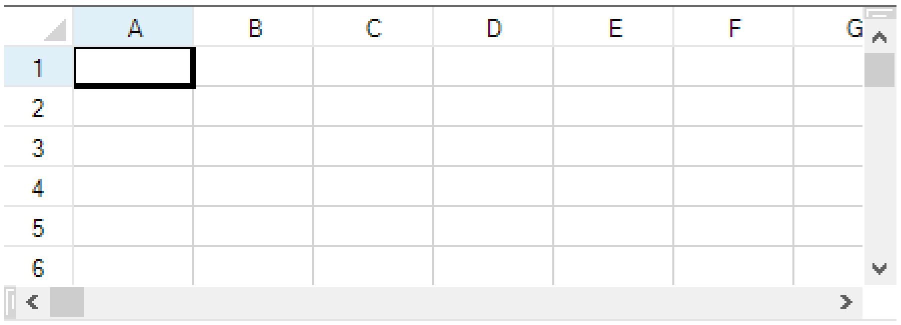
## Customizing the Style of Header Cells

You can customize the style of header cells if you want to change the default appearance. You can set or customize many features, including:

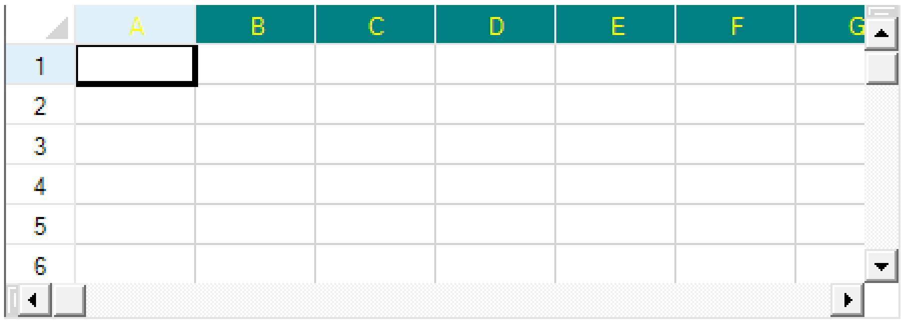
- style properties of the header classes
- members of the renderers
- properties of the default header classes

To customize style properties for the header classes, set the default style of the header cells by setting the **RowHeader ('RowHeader Class' in the on-line documentation) DefaultStyle ('DefaultStyle Property' in the on-line documentation)** property or the **ColumnHeader ('ColumnHeader Class' in the on-line documentation) DefaultStyle ('DefaultStyle Property' in the on-line documentation)** property. 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.

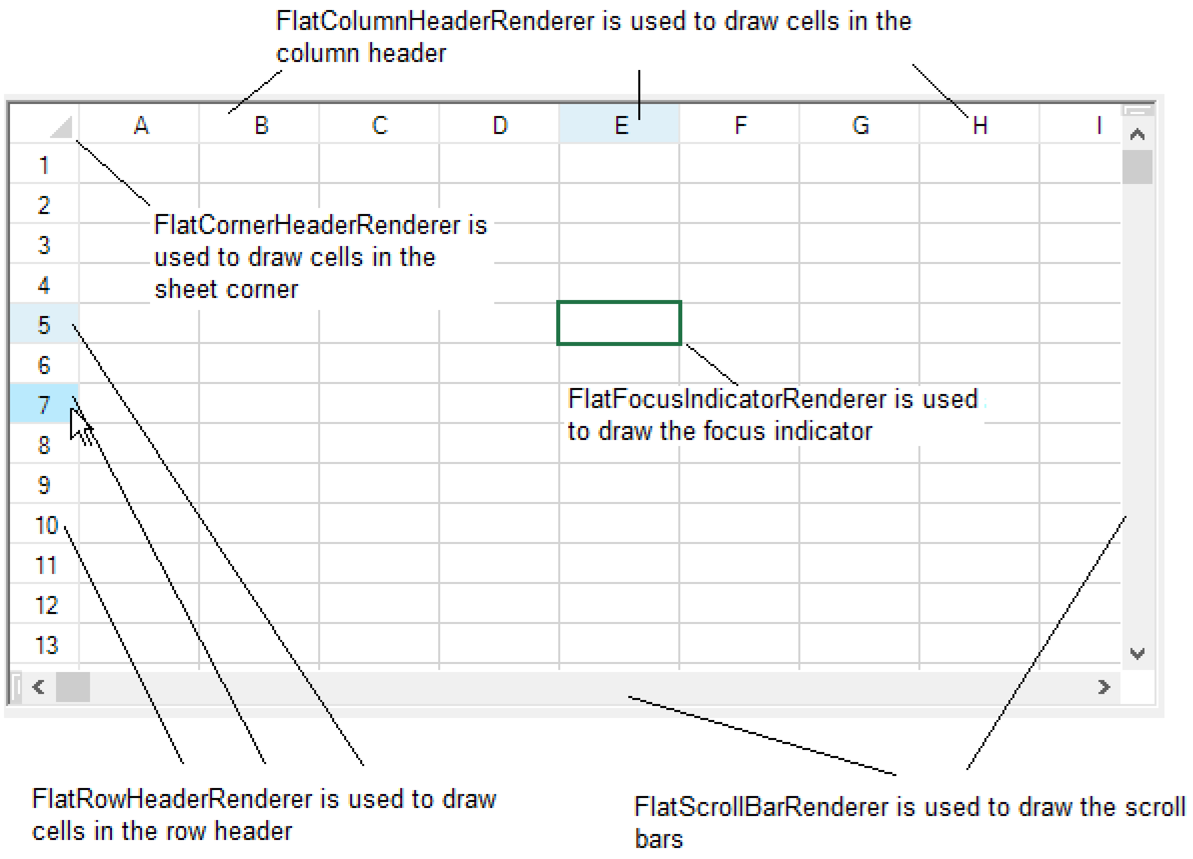
Before



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.


FlatColumnHeaderRenderer is used to draw cells in the column header
FlatCornerHeaderRenderer is used to draw cells in the sheet corner
FlatFocusIndicatorRenderer is used to draw the focus indicator
FlatRowHeaderRenderer is used to draw cells in the row header
FlatScrollBarRenderer is used to draw the scroll bars
A
B
C
D
E
F
G
H
I
1
2
3
4
5
6
7
8
9
10
11
12
13

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**.

## Using a Shortcut

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.
2. 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;
```

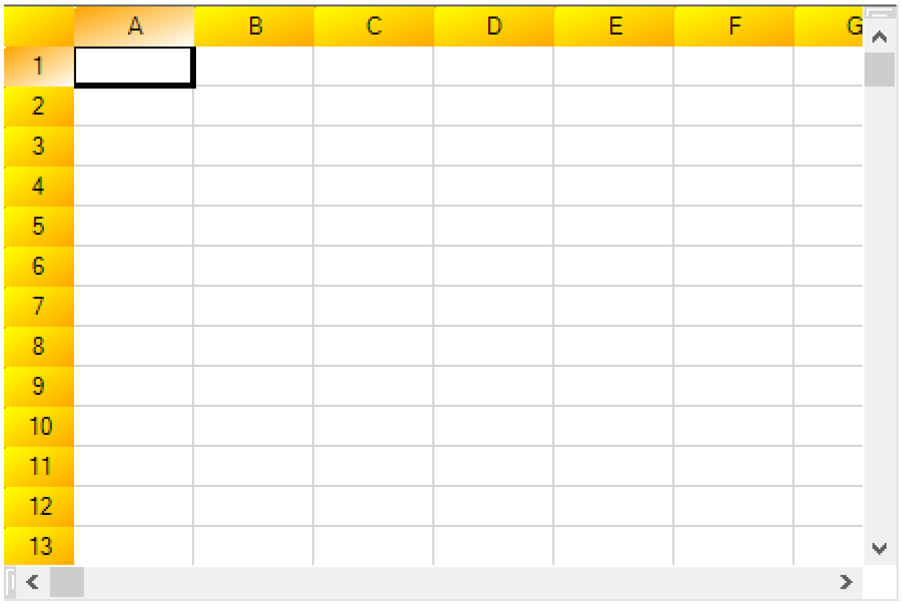
**VB**

```
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
```

## 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.



**Using a Shortcut**

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#**

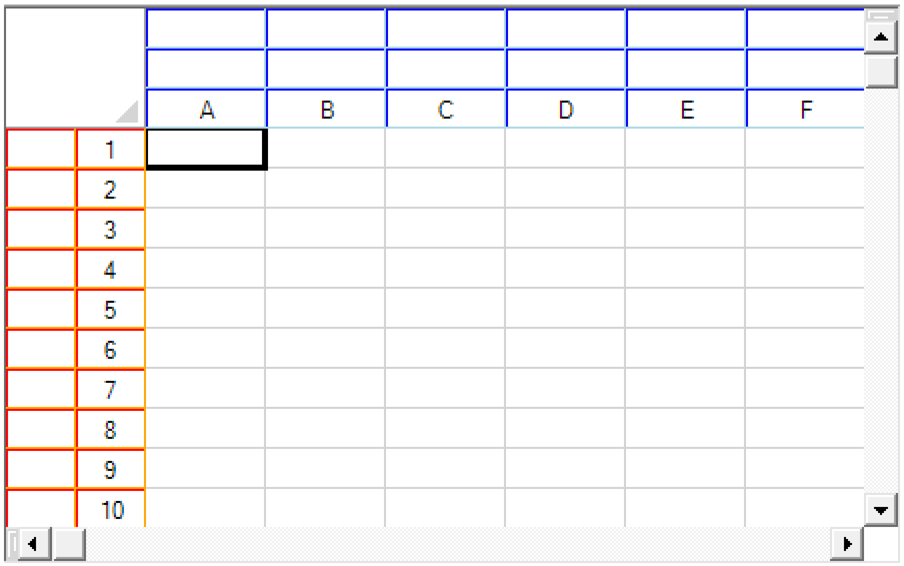
```
FarPoint.Win.Spread.CellType.GradientHeaderRenderer gr = new
FarPoint.Win.Spread.CellType.GradientHeaderRenderer(Color.Yellow,Color.Orange,
Color.YellowGreen, Color.Bisque,Drawing2D.LinearGradientMode.ForwardDiagonal);
FpSpread1.ActiveSheet.ColumnHeader.Rows[0].Renderer = gr;
FpSpread1.ActiveSheet.RowHeader.Columns[0].Renderer = 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
```

## 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.

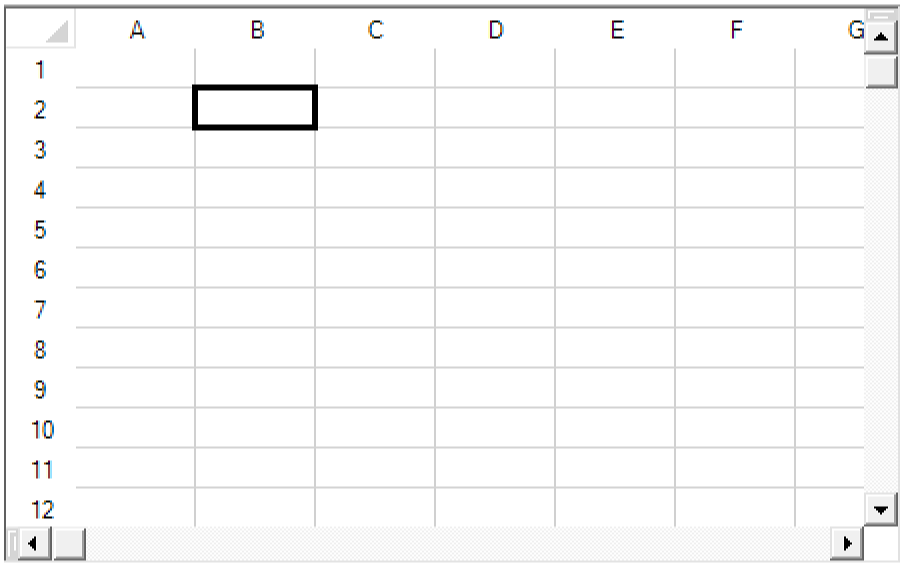
In code, you can customize the grid lines in a header either in the row or column header classes or in the **SheetView ('SheetView Class' in the on-line documentation)** class. You can use any of the following:

- **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.

**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 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
```

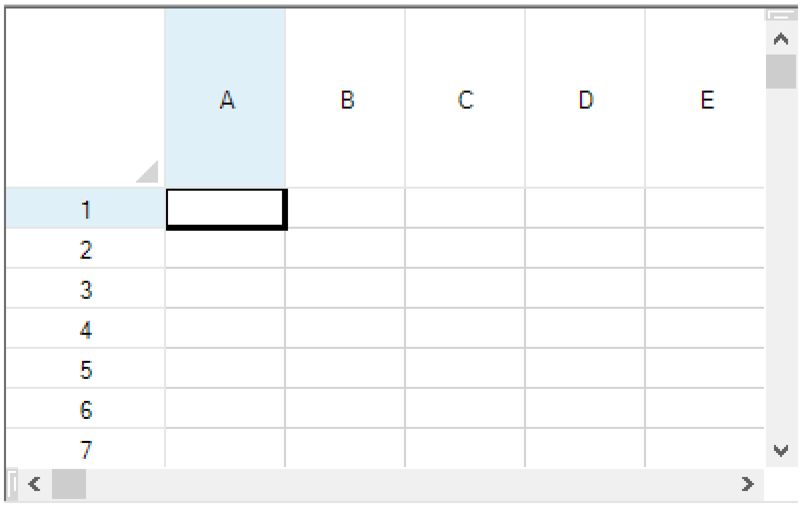
**Using the Spread Designer**

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 **ColumHeaderVerticalGridLine** 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.

## Setting the Height or Width of Header Cells

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.



**Using Code**

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#**

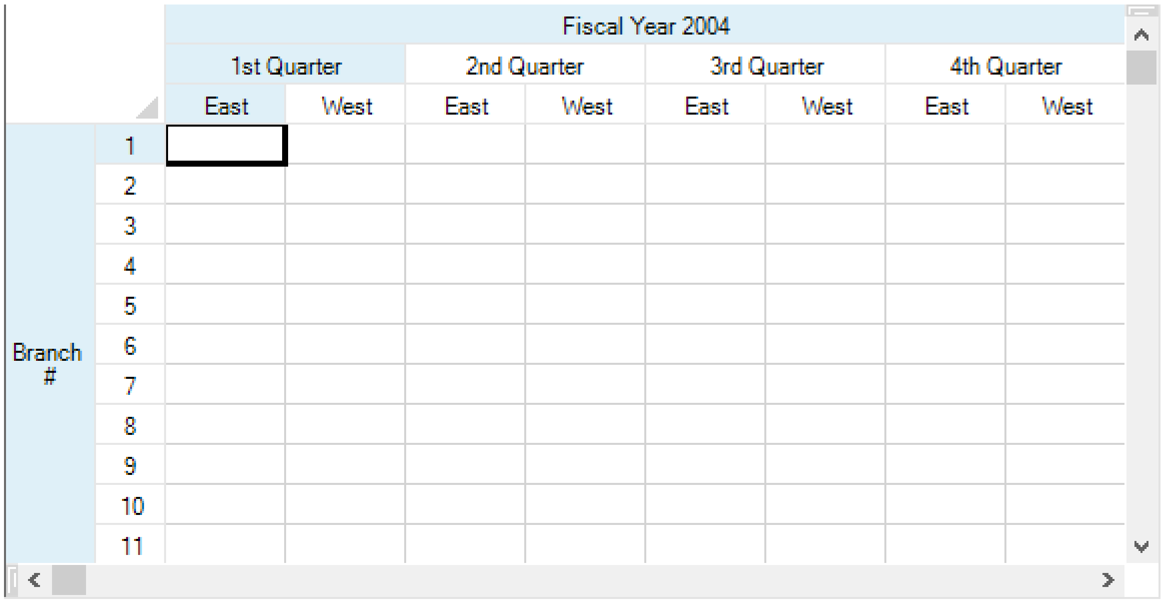
```
private void Form1_Load(object sender, System.EventArgs e)
{
   // Change the column header height to 90 pixels.
   fpSpread1.ActiveSheet.ColumnHeader.Rows[0].Height = 90;
   // Change the row header width to 80 pixels.
   fpSpread1.ActiveSheet.RowHeader.Columns[0].Width = 80;
}
```

**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
```

## 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.



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 **Customizing Header Label 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 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:

- **Coloring a Cell**
- **Setting a Background Image to a Cell**
- **Aligning Cell Contents**
- **Resizing a Cell to Fit the Data**
- **Resizing the Data to Fit the Cell**
- **Customizing Cell Borders**
- **Creating a Span of Cells**
- **Allowing Cells to Merge Automatically**
- **Allowing Cell Data to Overflow**
- **Creating and Applying a Style for Cells**
- **Using Sparklines**

**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**.

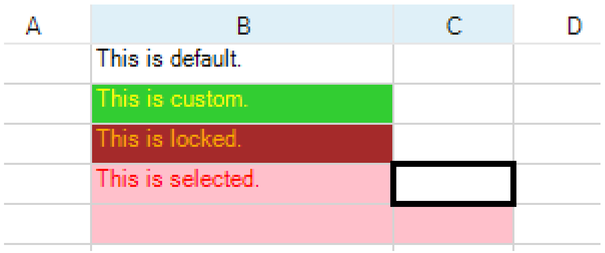
Other cell-level appearance settings are set by the cell type. For more information on settings related to cell types, refer to **Customizing Interaction with Cell Types**. You can edit properties of the **Cells** classes in the **Properties** window (in Spread Designer or in Visual Studio .NET). For 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)**.

## 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

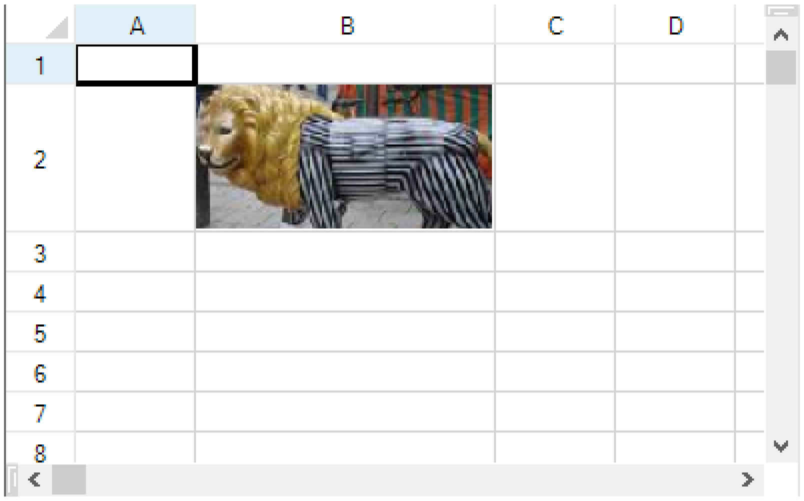
```
Dim cellA1 As FarPoint.Win.Spread.Cell
cellA1 = FpSpread1.ActiveSheet.Cells(0, 0)
cellA1.BackColor = Color.Azure
cellA1.ForeColor = Color.Navy
Dim cellrange As FarPoint.Win.Spread.Cell
cellrange = FpSpread1.ActiveSheet.Cells(2, 2, 3, 3)
cellrange.BackColor = Color.Bisque
```

**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 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;
}
```

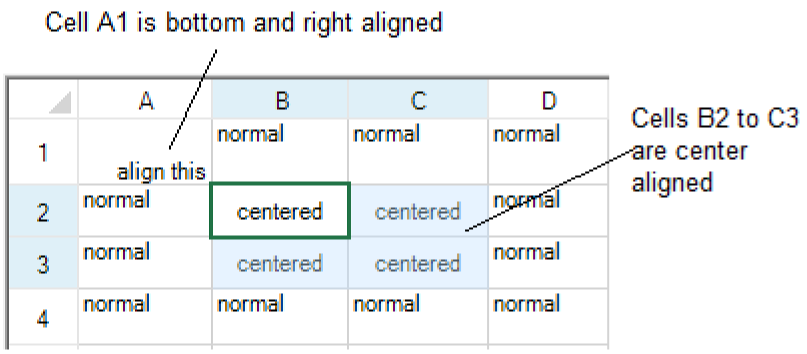
**VB**

```
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
```

## 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
Cells B2 to C3 are center 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 |

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

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 alignment.
7. In the properties list, select the **HorizontalAlignment** property and choose the alignment.
8. To set the vertical alignment, select the **VerticalAlignment** property in the properties list and choose the alignment.
9. Click **OK** to close the **Cell, Column, and Row Editor**.
10. 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;
```

**VB**

```
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;
```

**VB**

```
Dim cellA1 As FarPoint.Win.Spread.Cell
cellA1 = FpSpread1.ActiveSheet.Cells(0, 0)
cellA1.HorizontalAlignment = FarPoint.Win.Spread.CellHorizontalAlignment.Right
```

```
cellA1.VerticalAlignment = FarPoint.Win.Spread.CellVerticalAlignment.Bottom
Dim cellrange As FarPoint.Win.Spread.Cell
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**

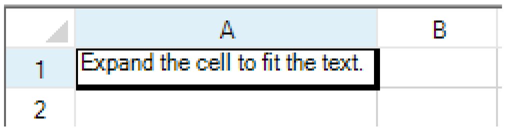
1. In the work area, select the cell or cells for which you want to set the horizontal alignment of the cell contents.
2. In the properties list (in the **Misc** category), select the **HorizontalAlignment** property.
3. Click the drop-down button to display the choices and choose the alignment.
4. Repeat 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.

## 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 **GetPreferredCellSize ('GetPreferredCellSize 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.



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 **GetPreferredRowHeight ('GetPreferredRowHeight 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 **GetPreferredRowHeight ('GetPreferredRowHeight Method' in the on-line documentation)** method, you can get a width for a column using the **GetPreferredColumnWidth ('GetPreferredColumnWidth 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.GetPreferredCellSize(0,0);
fpSpread1.ActiveSheet.Columns[0].Width = sz.Width;
MessageBox.Show("The width of the cell is " + sz.Width.ToString());
```

**VB**

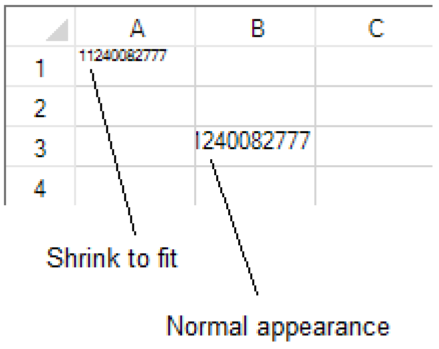
```
Dim sz As System.Drawing.Size
fpSpread1.ActiveSheet.SetValue(0, 0, "Expand the cell to fit the text.")
sz = fpSpread1.ActiveSheet.GetPreferredCellSize(0, 0)
fpSpread1.ActiveSheet.Columns(0).Width = sz.Width
MessageBox.Show("The width of the editor is " & sz.Width.ToString())
```

## 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.



**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.

## Customizing Cell Borders

You can customize the appearance of cells by setting borders for a cell or range of cells.

Tasks that relate to customizing cell borders include:

- **Creating and Customizing Cell Borders**
- **Creating Borders with Diagonal Lines**
- **Creating a Complex Border with Multiple Lines**

## Creating and Customizing Cell Borders

You can customize the appearance of the cells by setting borders for a cell or range of cells. Borders are set only for cells; you can set a border for a column, row, sheet, or range of cells, but the effect is the same as assigning the same border object to each individual cell in the column, row, sheet, or range of cells. For a range of cells, the same border object is used by each cell. A border can be displayed on the left, right, top, or bottom, or around all four sides of a cell or cell range. A border can be displayed as any of the built-in styles shown in the following table or customized borders that you define. By default, no border is displayed. To set the border, use 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.

In the **SheetView ('SheetView Class' in the on-line documentation)** class, there is the **SetOutlineBorder ('SetOutlineBorder Method' in the on-line documentation)** method that has the effect of setting a border around the outside of a range of cells. Actually, it sets individual borders in each cell in the perimeter of the range to achieve this effect. For the inside borders of that range, there is the **SetInsideBorder ('SetInsideBorder Method' in the on-line documentation)** method for setting the borders inside.

You can specify more than one style and color for the same cell, column, row, or block of cells. The cell borders are drawn from left to right and top to bottom in the sheet. 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. That is, cell settings override row, column, and sheet settings, in that order. For more information, see the list of precedence in the description of **Object Parentage**.

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.

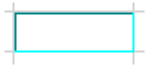
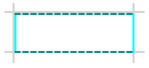
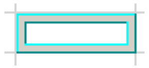
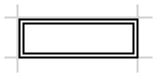
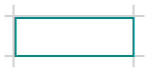
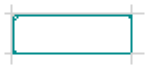
**Border Display**

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. When **BorderCollapse** is set to Separate, (which is the default) the left and top edges of the cell border are painted just inside the grid lines. Thus, the left and top cell border edges are displayed in the left and top rows. When **BorderCollapse** is set to Collapse, the left and top edges of the cell border are painted over the grid lines to the left and top of the cell.

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.

**Border Styles**

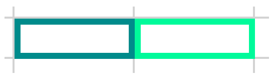
The table below summarizes the different cell border styles.

| 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)** |

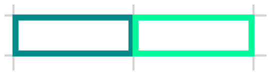
Different border styles let you set different options. For example, the complex border lets you set different styles of border display for each side of the cell. The **ComplexBorder ('ComplexBorder Class' in the on-line documentation)** class also allows you to create diagonal border lines. In the example shown above, the top and bottom borders are dashed borders and have a different color from the left and right borders. For each of these border styles, you can turn off the display of the border on any side of the cell.

**Overlapping Borders**

Cell borders are applied around the edge of each cell, and can overlap other cell borders but do not do so by default. The following figure shows two sets of cells. In the first set, the cell borders do not overlap, and are separate. In the second set, the cell borders overlap.

| Borders | Example Appearance |
|---|---|
| Separate borders (not collapsed) |  |

Collapsed borders (overlapping)



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. 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. Keep in mind that the sheet is drawn from left to right and from top to bottom on the screen. 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.

### Different from 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.

### Example

This 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
```

### Using Code

1. Create a new border object of the type of border you want to create (**BevelBorder ('BevelBorder Class' in the on-line documentation)**, **ComplexBorder ('ComplexBorder Class' in the on-line documentation)**, and so on).
2. Create a new **SheetView ('SheetView Class' in the on-line documentation)** object.
3. Set the **Border ('Border Property' in the on-line documentation)** property for a cell (or row or column) object equal to the border object you created.
4. Assign the **SheetView ('SheetView Class' in the on-line documentation)** object to a sheet in the component.

### Example

This example code creates a bevel border and then sets a cell’s border to be the bevel border.

**C#**

```
// Create a new bevel border.
FarPoint.Win.BevelBorder bevelbrdr = new
FarPoint.Win.BevelBorder(FarPoint.Win.BevelBorderType.Raised, Color.Cyan,
Color.DarkCyan);
// Create a new SheetView object.
FarPoint.Win.Spread.SheetView newsheet=new FarPoint.Win.Spread.SheetView();
// Set a cell's border to be the bevel border.
newsheet.Cells[4, 3].Border = bevelbrdr;
// Assign the SheetView object to the first sheet.
fpSpread1.Sheets[0] = newsheet;
```

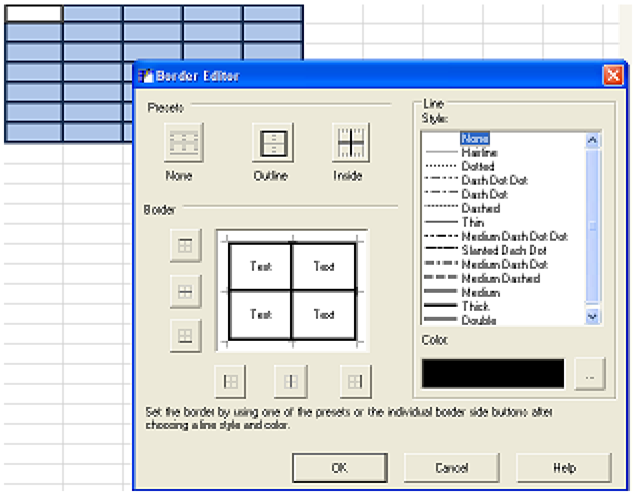
**VB**

```
' Create a new bevel border.
Dim bevelbrdr As New FarPoint.Win.BevelBorder(FarPoint.Win.BevelBorderType.Raised,
Color.Cyan, Color.DarkCyan)
' Create a new SheetView object.
Dim newsheet As New FarPoint.Win.Spread.SheetView()
' Set a cell's border to be the bevel border.
newsheet.Cells(4, 3).Border = bevelbrdr
' Assign the SheetView object to the first sheet.
FpSpread1.Sheets(0) = newsheet
```

### Using the Spread Designer

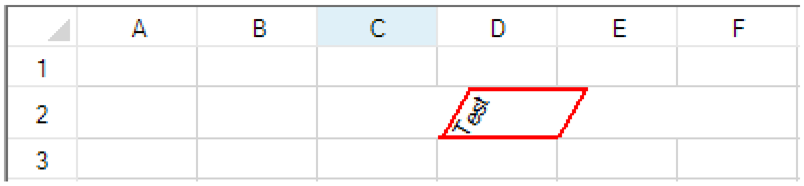
1. Select the sheet tab for the sheet for which you want to set a cell border.
2. Select the cell or range of cells for which you want to set the border.
3. Right-click and select **Borders**, or in the property list (in the **Misc** category), set the **Border** property.
4. If you want to customize 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.

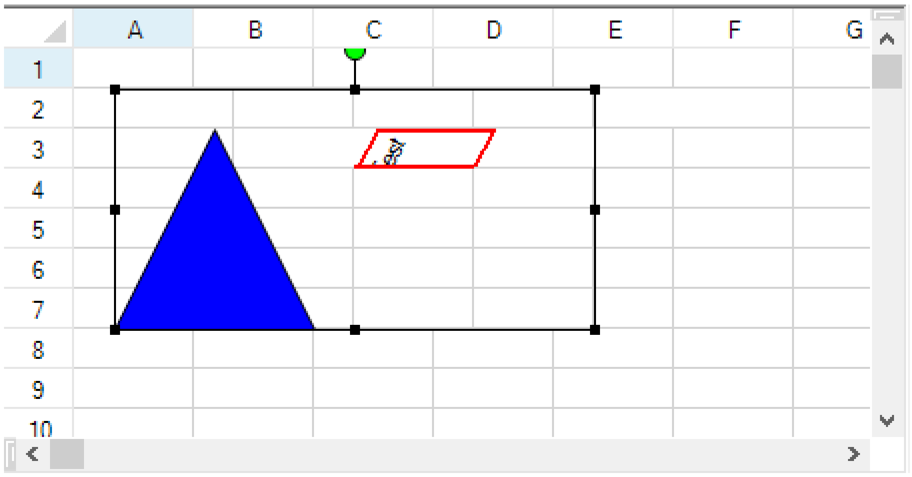


## Creating Borders with Diagonal Lines

You can create cell borders with diagonal lines as shown in the following image.



You can also display the diagonal lines in a cell border that is part of a shape as displayed in the following image.



The cell must have a border and the text orientation must not be zero. The **EnableDiagonalLine ('EnableDiagonalLine Property' in the on-line documentation)** property should also be set to true.

The **ComplexBorder ('ComplexBorder Class' in the on-line documentation)** class also allows you to create diagonal border lines in the cell.

**Using Code**

1. Create a text cell.
2. Set the **TextOrientation ('TextOrientation Property' in the on-line documentation)** property.
3. Set the **TextRotationAngle ('TextRotationAngle Property' in the on-line documentation)** property.

**Example**

This example code creates a text cell with a diagonal border.

**C#**

```
fpSpread1.Sheets[0].Cells[1, 3].Text = "Test"; //Cell has to have value
fpSpread1.Sheets[0].Cells[1, 3].Border = new FarPoint.Win.ComplexBorder(new
FarPoint.Win.ComplexBorderSide(Color.Red, 2));// Cell has to have border

FarPoint.Win.Spread.CellType.TextCellType cellType = new
FarPoint.Win.Spread.CellType.TextCellType();
cellType.TextOrientation = FarPoint.Win.TextOrientation.TextRotateCustom;
cellType.TextRotationAngle = 60; // Cell has to have rotation angle to see the effect.
fpSpread1.Sheets[0].Cells[1, 3].CellType = cellType;
fpSpread1.Sheets[0].EnableDiagonalLine = true;
```

**VB**

```
FpSpread1.Sheets(0).Cells(1, 3).Text = "Test" 'Cell has to have value
FpSpread1.Sheets(0).Cells(1, 3).Border = New FarPoint.Win.ComplexBorder(New
FarPoint.Win.ComplexBorderSide(Color.Red, 2)) ' Cell has to have border

Dim cellType As New FarPoint.Win.Spread.CellType.TextCellType()
cellType.TextOrientation = FarPoint.Win.TextOrientation.TextRotateCustom
cellType.TextRotationAngle = 60 ' Cell has to have rotation angle to see the effect.
FpSpread1.Sheets(0).Cells(1, 3).CellType = cellType
FpSpread1.Sheets(0).EnableDiagonalLine = True
```

**Example**

This example code creates a text cell with a diagonal border and a shape that contains the cell.

**C#**

```
fpSpread1.Sheets[0].Cells[1, 3].Text = "Test"; //Cell has to have value
fpSpread1.Sheets[0].Cells[1, 3].Border = new FarPoint.Win.ComplexBorder(new
FarPoint.Win.ComplexBorderSide(Color.Red, 2));// Cell has to have border

FarPoint.Win.Spread.CellType.TextCellType cellType = new
FarPoint.Win.Spread.CellType.TextCellType();
cellType.TextOrientation = FarPoint.Win.TextOrientation.TextRotateCustom;
cellType.TextRotationAngle = 60; // Cell has to have rotation angle to see the effect.
fpSpread1.Sheets[0].Cells[1, 3].CellType = cellType;

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:E6";
test.Location = new System.Drawing.Point(20, 20);
fpSpread1.Sheets[0].AddShape(test);
fpSpread1.Sheets[0].EnableDiagonalLine = true;
```

**VB**

```
FpSpread1.Sheets(0).Cells(1, 3).Text = "Test" 'Cell has to have value
FpSpread1.Sheets(0).Cells(1, 3).Border = New FarPoint.Win.ComplexBorder(New
FarPoint.Win.ComplexBorderSide(Color.Red, 2)) ' Cell has to have border

Dim cellType As New FarPoint.Win.Spread.CellType.TextCellType()
cellType.TextOrientation = FarPoint.Win.TextOrientation.TextRotateCustom
cellType.TextRotationAngle = 60 ' Cell has to have rotation angle to see the effect.
FpSpread1.Sheets(0).Cells(1, 3).CellType = cellType

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:E6"
test.Location = New System.Drawing.Point(20, 20)
FpSpread1.Sheets(0).AddShape(test)
FpSpread1.Sheets(0).EnableDiagonalLine = True
```

## Creating a Complex Border with Multiple Lines

You can create a cell border with multiple lines using the complex border.

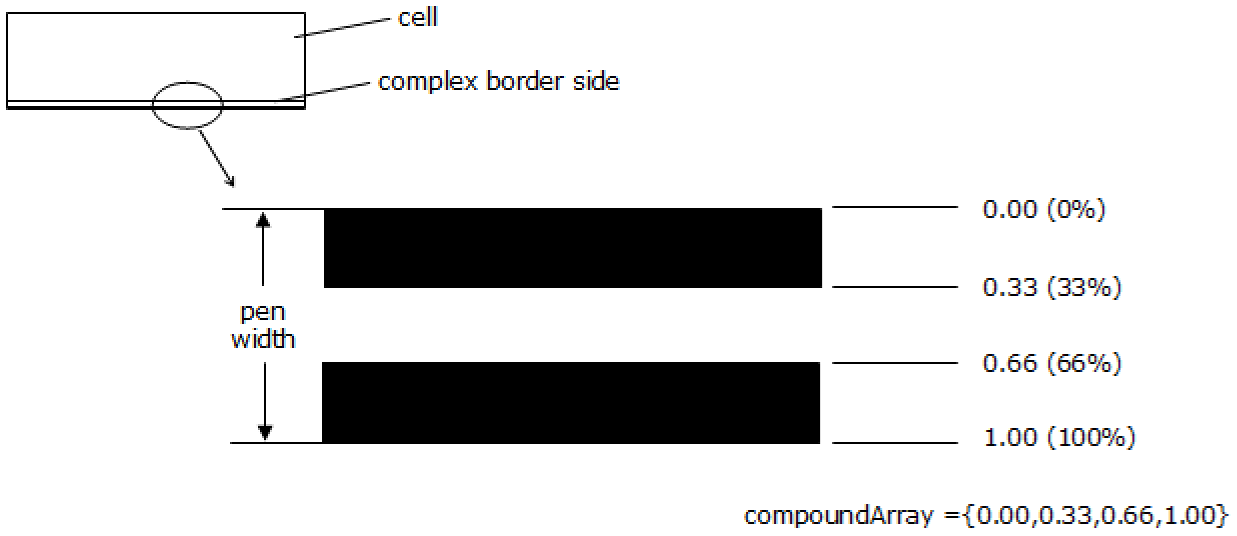
**Using Code**

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.



**C#**

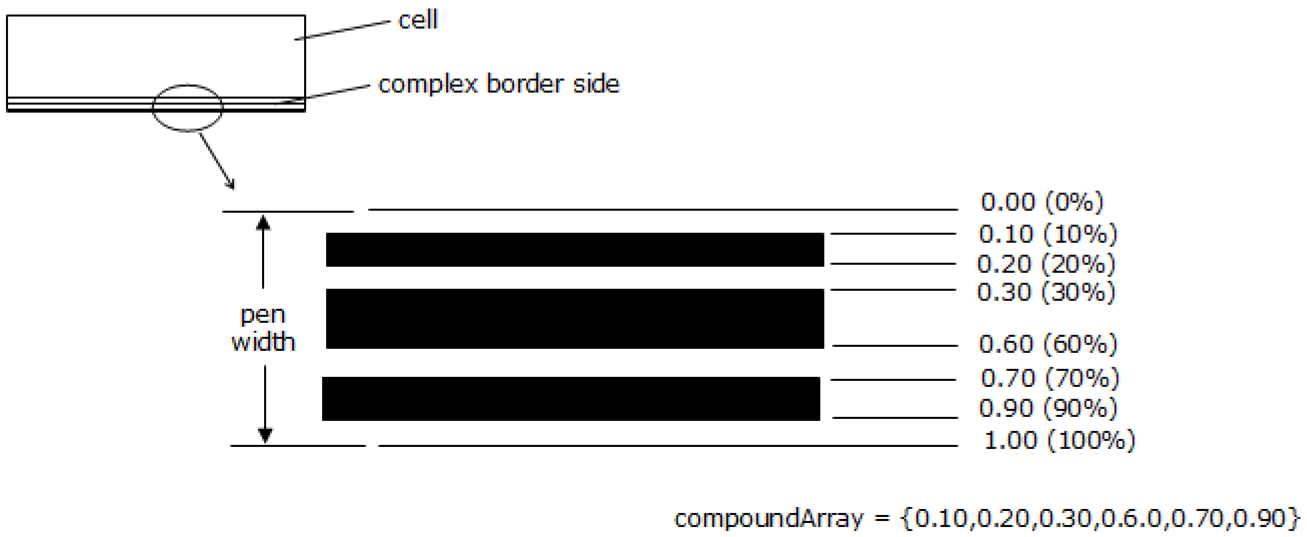
```
// Create a new complex border side with two lines.
FarPoint.Win.ComplexBorderSide bottomborder = new FarPoint.Win.ComplexBorderSide(true,
Color.Black, 3, System.Drawing.Drawing2D.DashStyle.Solid, null, new Single[] {0f,
0.33f, 0.66f, 1f});
fpSpread1.Sheets[0].Cells[3, 7].Border = new FarPoint.Win.ComplexBorder(null, 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.



**C#**

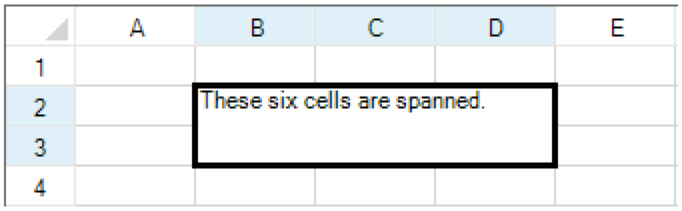
```
// Create a new complex border side with three lines.
FarPoint.Win.ComplexBorderSide bottomborder = new FarPoint.Win.ComplexBorderSide(true,
Color.Black, 3, System.Drawing.Drawing2D.DashStyle.Solid, null, new Single[] {0.1f,
0.2f, 0.3f, 0.6f, 0.7f, 0.9f});
fpSpread1.Sheets[0].Cells[3, 7].Border = 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)
```

## 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.



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.

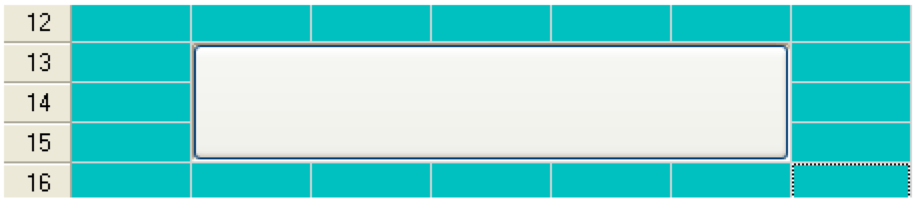
**C#**

```
// 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)
```

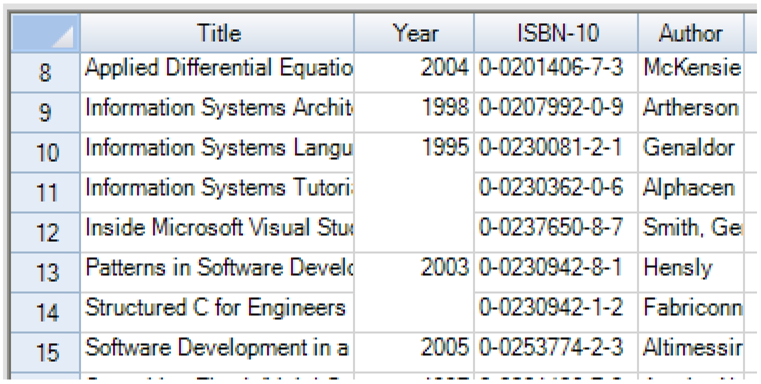
**Using the Spread Designer**

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.
   

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 Archit | 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 Stu | | 0-0237650-8-7 | Smith, Ge |
| 13 | Patterns in Software Develo | 2003 | 0-0230942-8-1 | Hensly |
| 14 | Structured C for Engineers | | 0-0230942-1-2 | Fabriconn |
| 15 | Software Development in a | 2005 | 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; b; c; c} and cells B1:B8 contain {1; 1; 1; 1; 2; 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:

- **GetColumMerge ('GetColumnMerge Method' in the on-line documentation)** and **SetColumMerge ('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 **SetColumMerge ('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 **SetColumMerge ('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**

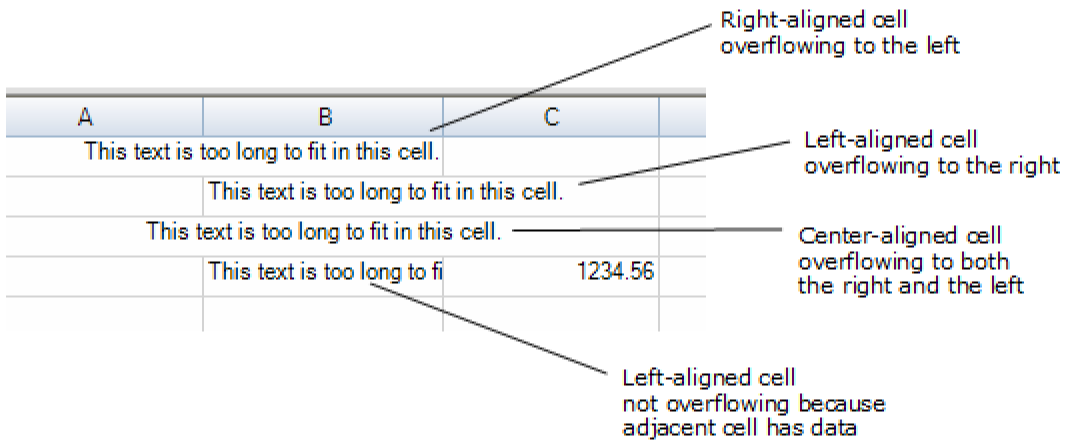
```
Dim Sheet0 As FarPoint.Win.Spread.SheetView
Sheet0 = FpSpread1.Sheets(0)
Sheet0.SetRowMerge(-1, FarPoint.Win.Spread.Model.MergePolicy.Always)
Sheet0.SetColumnMerge(-1, FarPoint.Win.Spread.Model.MergePolicy.Always)
```

## 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.


A
B
C
This text is too long to fit in this cell.
This text is too long to fit in this cell.
This text is too long to fit in this cell.
This text is too long to fi
1234.56
Right-aligned cell overflowing to the left
Left-aligned cell overflowing to the right
Center-aligned cell overflowing to both the right and the left
Left-aligned cell not overflowing because adjacent cell has data

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**.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

## 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";
```

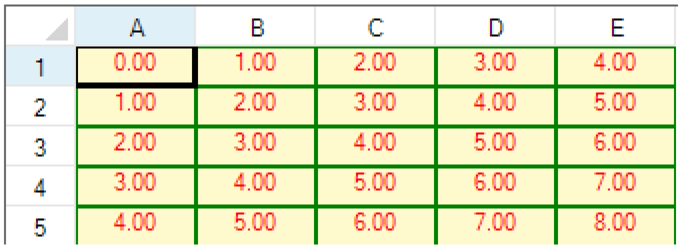
**VB**

```
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
```

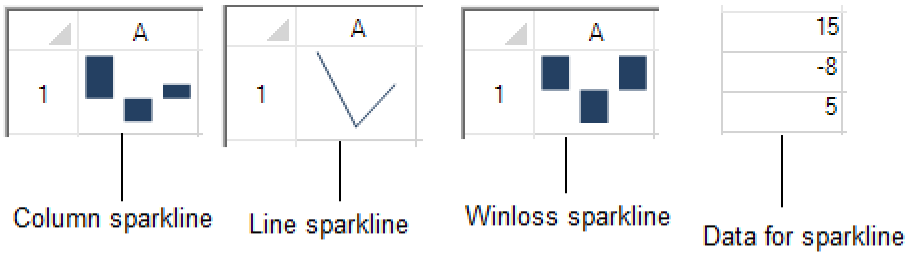
```
            FpSpread1.ActiveSheet.SetValue(i, j, i + j)
        Next
    Next
End Sub
```

**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 Sparklines

You can create a sparkline in a cell which is a small graph that uses data from a range of cells. The data for the sparkline is limited to one column or row of values. You can set the sparkline type to column, line, or winloss, as shown in the following figure. These images were created using a minimum axis of -9 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.

The graphs can display colors for the marker points. You can set colors for the high, low, negative, first, and last points.

The graphs have horizontal and vertical axes.

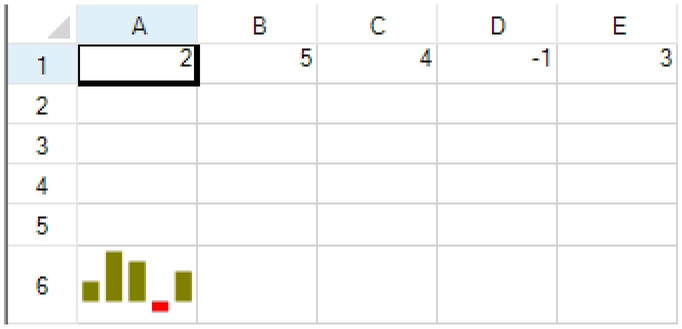
Sparklines are stored as groups. A group contains at least one sparkline.

For more information, see the following topics:

- **Adding a Sparkline to a Cell**
- **Customizing Markers and Points**
- **Specifying Horizontal and Vertical Axes**
- **Working with Sparklines**

## Adding a Sparkline to a Cell

You can add a sparkline to a cell using code or the designer.



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

**Using Code**

1. Specify a cell in which to create the sparkline.
2. Specify a range of cells for the data.
3. Set any properties for the sparkline (such as points and colors).
4. Add the sparkline to the cell.

**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
```

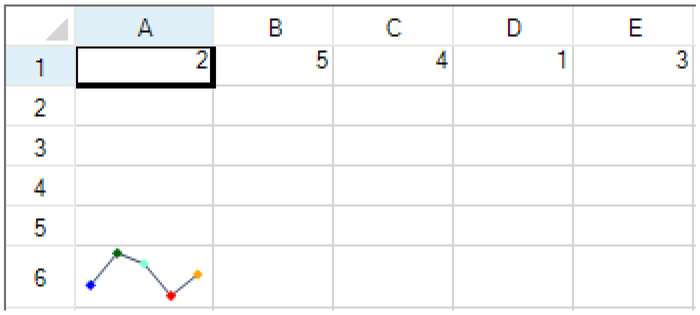
```
FpSpread1.Sheets(0) = sv
sv.Cells(0, 0).Value = 2
sv.Cells(0, 1).Value = 5
sv.Cells(0, 2).Value = 4
sv.Cells(0, 3).Value = -1
sv.Cells(0, 4).Value = 3
FpSpread1.Sheets(0).AddSparkline(data, data2, FarPoint.Win.Spread.SparklineType.Column,
ex)
```

**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 **Edit 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.

## 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)
```

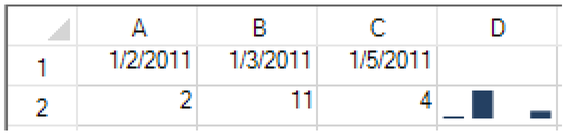
**Using the Spread Designer**

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.

## 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.

## Working with Sparklines

You can group, delete, cut, copy, and switch rows and columns with sparklines.

Grouping merges sparklines into a new group and removes them from the old 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. 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 you delete a group, all sparklines in the group are also deleted. You can clear a sparkline in code with the **ClearSparklines ('ClearSparklines Method' in the on-line documentation)** method in the **Sheetview** class.

You can use standard Windows shortcut keys to cut, copy, or paste 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.

You can switch the range of data used in the sparkline from row to column or column to row. Use the **SwitchRowColumn ('SwitchRowColumn Method' in the on-line documentation)** method in the **ExcelSparklineGroup ('ExcelSparklineGroup Class' in the on-line documentation)** class in code. The range of data 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.

**Using Code**

1. Use the **GroupSparkline ('GroupSparkline Method' in the on-line documentation)** method to group sparkline cells.
2. Use the **UnGroupSparkline ('UngroupSparkline Method' in the on-line documentation)** method to ungroup sparkline cells.

**Example**

This 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.

## Customizing Sheet 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:

- **Customizing Interaction in the Overall Component**
- **Customizing Interaction with a Sheet**
- **Customizing User Searching of Data**
- **Customizing User Selection of Data**
- **Setting and Resetting User Interaction**
- **Customizing Drawing**

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**.

For information about interactions in other areas, refer to:

- **Allowing the User to Automatically Sort Rows**
- **Managing Keyboard Interaction**
- **Managing Printing**

## Customizing Interaction in the Overall Component

You can customize several aspects of the user interface to customize user interaction with the Spread component. You can also customize how user interaction is handled. For more information, see the following topics concerning these two areas of customization:

User Interface

- **Customizing the Scroll Bars of the Component**
- **Customizing Scroll Bar Tips**
- **Customizing the Sheet Name Tabs of the Component**
- **Customizing the User Interface Images**
- **Allowing the User to Zoom the Display of the Component**
- **Customizing the Scale Mode**
- **Adding a Context Menu to a Component**
- **Adding a Status Bar (on-line documentation)**
- **Hosting the Component on a Web Page**

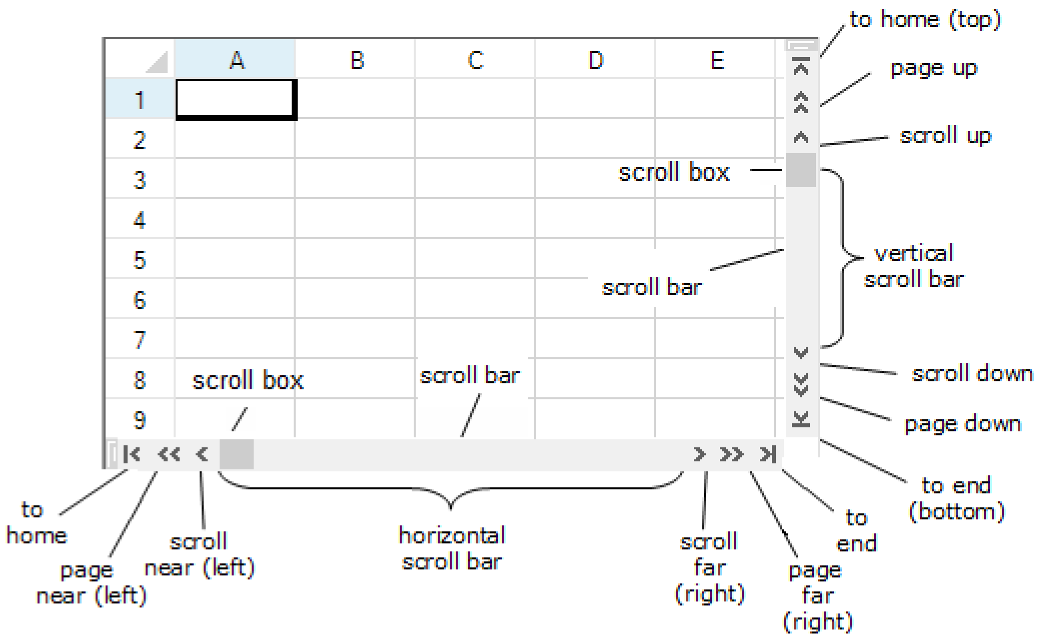
Interaction Handling

- **Customizing Clipboard Operation Options**
- **Customizing Undo and Redo Actions**
- **Locating the Pointer Using HitTest**
- **Customizing Interaction Based on Events**
- **Handling Events of Subeditors**
- **Customizing the User Error Messages**

You can also supply the user with a Search dialog. For more information, see **Allowing the User to Perform a Standard Search**.

## Customizing the Scroll Bars of the Component

You can customize the display and operation of the scroll bars on the spreadsheet. There are several aspects of the display of scroll bars that you can specify. The parts of the scroll bars are shown in the following figure.



One way to customize the scroll bar display and operation is to use the **EnhancedScrollBarRenderer ('EnhancedScrollBarRenderer Class' in the on-line documentation)** class members.

Another way is to set these customizations to the scroll bars for the entire component using the properties in the **FpSpread ('FpSpread Class' in the on-line documentation)** class. 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 ('FpSpread Class' in the on-line documentation)** class.

| Customization | Property in FpSpread |
|---|---|
| When to display either a vertical or horizontal scroll bar or both on the edges of the sheet in the component | **HorizontalScrollBarPolicy ('HorizontalScrollBarPolicy Property' in the on-line documentation)**<br>**VerticalScrollBarPolicy ('VerticalScrollBarPolicy Property' in the on-line documentation)** |
| Dimensions of the scroll bars | **HorizontalScrollBarHeight ('HorizontalScrollBarHeight Property' in the on-line documentation)**<br>**VerticalScrollBarWidth ('VerticalScrollBarWidth Property' in the on-line documentation)** |
| Whether the spreadsheet scrolls across the display when the user moves the scroll box (tracking) | **ScrollBarTrackPolicy ('ScrollBarTrackPolicy Property' in the on-line documentation)** |
| Whether scroll bars are based on only the area that has data or on the entire spreadsheet | **ScrollBarShowMax ('ScrollBarShowMax Property' in the on-line documentation)** |
| Whether to align the scroll bars with the last row and column | **ScrollBarMaxAlign ('ScrollBarMaxAlign Property' in the on-line documentation)** |

| | |
|---|---|
| Whether scroll bar tips are displayed | See **Customizing Scroll Bar Tips** |

If you want to set the width of a horizontal scroll bar, you can change the tab strip ratio, which determines the width of the tab strip, but also determines the width of the scroll bar. By default, the **TabStripRatio ('TabStripRatio Property' in the on-line documentation)** property is set to 0.50 (area is divided 50% tab strip and 50% scroll bar). 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. For more information on customizing the tab strip, refer to **Customizing the Sheet Name Tabs of the Component**.

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.

The scroll bars are controls and so inherit the benefits of those controls. For example, there are context menus available on both the horizontal and vertical scroll bars that are available by default.

The default scroll bars do not display the page, home, or end arrows.

### Smooth Scrolling

You can provide smooth scrolling with the **VerticalScrollBarMode ('VerticalScrollBarMode Property' in the on-line documentation)** and **HorizontalScrollBarMode ('HorizontalScrollBarMode Property' in the on-line documentation)** properties.

### Using the Properties Window

1. At design time, in the **Properties** window, select the Spread component.
2. Select (in the **Behavior** category) each of the scroll bar properties and set the values accordingly.
3. Repeat for each property.

### Using Code

To set the scroll bars for the component, set the **FpSpread ('FpSpread Class' in the on-line documentation)** component scroll bar properties.

### Example

This example creates a new Spread control on the form and specifies several aspects of the scroll bars for the component. 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;
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;
```

**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 = 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
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
```

**Using Code**

To display all the scroll bar buttons, set the **FpSpread ('FpSpread Class' in the on-line documentation)** component scroll bar properties.

**Example**

This example shows 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;
```

**VB**

```
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**.

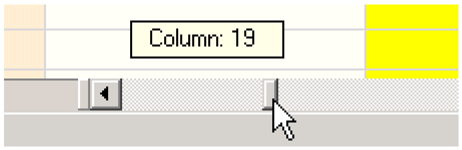
2. In the **Scroll Bar** tab, set the display and tracking by selecting the options.
3. Click **OK**.
4. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

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.

## Customizing Scroll Bar Tips

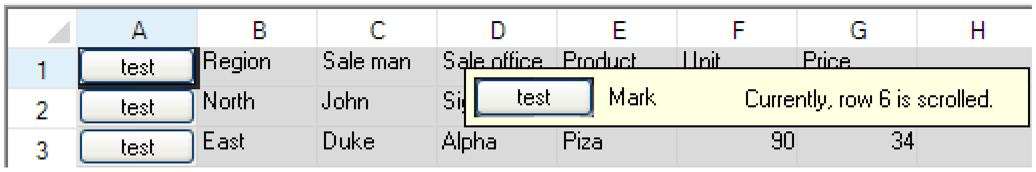
As an additional aid to the end users, you can turn on scroll bar tips which, by default, display the row number when the pointer is over the vertical scroll bar and the column number when the pointer is over the horizontal scroll bar. In the following figure, the scroll bar tip shows the column number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



In the example code in the topic **Customizing the Scroll Bars of the Component**, the scroll bar tips are set for the vertical scrolling but turned off for horizontal scrolling by using the **ScrollTipPolicy ('ScrollTipPolicy Property' in the on-line documentation)** property in the **FpSpread ('FpSpread Class' in the on-line documentation)** class.

You can specify how the scroll bar tips are displayed for the vertical scroll bar using properties in the **ScrollingContentInfo ('ScrollingContentInfo Class' in the on-line documentation)** class. This allows you to specify the location of the index numbers and the maximum height for the vertical scroll bar tips.

You can also show image or button cells in the scroll bar tips. If you set the column indices to columns with button or image cells, then the cells will be displayed in the vertical scroll tip. The following image shows a vertical scroll tip with a button cell. The extra text in the scroll tip is added using the **ScrollTipFetch ('ScrollTipFetch Event' in the on-line documentation)** event. The column index of zero causes the button cell to be displayed and the column index of two causes the name to be displayed in the scroll tip.



| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 1 | test | Region | Sale man | Sale office | Product | Unit | Price | |
| 2 | test | North | John | Si | | | | |
| 3 | test | East | Duke | Alpha | Piza | 90 | 34 | |

test Mark Currently, row 6 is scrolled.

You can also customize the text for the scroll bar text tip by using the **TipText ('TipText Property' in the on-line documentation)** property of the **ScrollTipFetchEventArgs ('ScrollTipFetchEventArgs Class' in the on-line documentation)** (for the **ScrollTipFetch** event). This event can be used with the settings in the **ScrollingContentInfo ('ScrollingContentInfo Class' in the on-line documentation)** class to combine the column content with the text set in the event (as illustrated in the above image).

**Example**

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

**C#**

```
private void Form1_Load(object sender, System.EventArgs e)
{
  // Display pop-ups when scrolled horizontally/vertically.
  fpSpread1.ScrollTipPolicy = FarPoint.Win.Spread.ScrollTipPolicy.Both;
  // Scroll sheets all together.
  fpSpread1.ScrollBarTrackPolicy = FarPoint.Win.Spread.ScrollBarTrackPolicy.Both;
}

private void fpSpread1_ScrollTipFetch(object sender,
FarPoint.Win.Spread.ScrollTipFetchEventArgs e)
{
  if (e.Column == -1)
  {
    // Customize text to be displayed.
    e.TipText = "Currently, row " + e.Row.ToString() + " is scrolled.";
  }
}
```

**VB**

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
MyBase.Load
  ' Display pop-ups when scrolled horizontally/vertically.
  FpSpread1.ScrollTipPolicy = FarPoint.Win.Spread.ScrollTipPolicy.Both
  ' Scroll sheets all together.
  FpSpread1.ScrollBarTrackPolicy = FarPoint.Win.Spread.ScrollBarTrackPolicy.Both
End Sub

Private Sub FpSpread1_ScrollTipFetch(ByVal sender As Object, ByVal e As
FarPoint.Win.Spread.ScrollTipFetchEventArgs) Handles FpSpread1.ScrollTipFetch
  If e.Column = -1 Then
    ' Customize text to be displayed.
    e.TipText = "Currently, row " + e.Row.ToString + " is scrolled."
  End If
End Sub
```

**Example**

In this example, the scroll bar tip is set using the **ScrollingContentInfo** class. You can also use the **ScrollTipFetch** event code in the previous sample with this sample.

**C#**

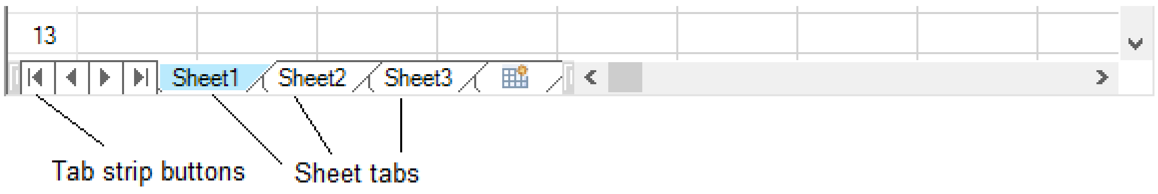
```
fpSpread1.ScrollTipPolicy = FarPoint.Win.Spread.ScrollTipPolicy.Both;
FarPoint.Win.Spread.ScrollingContentInfo scrollingContentInfo = new
FarPoint.Win.Spread.ScrollingContentInfo();
scrollingContentInfo.ColumnIndices = "0,2";
scrollingContentInfo.MaxHeight = 100;
scrollingContentInfo.RowNumberPolicy =
FarPoint.Win.Spread.ScrollingContentRowNumberPolicy.Last;
fpSpread1.Sheets[0].ScrollingContentInfo = scrollingContentInfo;
// The following code creates button cells for the text tip
// FarPoint.Win.Spread.CellType.ButtonCellType btest = new
FarPoint.Win.Spread.CellType.ButtonCellType();
// btest.Text = "test";
// fpSpread1.Sheets[0].Columns[0].CellType = btest;
```

**VB**

```
FpSpread1.ScrollTipPolicy = FarPoint.Win.Spread.ScrollTipPolicy.Both
Dim scrollingContentInfo As New FarPoint.Win.Spread.ScrollingContentInfo()
scrollingContentInfo.ColumnIndices = "0,2"
scrollingContentInfo.MaxHeight = 100
scrollingContentInfo.RowNumberPolicy =
FarPoint.Win.Spread.ScrollingContentRowNumberPolicy.Last
FpSpread1.Sheets(0).ScrollingContentInfo = scrollingContentInfo
' The following code creates button cells for the text tip
' Dim btest As New FarPoint.Win.Spread.CellType.ButtonCellType
' btest.Text = "test"
' FpSpread1.Sheets(0).Columns(0).CellType = btest
```

## Customizing the Sheet Name Tabs of the Component

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. If you add additional sheets, the tab strip by default is then displayed showing the sheet name tabs.

You can customize the following features for the tab strip:

- Display
- Appearance
- Placement
- Width
- Pointer display
- First tab displayed
- Events

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 (on-line documentation)** in the Spread Designer Guide.

For general information, refer to these classes:

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

**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:

- FpSpread class, **TabStripPolicy ('TabStripPolicy Property' in the on-line documentation)** property
- FpSpread class, **TabStrip ('TabStrip Property' in the on-line documentation)** property
- TabStrip class, **ButtonPolicy ('ButtonPolicy Property' in the on-line documentation)**

- **TabStripButtonPolicy ('TabStripButtonPolicy Enumeration' in the on-line documentation)** enumeration

**Note:** 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, 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.

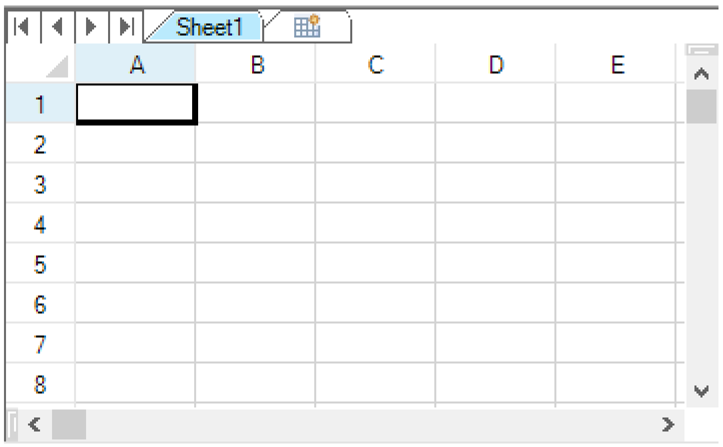
You can set properties for the tab strip such as the background color (set the InterfaceRenderer to Nothing or null and visual styles to Off) and text font for the sheet tabs. The default sheet names are Sheet1, Sheet2, and so on. You can specify other names for the sheets and these appear in the sheet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

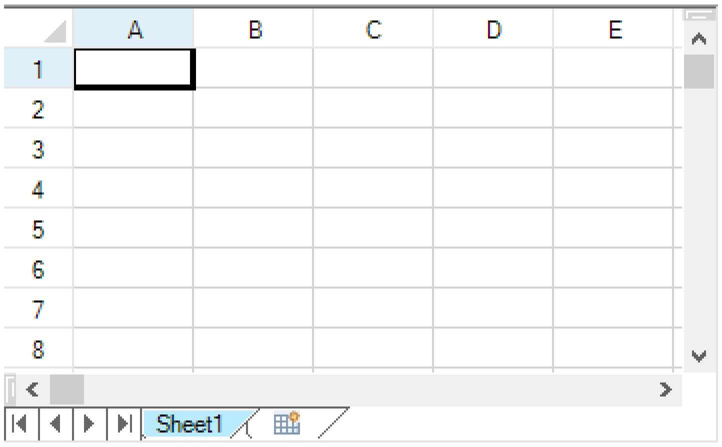
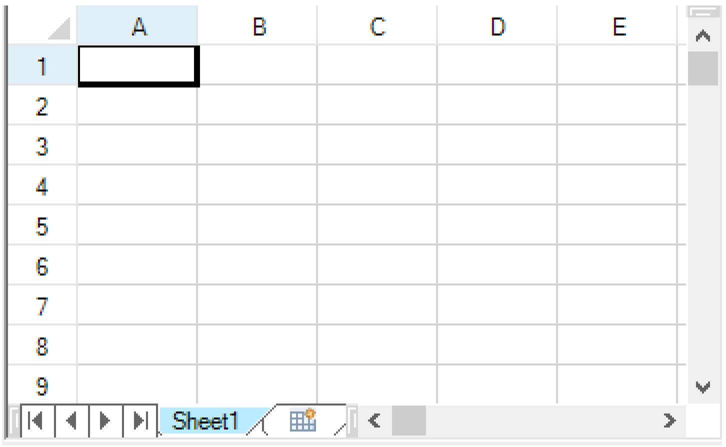
You can customize the name of each sheet. Use the **SheetName ('SheetName Property' in the on-line documentation)** property in the **SheetView ('SheetView Class' in the on-line documentation)** class. For more information, refer to these members:

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

**Tab Strip Placement**

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

| Placement Value | Sample Showing Placement |
|---|---|
| Top (at the top of the component above the headers) |  |

| | |
|---|---|
| Bottom (under the scroll bar at the bottom of the component) |  |
| WithScrollBar (alongside the scroll bar at the bottom of the component) |  |

If the tab strip is placed at the top of the sheet, and the grouping display is turned on, the tab strip appears below the group bar but above the column headers and the rest of the sheet.

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

**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, 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, refer to **Customizing the Scroll Bars of the Component**.

**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 ('FpSpread Class' in the on-line documentation)** class.

### Tab Strip Events

You can work with the following events and event handlers.

- FpSpread Class, **OnSheetTabClick ('OnSheetTabClick Method' in the on-line documentation)** Method and **OnSheetTabDoubleClick ('OnSheetTabDoubleClick Method' in the on-line documentation)** Method
- **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 **e.SheetTabIndex** event parameter returns the tab that was clicked on.

To display the tab name you can use a message box as shown in this code:

```
MsgBox(FpSpread1.Sheets(e.SheetTabIndex).SheetName)
```

### Using the Properties Window

1. In the **Properties** window, select the Spread component.
2. Specify the width of the tab strip by setting the **TabStripRatio** property.
3. Specify when sheet tabs are displayed by setting the **TabStripPolicy** property.
4. Specify the location of the tab strip by setting the **TabStripPlacement** property.
5. Specify when the buttons are displayed by clicking the **TabStrip** property, then setting the **ButtonPolicy** property.
6. Specify the background color for the sheet tabs by clicking the **TabStrip** property, then setting the **BackColor** property.

### Using a Shortcut

1. Specify the width of the tab strip by setting **FpSpread ('FpSpread Class' in the on-line documentation)** class **TabStripRatio ('TabStripRatio Property' in the on-line documentation)** property.
2. Specify when sheet tabs are displayed by setting **FpSpread ('FpSpread Class' in the on-line documentation)** class **TabStripPolicy ('TabStripPolicy Property' in the on-line documentation)** property.
3. Specify where the tab strip is displayed by setting the **FpSpread ('FpSpread Class' in the on-line documentation)** class **TabStripPlacement ('TabStripPlacement Property' in the on-line documentation)** property.
4. Specify when the buttons are displayed by setting **TabStrip ('TabStrip Class' in the on-line documentation)** class **ButtonPolicy ('ButtonPolicy Property' in the on-line documentation)**

property.

5. Specify the background color for the sheet tabs by setting **TabStrip ('TabStrip Class' in the on-line documentation)** class **BackColor ('BackColor Property' in the on-line documentation)** property.

**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;
```

**VB**

```
' 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 Code**

1. Specify the width of the tab strip by setting the **FpSpread ('FpSpread Class' in the on-line documentation)** class **TabStripRatio ('TabStripRatio Property' in the on-line documentation)** property.
2. Specify when sheet tabs are displayed by setting the **FpSpread ('FpSpread Class' in the on-line documentation)** class **TabStripPolicy ('TabStripPolicy Property' in the on-line documentation)** property.
3. Create a new **TabStrip ('TabStrip Class' in the on-line documentation)** object, and set its value equal to the **FpSpread ('FpSpread Class' in the on-line documentation)** object **TabStrip ('TabStrip Property' in the on-line documentation)** property.
4. Set the**TabStrip ('TabStrip Class' in the on-line documentation)** object **ButtonPolicy ('ButtonPolicy Property' in the on-line documentation)** property to specify when the buttons are displayed, and set its **BackColor ('BackColor Property' in the on-line documentation)** property to specify the background color.

**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;
// Create new tab strip.
FarPoint.Win.Spread.TabStrip tstrip;
tstrip = fpSpread1.TabStrip;
// Display the tab strip buttons as needed.
tstrip.ButtonPolicy = FarPoint.Win.Spread.TabStripButtonPolicy.AsNeeded;
// Set the background color.
tstrip.BackColor = Color.Bisque;
fpSpread1.InterfaceRenderer = null;
```

**VB**

```
' Set the sheet tabs to always appear.
FpSpread1.TabStripPolicy = FarPoint.Win.Spread.TabStripPolicy.Always
' Set the width to 60%.
FpSpread1.TabStripRatio = 0.60
' Create new tab strip.
Dim tstrip As New FarPoint.Win.Spread.TabStrip()
tstrip = FpSpread1.TabStrip
' Display the tab strip buttons as needed.
tstrip.ButtonPolicy = FarPoint.Win.Spread.TabStripButtonPolicy.AsNeeded
' Set the background color.
tstrip.BackColor = Color.Bisque
FpSpread1.InterfaceRenderer = Nothing
```

**Using the Spread Designer**

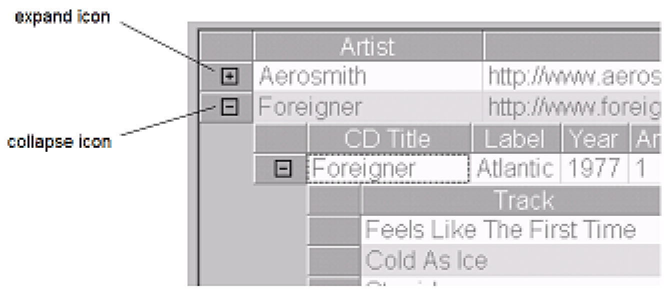
1. From the **Settings** menu, choose **Tab Strip** (**Appearance** section).
2. Under **TabStripPolicy**, select when you want the sheet tabs to be displayed or select **Never** to hide the sheet tabs.
   No matter which item you select, inside Spread Designer the sheet tabs are always displayed to 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.

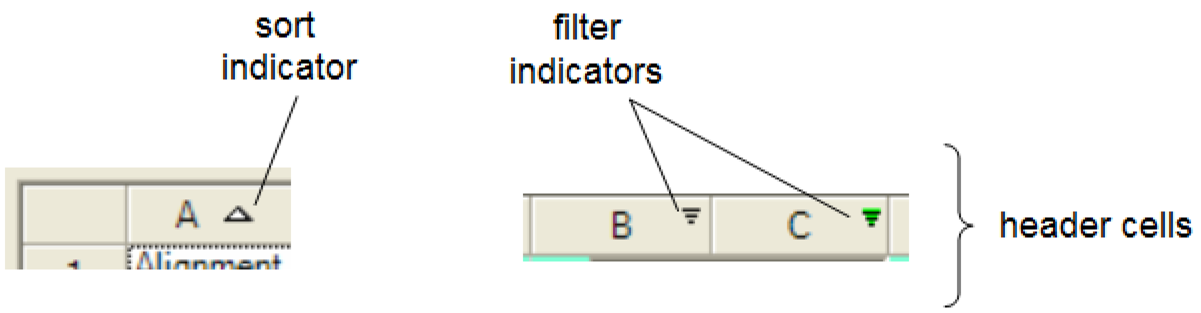
## Customizing the User Interface Images

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 **CellType ColumnHeaderRenderer ('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**.

## 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. Use the **AllowUserZoom ('AllowUserZoom Property' in the on-line documentation)** property of the **FpSpread ('FpSpread Class' in the on-line documentation)** class. This 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.

## Customizing the Scale Mode

You can provide support for the 120 dpi scale mode in Spread with the **SpreadScaleMode ('SpreadScaleMode Property' in the on-line documentation)** property. Spread supports resizing rows, columns, cells, and data based on the dpi.

The automatic scaling only occurs when the form is loaded. The scaling can change at run time if there are changes to the control size and location and the container layout is suspended.

For the best results, set the **AutoScaleMode** property of the container to **Dpi** when using the **ZoomDpiSupport** option of the **SpreadScaleMode ('SpreadScaleMode Property' in the on-line documentation)** property.

**Using Code**

1. Suspend the layout.
2. Set the location and size for the control.
3. Set the **SpreadScaleMode ('SpreadScaleMode Property' in the on-line documentation)** property to **ZoomDpiSupport**.
4. Use the **ResumeLayout ('ResumeLayout Method' in the on-line documentation)** method to see the changes.

**Example**

This example sets the **SpreadScaleMode** property to **ZoomDpiSupport** and suspends and resumes the layout.

**C#**

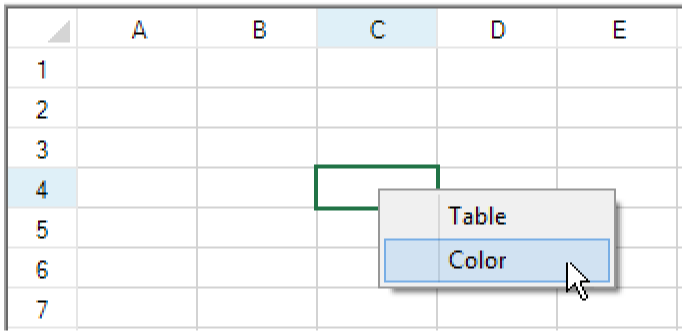
```
fpSpread.SuspendLayout();

AutoScaleDimensions = new System.Drawing.SizeF(96.0F, 96.0F);
AutoScaleMode = System.Windows.Forms.AutoScaleMode.Dpi;
fpSpread1.Location = new System.Drawing.Point(23, 86);
fpSpread1.Size = new System.Drawing.Size(356, 161);
fpSpread1.SpreadScaleMode = FarPoint.Win.Spread.ScaleMode.ZoomDpiSupport;
fpSpread1.ResumeLayout();
```

**VB**

```
FpSpread1.SuspendLayout()

AutoScaleDimensions = New System.Drawing.SizeF(96.0F, 96.0F)
AutoScaleMode = System.Windows.Forms.AutoScaleMode.Dpi
FpSpread1.Location = New System.Drawing.Point(23, 86)
FpSpread1.Size = New System.Drawing.Size(356, 161)
FpSpread1.SpreadScaleMode = FarPoint.Win.Spread.ScaleMode.ZoomDpiSupport
FpSpread1.ResumeLayout()
```

## 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
```

## Hosting the Component on a Web Page

If you are hosting the Spread Windows Forms component as a user control on a Web page in Microsoft Internet Explorer (IE), you should make security permission adjustments by setting the level of trust.

**For Microsoft IE**

- In Microsoft 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).
- In Microsoft 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**.

## 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.

**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)**

## 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**.

**Action Assignment**

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.

## 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 | Class Name |
|---|---|

| Area | |
|---|---|
| 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 |

## 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.

## Handling Events of Subeditors

You can handle events of subeditors within a cell in the Spread component.

The subeditor has a **SubEditorClosed ('SubEditorClosed Event' in the on-line documentation)** event and a **SubEditorOpening ('SubEditorOpening Event' in the on-line documentation)** event. This allows you to know when the subeditor opens and closes.

The cell type brings up an editor when editing the cell. The editor control can be text based or graphics based. The editor control can drop-down lists, bring up pop-up dialogs, and so on. The drop-down list or pop-up dialog is known as the subeditor. The **ISubEditor ('ISubEditor Interface' in the on-line documentation)** interface can be used to create a custom subeditor. The **IEditor ('IEditor Interface' in the on-line documentation)** interface can be used to create a custom editor.

Existing editors can be used in cells. The following code uses the default date time cell formatting on cell (1,1):

### C#

```
fpSpread1.Sheets[0].Cells[1, 1].Editor = new
FarPoint.Win.Spread.CellType.DateTimeCellType();
```

### Visual Basic

```
FpSpread1.Sheets(0).Cells(1,1).Editor = New
FarPoint.Win.Spread.CellType.DateTimeCellType
```

## 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.

## Customizing Interaction with a Sheet

You can customize aspects of the view of an individual sheet that provide ways for the user to interact with the sheet. To customize this aspect of user interaction, you can perform any of the following tasks:

- **Customizing Viewports**
- **Customizing Split Boxes**
- **Customizing the Position in the Display**
- **Placing Child Controls on a Sheet**
- **Creating Tables**

You can also allow aspects of the user interaction that are described elsewhere in the documentation, including the following features that work with an entire sheet:

- **Customizing the Sheet Name Tabs of the Component**
- **Rearranging Data on a Sheet**
- **Customizing User Searching of Data**
- **Working with Hierarchical Data Display**
- **Customizing Drawing**
- **Locking a Cell**
- **Customizing Interaction in Cells**
- **Customizing the Individual Sheet Appearance**

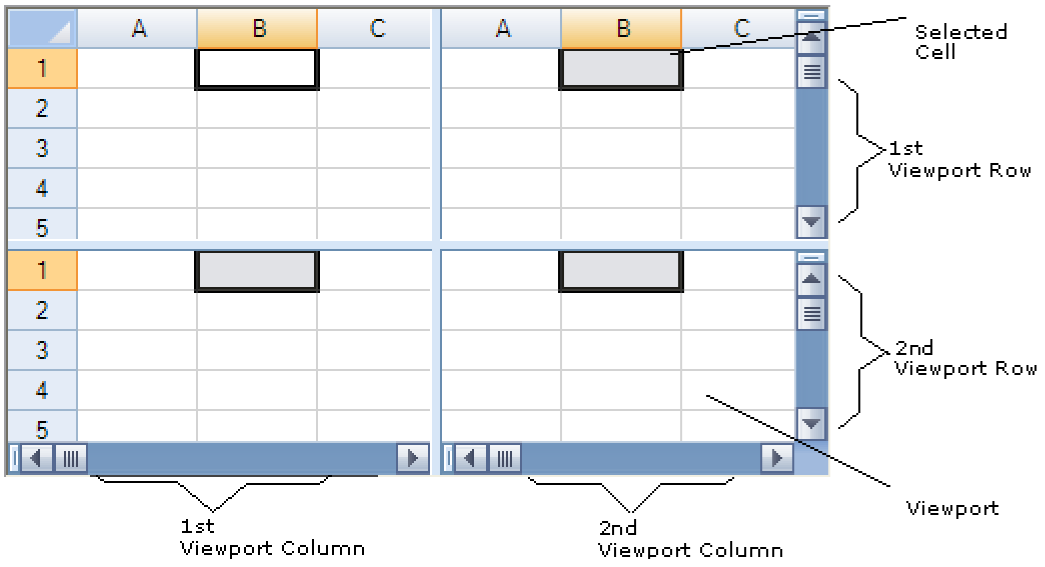
## Customizing Viewports

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

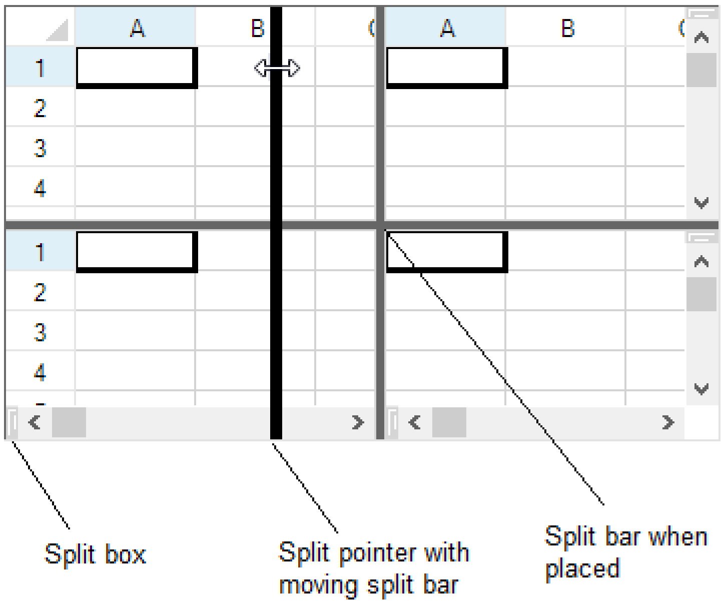
- A set of horizontal viewports (called a viewport row since it is a row of viewports)

- 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 split bars show the border of each viewport. Each viewport row or viewport column has its own scroll bars.

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 of the Component**.

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 | Method or Property in FpSpread (or SpreadView) class |
|---|---|
| Add and remove viewports | **AddViewport ('AddViewport Method' in the on-line documentation)**<br>**RemoveViewport ('RemoveViewport Method' in the on-line documentation)** |
| Set the height and width of viewports | **SetViewportPreferredHeight ('SetViewportPreferredHeight Method' in the on-line documentation)**<br>**SetViewportPreferredWidth ('SetViewportPreferredWidth Method' in the on-line documentation)**<br>**GetViewportPreferredHeight ('GetViewportPreferredHeight Method' in the on-line documentation)**<br>**GetViewportPreferredWidth ('GetViewportPreferredWidth Method' in the on-line documentation)** |
| Determine the viewport for a given location in the display | **GetViewPortX ('GetViewportX Method' in the on-line documentation)**<br>**GetViewPortY ('GetViewportY Method' in the on-line documentation)**<br>**GetViewPortHeight ('GetViewportHeight Method' in the on-line documentation)**<br>**GetViewPortWidth ('GetViewportWidth Method' in the on-line documentation)** |
| Determine the row or column of cells in a particular viewport row or viewport column | **GetViewportBottomRow ('GetViewportBottomRow Method' in the on-line documentation)**<br>**GetViewportLeftColumn ('GetViewportLeftColumn Method' in the on-line documentation)**<br>**GetViewportRectangle ('GetViewportRectangle Method' in the on-line documentation)**<br>**GetViewportTopRow ('GetViewportTopRow Method' in the on-line documentation)**<br>**SetViewportLeftColumn ('SetViewportLeftColumn Method' in the on-line documentation)**<br>**SetViewportTopRow ('SetViewportTopRow Method' in the on-line documentation)** |

| | |
|---|---|
| Determine which is the active viewport | **GetActiveColumnViewportIndex ('GetActiveColumnViewportIndex Method' in the on-line documentation)**<br>**GetActiveRowViewportIndex ('GetActiveRowViewportIndex Method' in the on-line documentation)**<br>**SetActiveViewport ('SetActiveViewport Method' in the on-line documentation)** |
| Determine the index of a particular viewport row or viewport column | **GetColumnViewportIndexFromX ('GetColumnViewportIndexFromX Method' in the on-line documentation)**<br>**GetRowViewportIndexFromY ('GetRowViewportIndexFromY Method' in the on-line documentation)** |
| Determine the number of viewports in either orientation | **GetColumnViewportCount ('GetColumnViewportCount Method' in the on-line documentation)**<br>**GetRowViewportCount ('GetRowViewportCount Method' in the on-line documentation)** |
| Position the viewport in the display | **ShowColumn ('ShowColumn Method' in the on-line documentation)**<br>**ShowRow ('ShowRow Method' in the on-line documentation)** |

The preferred height and preferred width are suggested sizes. 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 preferred height and preferred width 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.

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:

| 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**.

**Using Code**

To create or customize a viewport in code, use the following add and set methods in the **FpSpread ('FpSpread Class' in the on-line documentation)** class. Refer to the preceding table for descriptions of these. Refer to the **Assembly Reference (on-line documentation)** for more details on each.

- **AddViewport ('AddViewport Method' in the on-line documentation)**
- **RemoveViewport ('RemoveViewport Method' in the on-line documentation)**
- **GetViewportPreferredWidth ('GetViewportPreferredWidth Method' in the on-line documentation)** and **SetViewportPreferredWidth ('SetViewportPreferredWidth Method' in the on-line documentation)**
- **GetViewportPreferredHeight ('GetViewportPreferredHeight Method' in the on-line documentation)** and **SetViewportPreferredHeight ('SetViewportPreferredHeight Method' in the on-line documentation)**
- **SetActiveViewport ('SetActiveViewport Method' in the on-line documentation)**

For more information, refer to methods in the **FpSpread ('FpSpread Class' in the on-line documentation)** class.

**Example**

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);
```

**VB**

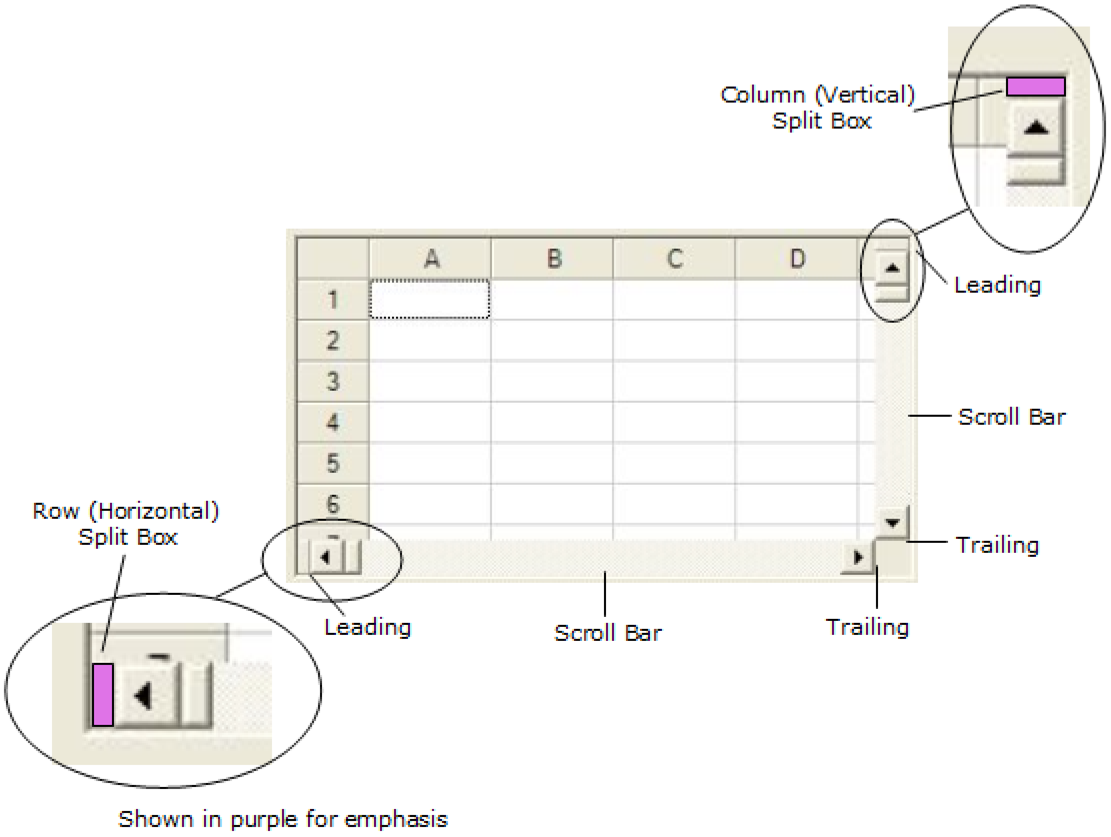
```
FpSpread1.AddViewport(1, 2)
FpSpread1.SetViewportLeftColumn(1, 3)
FpSpread1.SetViewportTopRow(0, 6)
FpSpread1.SetViewportPreferredHeight(0, 100)
FpSpread1.SetViewportPreferredWidth(0, 100)
```

**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

You can determine the display and placement of the 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. You can change these by setting a property or by using Spread Designer.


Column (Vertical) Split Box
Leading
Scroll Bar
Trailing
Row (Horizontal) Split Box
Leading
Scroll Bar
Trailing
Shown in purple for emphasis

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**.

| Customization | Property and Enumeration |
|---|---|
| Placement of split box relative to the scroll bar | **ColumnSplitBoxAlignment ('ColumnSplitBoxAlignment Property' in the on-line documentation)** property in **FpSpread ('FpSpread Class' in the on-line documentation)** class<br>**RowSplitBoxAlignment ('RowSplitBoxAlignment Property' in the on-line documentation)** property in **FpSpread ('FpSpread Class' in the on-line documentation)** class<br>**SplitBoxAlignment ('SplitBoxAlignment Enumeration' in the on-line documentation)** enumeration |
| Whether or when to display the split box | **ColumnSplitBoxPolicy ('ColumnSplitBoxPolicy Property' in the on-line documentation)** property in **FpSpread ('FpSpread Class' in the on-line documentation)** class<br>**RowSplitBoxPolicy ('RowSplitBoxPolicy Property' in the on-line documentation)** property in **FpSpread ('FpSpread Class' in the on-line documentation)** class<br>**SplitBoxPolicy ('SplitBoxPolicy Enumeration' in the on-line documentation)** enumeration |

**Using the Properties Window**

1. At design time, in the **Properties** window, select the Spread component.

2. Select (in the **Splitters** category) 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.

**Using Code**

Set the **ColumnSplitBoxAlignment ('ColumnSplitBoxAlignment Property' in the on-line documentation)**, **RowSplitBoxAlignment ('RowSplitBoxAlignment Property' in the on-line documentation)**, **ColumnSplitBoxPolicy ('ColumnSplitBoxPolicy Property' in the on-line documentation)**, and **RowSplitBoxPolicy ('RowSplitBoxPolicy Property' in the on-line documentation)** properties for the row or column of viewports for the **FpSpread ('FpSpread Class' in the on-line documentation)** component.

**Example**

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;
```

**VB**

```
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.

## 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**

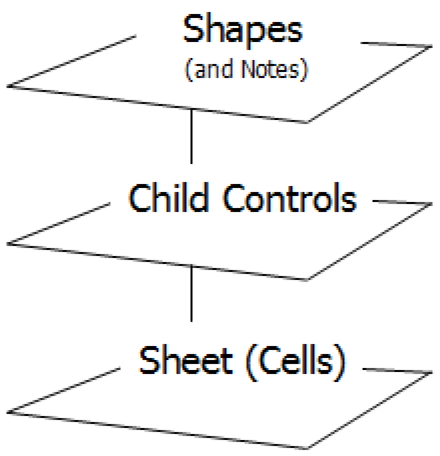
```
FpSpread1.Sheets(0).SetActiveCell(3,4)
FpSpread1.ShowActiveCell(FarPoint.Win.Spread.VerticalPosition.Top,
FarPoint.Win.Spread.HorizontalPosition.Center)
```

## 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 IEmbeddedControl 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 **Activation** 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 policy 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:

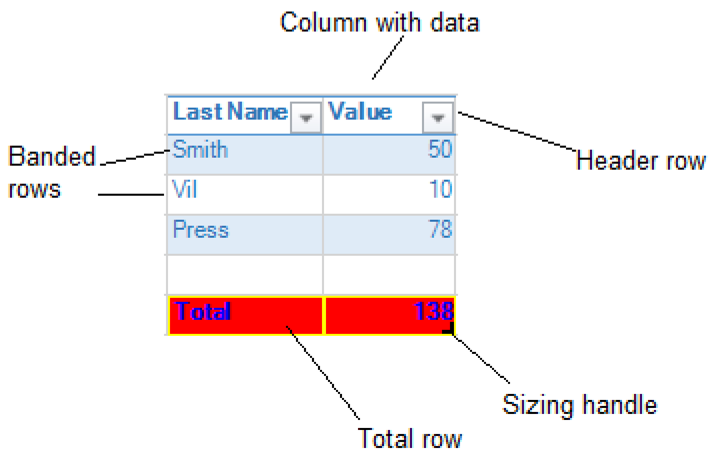
- FpSpread.**ChildControlActivated ('ChildControlActivated Event' in the on-line documentation)**
- FpSpread.**ChildControlDeactivated ('ChildControlDeactivated Event' in the on-line documentation)**
- FpSpread.**AddControl ('AddControl Method' in the on-line documentation)**
- FpSpread.**AllowChildControlDesign ('AllowChildControlDesign Property' in the on-line documentation)**
- FpSpread.**RemoveControl ('RemoveControl Method' in the on-line documentation)**

## Creating 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.



| Last Name | Value |
|---|---|
| Smith | 50 |
| Vil | 10 |
| Press | 78 |
| | |
| Total | 138 |

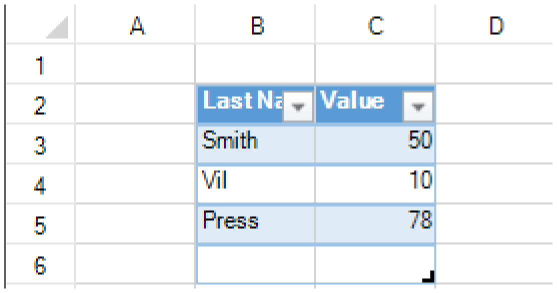
You can also move tables and create structured references in tables.

For more information, see the following topics:

- **Adding a Table**
- **Using Table Filters**
- **Resizing a Table**
- **Sorting a Table**
- **Setting Table Styles**
- **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 Na | 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)
```

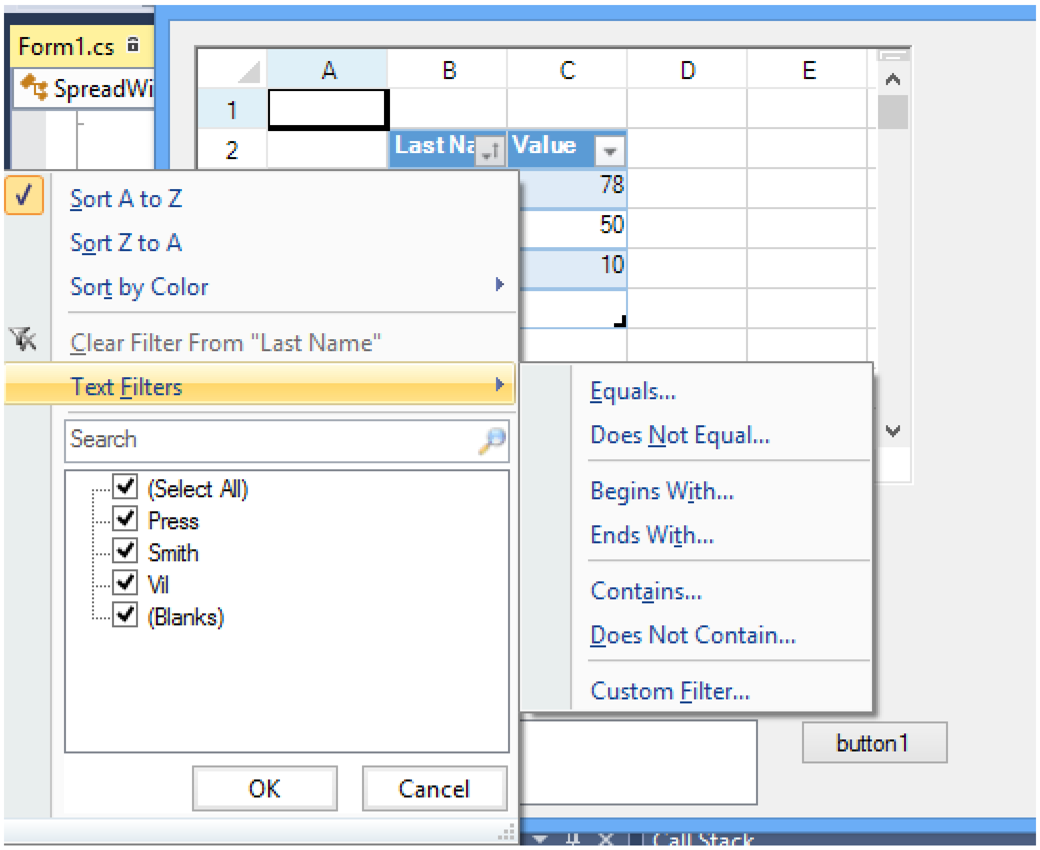
**Using the Spread Designer**

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.

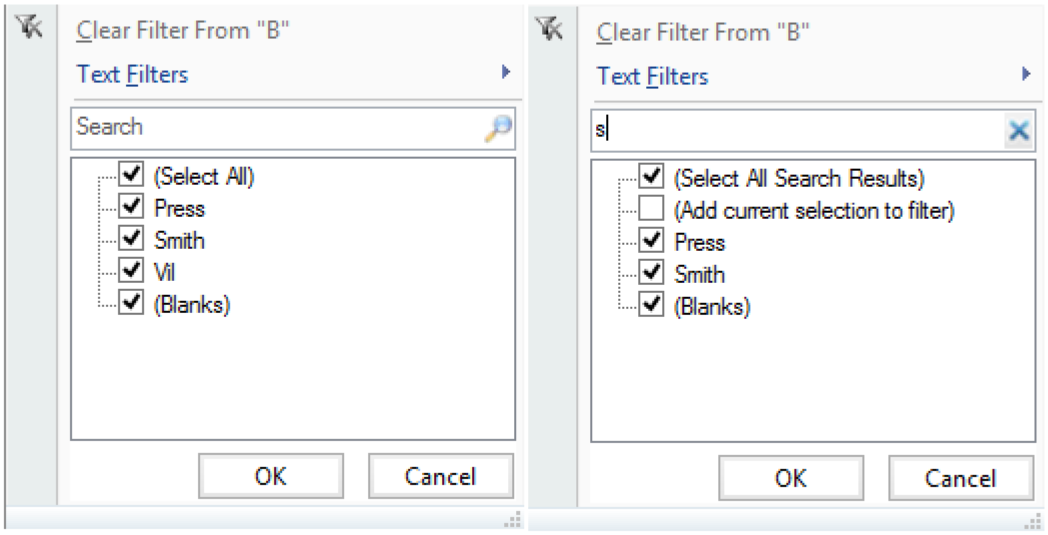
## Using Table Filters

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 |

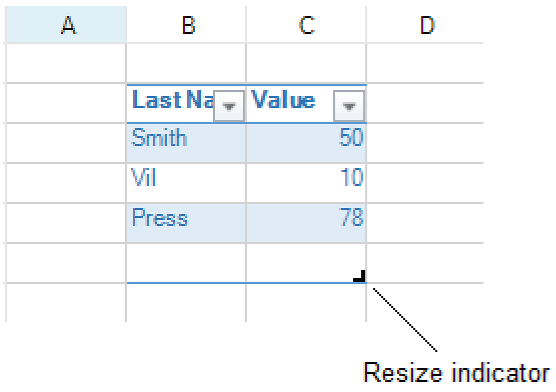
Users can specify wildcards in conditions. The "?" character represents any single character. The "*" character represents any series of characters.

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.



Select the indicator and drag to the right to add columns or down to add rows.

**Using Code**

Use the **Resize ('Resize Method' in the on-line documentation)** method to add columns or rows to a table.

**Example**

This example code adds rows and columns to the table.

**C#**

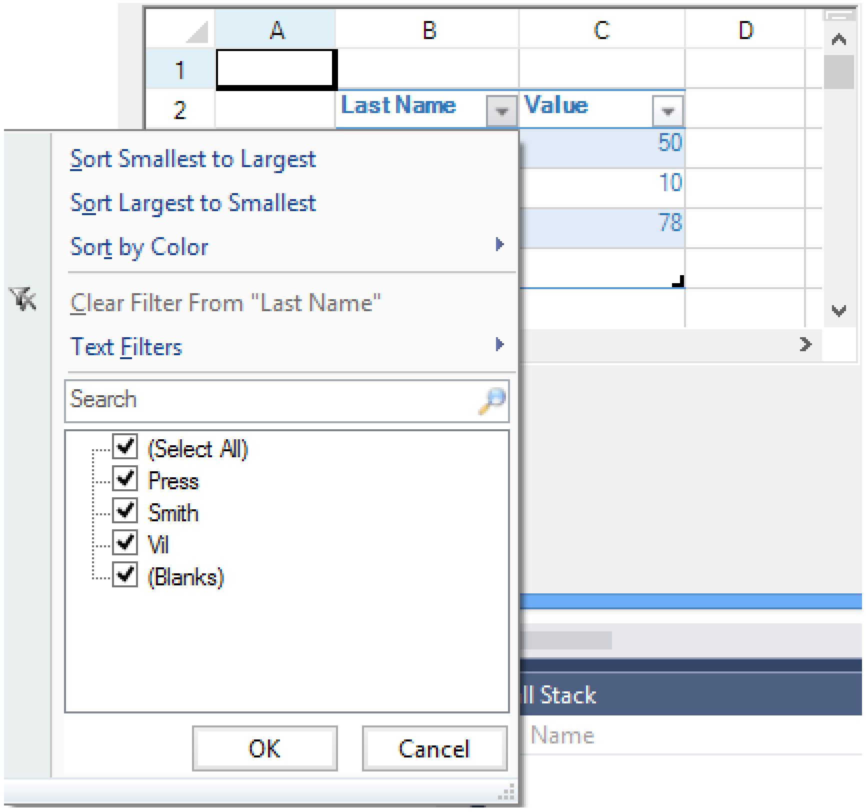
```
FarPoint.Win.Spread.TableStyle tstyle = fpSpread1.CreateTableStyle("Style1",
FarPoint.Win.Spread.TableStyle.TableStyleLight2);
fpSpread1.Sheets[0].Cells[1, 1].Text = "Last Name";
fpSpread1.Sheets[0].Cells[1, 2].Text = "Value";
fpSpread1.Sheets[0].Cells[2, 1].Text = "Smith";
fpSpread1.Sheets[0].Cells[2, 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, 3);
```

**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")
table.Resize(6, 3)
```

## 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#**

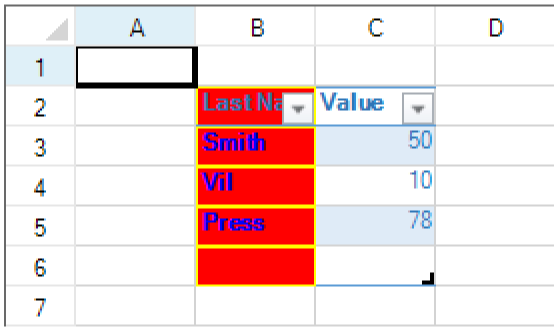
```
FarPoint.Win.Spread.TableStyle tstyle = fpSpread1.CreateTableStyle("Style1",
FarPoint.Win.Spread.TableStyle.TableStyleLight2);
fpSpread1.Sheets[0].Cells[1, 1].Text = "Last Name";
fpSpread1.Sheets[0].Cells[1, 2].Text = "Value";
fpSpread1.Sheets[0].Cells[2, 1].Text = "Smith";
fpSpread1.Sheets[0].Cells[2, 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 Na | 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**

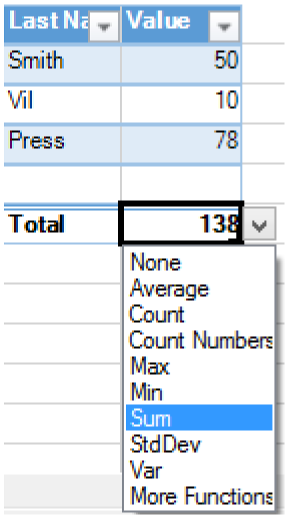
```
Dim bside As New FarPoint.Win.ComplexBorderSide(Color.Yellow)
Dim tborder As New FarPoint.Win.Spread.TableBorder(bside)
Dim testyle As New FarPoint.Win.Spread.TableElementStyle(tborder, Color.Red,
Color.Blue, FarPoint.Win.Spread.RegularBoldItalicFontStyle.Bold)
Dim tstyle As FarPoint.Win.Spread.TableStyle
tstyle = FpSpread1.CreateTableStyle("Style1",
FarPoint.Win.Spread.TableStyle.TableStyleLight2)
tstyle.FirstColumn = testyle
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.FirstColumn = True
```

## Adding a Table Formula

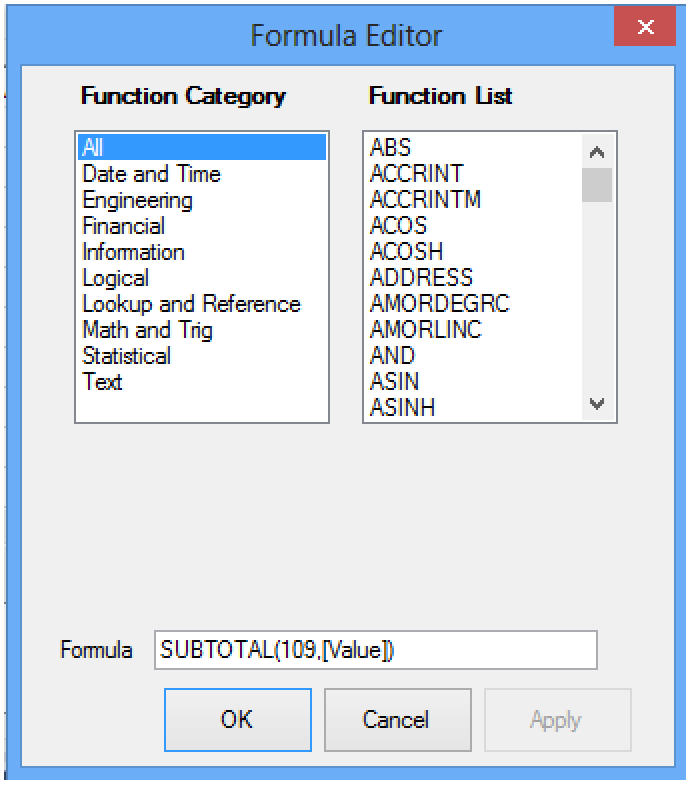
You can add formulas to the table in a total row at run time with the **Formula Editor** or with code.

Add a total row and then select the drop-down arrow at the bottom right corner of the table to display formulas.



| Last Na | Value |
|---|---|
| Smith | 50 |
| Vil | 10 |
| Press | 78 |
| | |
| Total | 138 |

You can select **More Functions** to display the **Formula Editor** as shown in the following figure.



Formula SUBTOTAL(109,[Value])

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**.

**Using Code**

Use the **TotalRowVisible ('TotalRowVisible Property' in the on-line documentation)** property to display the total row for the table.

**Example**

This 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
```

## Understanding Structured References

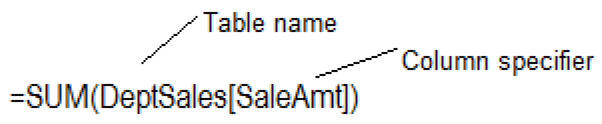
Spread supports structured reference formulas in tables. Components of the structured reference include the table name, the column specifier, the special item specifier, and the table specifier.

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 |
|---|---|---|---|
| =DeptSales[[SalesPerson]:[Region]] | All of the cells in two or more adjacent columns | : (colon) range operator | A2:B7 |
| =DeptSales[SaleAmt],DeptSales[ComAmt] | A combination of two or more columns | , (comma) union operator | C2:C7, E2:E7 |
| =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] | Multiples 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])"
```

## 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:

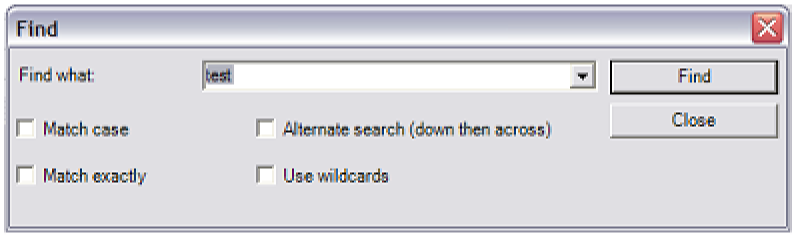
- **Allowing the User to Perform a Standard Search**
- **Allowing the User to Perform an Advanced Search**
- **Searching for Data with Code**

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. None of the information in the sheet, column, or row object is included in the search. Not all tags are included when you search and "include tags"; only cell tags are included.

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.

## 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**.

For information about performing a search without a dialog, refer to **Searching for Data with Code**.

**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#**

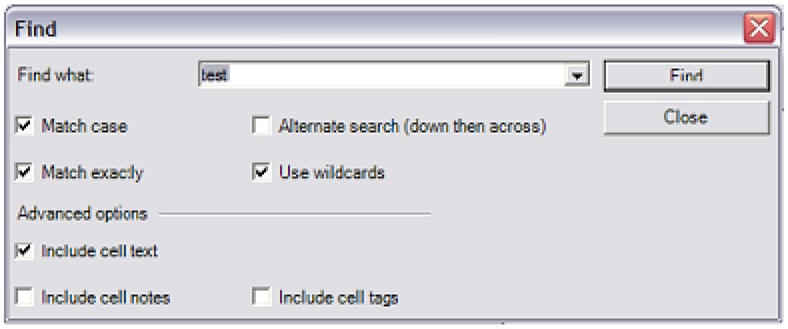
```
fpSpread1.SearchWithDialog(3,"Not Available",true,true,false,false,0,0);
```

**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. For a description of the standard search options (shown in the top half of the advanced dialog), refer to **Allowing the User to Perform a Standard 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**.

For information about performing a search without a dialog, refer to **Searching for Data with Code**.

### 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#**

```
int rowindx = 0;
int colindx = 0;
fpSpread1.Search(2,"Total",true,true,false,false,1,1,56,56, ref rowindx,ref colindx);
```

**VB**

```
Dim rowindx as Integer
Dim colindx as Integer
FpSpread1.Search(2,"Total",True,True,False,False,1,1,56,56,rowindx,colindx)
```

## Customizing User Selection 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**

You can customize several aspects of user selection and whether to hide the selection when focus is lost. For example, you can set whether to move the active cell in the view with the **MoveActiveOnFocus ('MoveActiveOnFocus Property' in the on-line documentation)** property of the **FpSpread ('FpSpread Class' in the on-line documentation)** class. For more actions with selections, refer to **Working with Selections**.

To set the selection model for a sheet, use the SheetView **Selection ('Selection Property' in the on-line documentation)** property.

For information regarding selections in Spread Designer, refer to **Selecting a Contiguous Range of Cells (on-line documentation)** in the **Spread Designer Guide (on-line documentation)**.

## 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 | FpSpread.**SelectionBlockOptions ('SelectionBlockOptions Property' in the on-line documentation)**.Cells |
| Rows | FpSpread.**SelectionBlockOptions ('SelectionBlockOptions Property' in the on-line documentation)**.Rows |
| Columns | FpSpread.**SelectionBlockOptions ('SelectionBlockOptions Property' in the on-line documentation)**.Columns |
| Sheet | FpSpread.**SelectionBlockOptions ('SelectionBlockOptions Property' in the on-line documentation)**.Sheet |
| Combination | FpSpread.**SelectionBlockOptions ('SelectionBlockOptions Property' in the on-line documentation)**.number 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 | **OperationMode ('OperationMode Property' in the on-line documentation)**.Normal with **SelectionPolicy ('SelectionPolicy Property' in the on-line documentation)** property |
| Only rows, no editing | **OperationMode ('OperationMode Property' in the on-line documentation)**.SingleSelect |
| Only rows, editing | **OperationMode ('OperationMode Property' in the on-line documentation)**.RowMode |
| Multiple contiguous rows, no editing | **OperationMode ('OperationMode Property' in the on-line documentation)**.MultiSelect |
| Multiple noncontiguous rows, no editing | **OperationMode ('OperationMode Property' in the on-line documentation)**.ExtendedSelect |

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**

1. 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.
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 **SheetView** object **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 **SheetView** object **SelectionUnit ('SelectionUnit Property' in the on-line documentation)** property to specify the unit of selection allowed.
4. 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#**

```
// 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.
FarPoint.Win.Spread.SheetView newsheet = 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;
```

**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
```

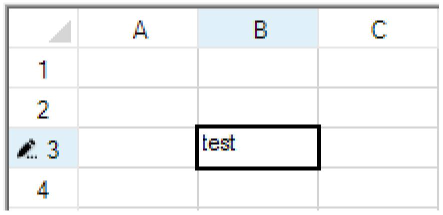
**Using the Spread Designer**

1. Select the sheet tab for the sheet for which you want to set the selection operation mode.
2. 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

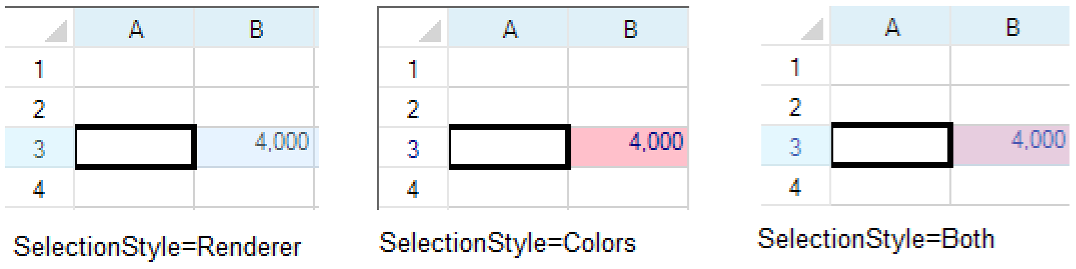
Selections have a default appearance provided by the Spread component and the selection renderer. You can change that appearance, including the background and foreground colors and font. You can also specify a row selector icon with the **ShowRowSelector ('ShowRowSelector Property' in the on-line documentation)** 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.



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.



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** | Determines the background color of selections |

| | |
|---|---|
| **Property' in the on-line documentation)** | |
| **SelectionForeColor ('SelectionForeColor Property' in the on-line documentation)** | Determines the text color of selections |
| **SelectionStyle ('SelectionStyle 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 **SelectionStyle ('SelectionStyle 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 **SelectionStyle ('SelectionStyle 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.

```
Brush selectionBrush = new SolidBrush(Color.FromArgb(100, 193, 224, 255));
```

```
g.FillRectangle(selectionBrush, x, y, width, height);
```

For more information, refer to the **SelectionStyles ('SelectionStyles 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 **SelectionStyle** property, then select one of the values from the drop-down list of values.
6. If you set the **SelectionStyle** 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 SelectionStyle ('SelectionStyle Property' in the on-line documentation)** property.
2. If you set the **SelectionStyle ('SelectionStyle 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].SelectionStyle = 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).SelectionStyle = 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 **SelectionStyle ('SelectionStyle Property' in the on-line documentation)** property for a **SheetView ('SheetView Class' in the on-line documentation)** object.
2. If you set the **SelectionStyle ('SelectionStyle 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.SelectionStyle = 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.SelectionStyle = 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
```

**Using the Spread Designer**

1. Select the sheet tab for the sheet for which you want to set the selection style.

2. In the property list, set the **SelectionStyle** property.
3. If you set the **SelectionStyle** to SelectionColors or Both, set the **SelectionBackColor** and **SelectionForeColor** properties to specify the background and text colors for the selection highlighting.
4. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

## 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);
```

**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
```

## 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**.

## Using Application Tags

You can add an application tag to a sheet, to a row or column, or to a cell in the spreadsheet.

- **Adding a Tag to a Sheet**
- **Adding a Tag to a Row or Column**
- **Adding a Tag to a Cell**

## 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.

Because 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 in the **SheetView ('SheetView 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.

## 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.

## 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."
```

## Setting and Resetting User Interaction

You can set various aspects of user interaction. If you ever need to reset the component back to its default values or clear values in order to start over, follow the topics in this section.

- **Allowing User Functionality**
- **Resetting Parts of the Interface**
- **Clearing or Removing Parts of the Interface**

## 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 data | FpSpread.**AllowDragDrop ('AllowDragDrop Property' in the on-line documentation)** property |
| Drag and fill cell data | FpSpread.**AllowDragFill ('AllowDragFill Property' in the on-line documentation)** property |
| Edit cell notes | SheetView.**AllowNoteEdit ('AllowNoteEdit Property' in the on-line documentation)** property |

| | |
|---|---|
| Enter formulas | FpSpread.**AllowUserFormulas ('AllowUserFormulas Property' in the on-line documentation)** property |
| Filter rows | Column.**AllowAutoFilter ('AllowAutoFilter Property' in the on-line documentation)** property |
| Expand or collapse hierarchy | **GetRowExpandable ('GetRowExpandable Method' in the on-line documentation)**, **SetRowExpandable ('SetRowExpandable Method' in the on-line documentation)** methods |
| Move rows and columns | FpSpread.**AllowRowMove ('AllowRowMove Property' in the on-line documentation)** property and FpSpread.**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 advanced search | FpSpread.**SearchWithDialogAdvanced ('SearchWithDialogAdvanced Method' in the on-line documentation)** method |
| Resize rows or columns | Column.**Resizable ('Resizable Property' in the on-line documentation)** property and Row.**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 |

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<br>SheetView.**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) | FarPoint.Win.SuperEditBase.**AllowClipboardKeys ('AllowClipboardKeys Property' in the on-line documentation)** property |
| Edit the sheet names | FpSpread.TabStrip.**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.**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 |

You can customize how the user interacts with the keyboard. For more information, refer to **Managing Keyboard**

**Interaction**.

You can also customize how the user interacts with printing. For more information, refer to **Managing Printing**.

For more information on shapes, refer to **Customizing Drawing**.

## 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 ('DefaultSize Property' in the on-line documentation)** 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 | Method Name |
|---|---|
| Background color for individual cells or for row or column of cells | **ResetBackColor ('ResetBackColor Method' in the on-line documentation)** |
| Border for individual cells or for row or column of cells | **ResetBorder ('ResetBorder Method' in the on-line documentation)** |
| Cell type for individual cells or for row or column of cells | **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 | Resetting Desired |
|---|---|
| **Reset ('Reset Method' in the on-line documentation)** | All the properties of the style |
| **ResetBackColor ('ResetBackColor Method' in the on-line documentation)** | Background color of a cell |
| **ResetBorder ('ResetBorder Method' in the on-line documentation)** | Border around a cell |
| **ResetCanFocus ('ResetCanFocus Method' in the on-line documentation)** | Whether the cell can receive focus |
| **ResetCellType ('ResetCellType Method' in the on-line documentation)** | Cell type of the cell |
| **ResetEditor ('ResetEditor Method' in the on-line documentation)** | Editor used for editing the cell |
| **ResetFont ('ResetFont Method' in the on-line documentation)** | 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 ('DefaultSize Property' in the on-line documentation)** 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)**.

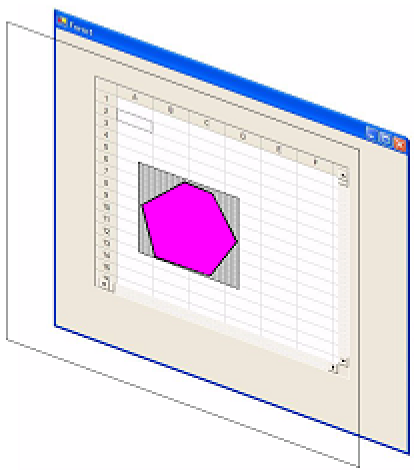
## Clearing or Removing Parts of the Interface

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.

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

## 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**
- **Working with Annotations**
- **Allowing the User to Draw with a Tablet PC**
- **Creating Camera Shapes**

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.

For more information on shapes, refer to the **Spread Designer Guide (on-line documentation)**.

## 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.

The **DynamicSize ('DynamicSize Property' in the on-line documentation)** and **DynamicMove ('DynamicMove Property' in the on-line documentation)** properties specify how shapes are moved and resized when hiding or showing, resizing, or moving rows or columns that intersect with the shape or that appear before the shape (for example, resizing a column to the left or a row above the shape).

The following options are available:

- Move and size with cells (DynamicMove = True and DynamicSize = True)
- Move, but do not size with cells (DynamicMove = True and DynamicSize = False)
- Do not move or size with cells (DynamicMove = False and DynamicSize = False)

Hiding columns and rows that contain a shape hides the shape as well.

The FarPoint.Win.Spread.DrawingSpace.**DrawingToolbar** class allows you to bring up the shape toolbar at run-time.

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.

The simplest way to add a shape can be performed in one line of code, as shown in the following example:

```
FpSpread1.ActiveSheet.AddShape(New FarPoint.Win.Spread.DrawingSpace.RectangleShape())
```

This constructs a basic rectangle shape with default properties and adds the shape the active sheet. The following example creates a shape with custom settings.

For more information on shapes, refer to the **Spread Designer Guide (on-line documentation)**.

**Example**

This example creates a shape, changes some of the most used properties, and then adds it to the active sheet.

**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);
   // This code will display the shape property dialog
   //FarPoint.Win.Spread.DrawingSpace.ShapeProps f = new
FarPoint.Win.Spread.DrawingSpace.ShapeProps(fpSpread1);
   //f.Shape = fpSpread1.Sheets[0].DrawingContainer.GetShape("rShape1");
   //f.ShowDialog();
```

**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)
  ' This code will display the shape property dialog
  ' Dim f As New
  'FarPoint.Win.Spread.DrawingSpace.ShapeProps(FpSpread1)
  'f.Shape =
  'FpSpread1.Sheets(0).DrawingContainer.GetShape("rShape1")
  'f.ShowDialog()
```

## Working with Annotations

You can add free-hand drawing to the drawing layer of the sheet using annotation mode. Annotation mode allows the user to draw in free-hand mode on the shapes layer and to save the annotation on the sheet as a shape.

Use **StartAnnotationMode ('StartAnnotationMode Method' in the on-line documentation)** and **StopAnnotationMode ('StopAnnotationMode Method' in the on-line documentation)** to turn on and off the ability of the user to draw on the drawing layer. There are corresponding events, **AnnotationModeStarting ('AnnotationModeStarting Event' in the on-line documentation)** and **AnnotationModeEnding ('AnnotationModeEnding Event' in the on-line documentation)**. You can also customize what occurs using the **OnAnnotationModeStarting ('OnAnnotationModeStarting Method' in the on-line documentation)** and **OnAnnotationModeEnding ('OnAnnotationModeEnding Method' in the on-line documentation)** methods.

By default a thin black line is drawn. You can change this and you can have the component display a dialog for the user to select the drawing pen or stencil or bitmap pattern.

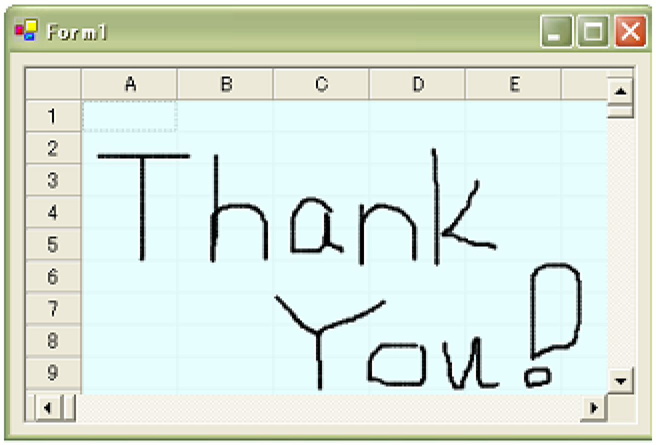
Use the *CloseFigure* parameter if you want to close the drawing between the start and end points with a straight line.

You can use the **CancelAnnotationMode ('CancelAnnotationMode Method' in the on-line documentation)** to cancel annotation mode and return to normal cell editing mode.

For more information on built-in and custom shapes that can also be placed on the drawing layer, refer to **Working with Shapes in Code**.

## 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

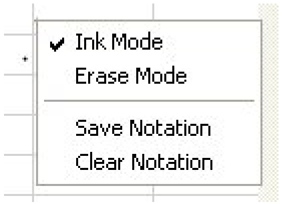
Microsoft Tablet PC SDK.

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.

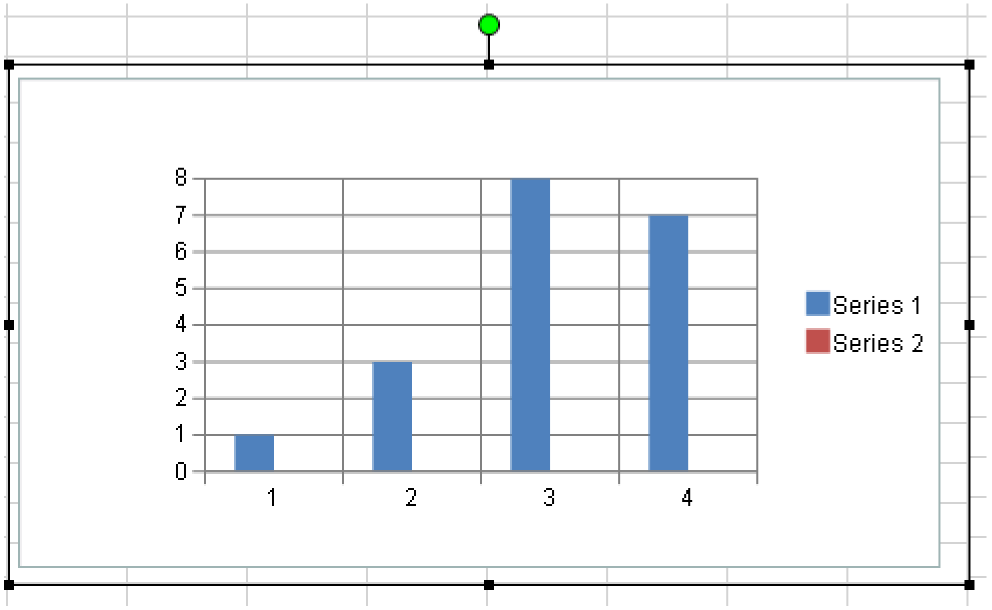
When the user right-clicks on the mouse, a context menu is displayed as shown in the following figure.



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.

## 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.

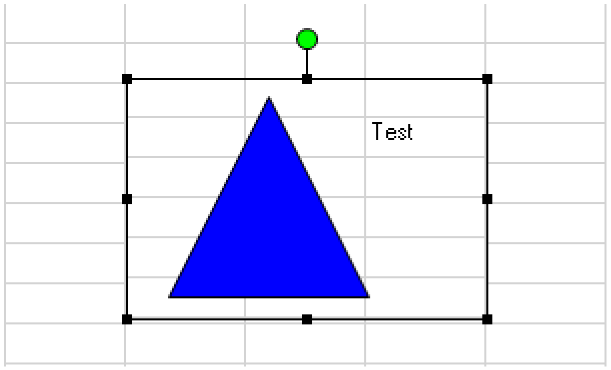
The camera shape cannot use another camera shape.

**Using Code**

1. Create a camera shape object by using the **SpreadCameraShape ('SpreadCameraShape Class' in the on-line 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. The following image displays a camera shape that contains a triangle shape and a cell with text.



## 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)
```

**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.

## 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:

- **Allowing User Interaction with Rows and Columns**
- **Managing Filtering of Rows of User Data**
- **Managing Grouping of Rows of User Data**
- **Managing Outlines (Range Groups) of Rows and Columns**
- **Managing Sorting of Rows of User 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 the static appearance of rows and columns, refer to **Customizing the Row or Column Appearance**.

To customize other aspects of interactivity at the sheet level, refer to **Customizing Sheet Interaction** and 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)**

## Allowing User Interaction with Rows and Columns

You can customize the user interaction with rows and columns. To customize this aspect of user interaction, you may perform the following tasks:

- **Allowing the User to Enter Data in Rows or Columns**
- **Allowing the User to Move Rows or Columns**
- **Allowing the User to Resize Rows or Columns**
- **Setting Fixed (Frozen) Rows or Columns**
- **Setting up Preview Rows**

For more information about interacting with rows and columns, refer to:

- **Resizing the Row or Column to Fit the Data**
- **Allowing the User to Filter Rows**
- **Customizing User Selection of Data**
- **Allowing the User to Automatically Sort Rows**

For information on the underlying model responsible for rows and columns, refer to the **Understanding the Axis Model**.

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.

## Allowing the User to Enter 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.

**Using Code**

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**

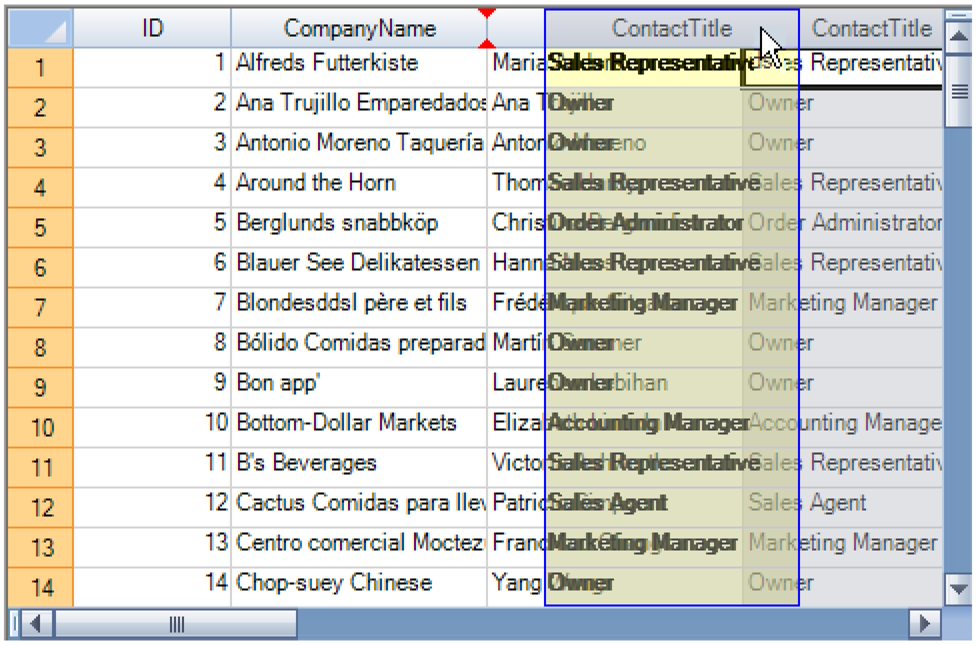
1. Select **Sheet** from the drop-down combo list located on the top right side of the Designer.
2. From the **Behavior** section, select **True** or **False** for **RestrictRows** or **RestrictColumns**.
3. From the **File** menu, select **Save and Exit** to save the changes.

## Allowing the User to Move 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 | Sales Representative | Representativ |
| 2 | 2 | Ana Trujillo Emparedados | Ana T | Owner | Owner |
| 3 | 3 | Antonio Moreno Taquería | Anton | Owner | Owner |
| 4 | 4 | Around the Horn | Thom | Sales Representative | Sales Representativ |
| 5 | 5 | Berglunds snabbköp | Chris | Order Administrator | Order Administrator |
| 6 | 6 | Blauer See Delikatessen | Hann | Sales Representative | Sales Representativ |
| 7 | 7 | Blondesddsl père et fils | Fréde | Marketing Manager | Marketing Manager |
| 8 | 8 | Bólido Comidas preparad | Martí | Owner | Owner |
| 9 | 9 | Bon app' | Laure | Owner | Owner |
| 10 | 10 | Bottom-Dollar Markets | Eliza | Accounting Manager | Accounting Manage |
| 11 | 11 | B's Beverages | Victo | Sales Representative | Sales Representativ |
| 12 | 12 | Cactus Comidas para lle | Patric | Sales Agent | Sales Agent |
| 13 | 13 | Centro comercial Moctez | Franc | Marketing Manager | Marketing Manager |
| 14 | 14 | Chop-suey Chinese | Yang | Owner | Owner |

### Moving the Row or Column in Code

To relocate a row, use the SheetView.**MoveRow** method; to remove multiple rows at a time, use the SheetView.**RemoveRows** method. To relocate a column, use the SheetView.**MoveColumn** method; to remove multiple columns at a time, use the SheetView.**RemoveColumns** method. Alternatively, you can use the DefaultSheetAxisModel.**Move** method.

You can use the **GetColumnFromTag** method to find columns based on their **Tag** property, which you can set programmatically to whatever you want. You can do the same with the **GetRowFromTag** method for rows.

You can remove more than one column at a time using the **Remove** method on a range of columns or rows. For example,

```
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.

### Using Code

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
```

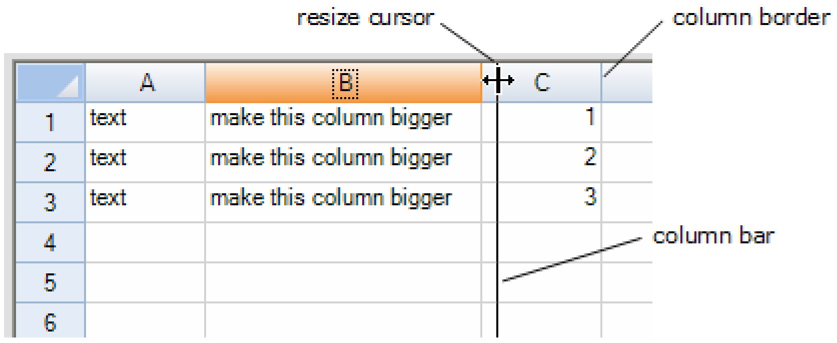
**Using the Spread Designer**

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.

## 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 to allow the user to resize rows and the **Resizable ('Resizable Property' in the on-line documentation)** property for the column 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.



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.

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.

**Using Code**

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
```

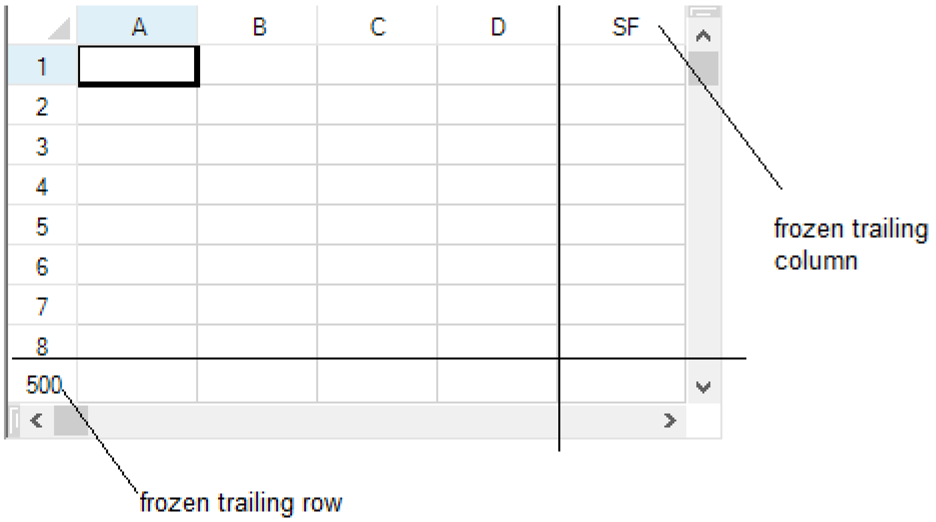
**Using the Spread Designer**

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.

## Setting Fixed (Frozen) Rows or Columns

You can freeze (make unscrollable) a number of either rows or columns or both in a sheet. You can freeze any number of top rows, called leading rows or any number of left-most columns, called leading columns. You can also freeze any number of the trailing bottom rows or trailing right-most columns. The frozen leading rows and columns stay at the top and left of the view regardless of the scrolling. The frozen trailing rows and columns stay at the bottom and right of the view regardless of the scrolling.

The figure below displays one frozen trailing row and column that stay in view:



Properties involved with frozen rows and columns are:

- **FrozenRowCount ('FrozenRowCount Property' in the on-line documentation)**
- **FrozenColumnCount ('FrozenColumnCount Property' in the on-line documentation)**
- **FrozenTrailingColumnCount ('FrozenTrailingColumnCount Property' in the on-line documentation)**
- **FrozenTrailingRowCount ('FrozenTrailingRowCount Property' in the on-line documentation)**

Frozen rows or columns are nonscrollable at run time, but during design time, they are still scrollable.

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 repeated. For more information about repeating rows and columns, refer to **Repeating Rows or Columns on Printed Pages**.

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:

| | | |
|---|---|---|
| -1 | = | leading frozen |
| 0 | = | first scrollable |
| 1 | = | second scrollable |
| ... | | |
| n-1 | = | last scrollable |
| n | = | trailing frozen |

For more information about viewports, refer to **Customizing Viewports**.

In hierarchical displays, frozen leading rows and columns are possible in child sheets.

**Using the Properties Window**

1. At design time, in the **Properties** window, select the Sheet.
2. In the **SheetView Collection Editor** (**Appearance** section), select the **FrozenRowCount**, **FrozenColumnCount**, **FrozenTrailingRowCount**, or the **FrozenTrailingColumnCount** property.
3. Type a value. (The default is 0.) Repeat this for each property.

**Using Code**

Set the **FrozenRowCount ('FrozenRowCount Property' in the on-line documentation)**, **FrozenColumnCount ('FrozenColumnCount Property' in the on-line documentation)**, **FrozenTrailingColumnCount ('FrozenTrailingColumnCount Property' in the on-line documentation)**, or the **FrozenTrailingRowCount ('FrozenTrailingRowCount Property' in the on-line documentation)** property for the sheet.

**Example**

**C#**

```
fpSpread1.Sheets[0].FrozenColumnCount = 2;
fpSpread1.Sheets[0].FrozenRowCount = 2;
fpSpread1.Sheets[0].FrozenTrailingColumnCount = 2;
fpSpread1.Sheets[0].FrozenTrailingRowCount = 2;
```

**VB**

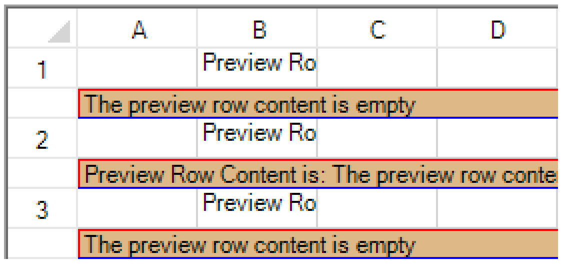
```
FpSpread1.Sheets(0).FrozenColumnCount = 2
FpSpread1.Sheets(0).FrozenRowCount = 2
FpSpread1.Sheets(0).FrozenTrailingColumnCount = 2
FpSpread1.Sheets(0).FrozenTrailingRowCount = 2
```

**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**, type a number for **FrozenColumnCount**, **FrozenRowCount**, **FrozenTrailingColumnCount**, or **FrozenTrailingRowCount**.
3. From the **File** menu, select **Save and Exit** to save the changes.

## 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 **PreviewRowInfo ('PreviewRowInfo Class' in the on-line documentation) Visible ('Visible Property' in the on-line documentation)** property to true to see the preview row. Use the **PreviewRowInfo ('PreviewRowInfo Class' in the on-line documentation) ColumnIndex ('ColumnIndex Property' in the on-line documentation)** property 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.

**Using Code**

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
```

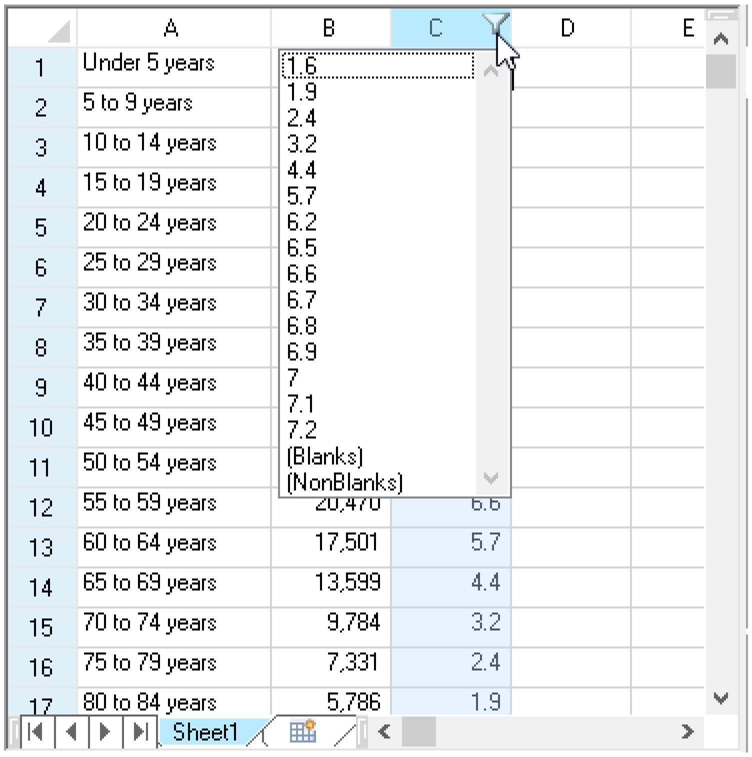
**Using the Spread Designer**

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.

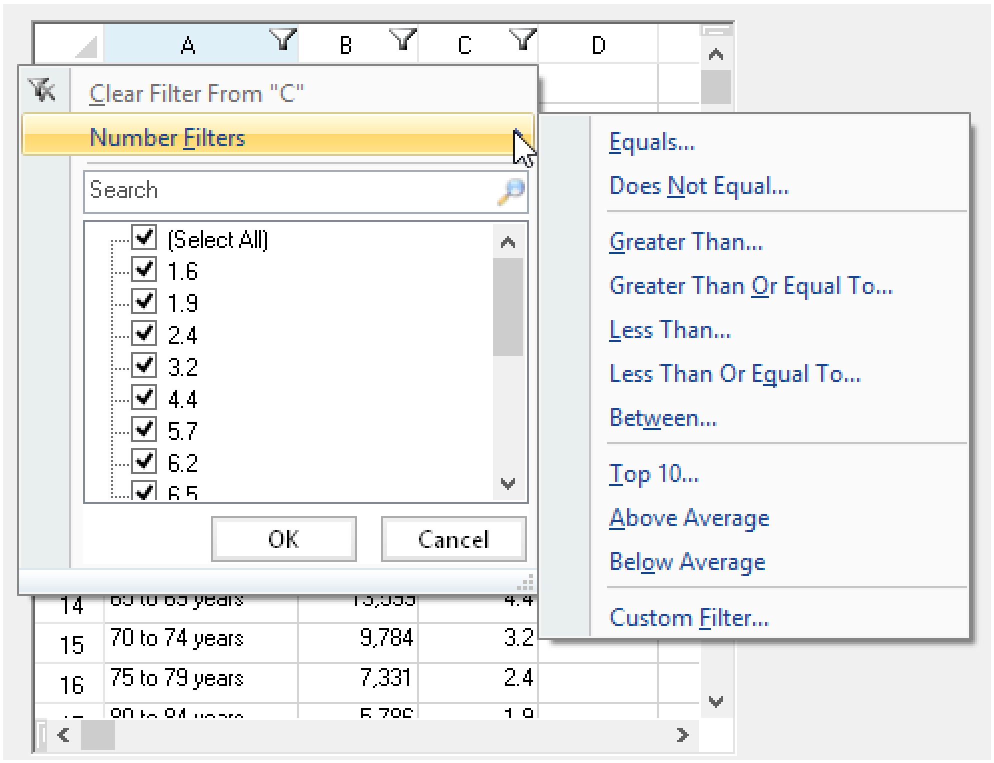
## Managing Filtering of Rows of User Data

Spread provides two types of filtering, simple and enhanced. The simple filtering is the style of filtering provided in this and previous releases of Spread. The enhanced filtering is similar to Excel's filter feature.

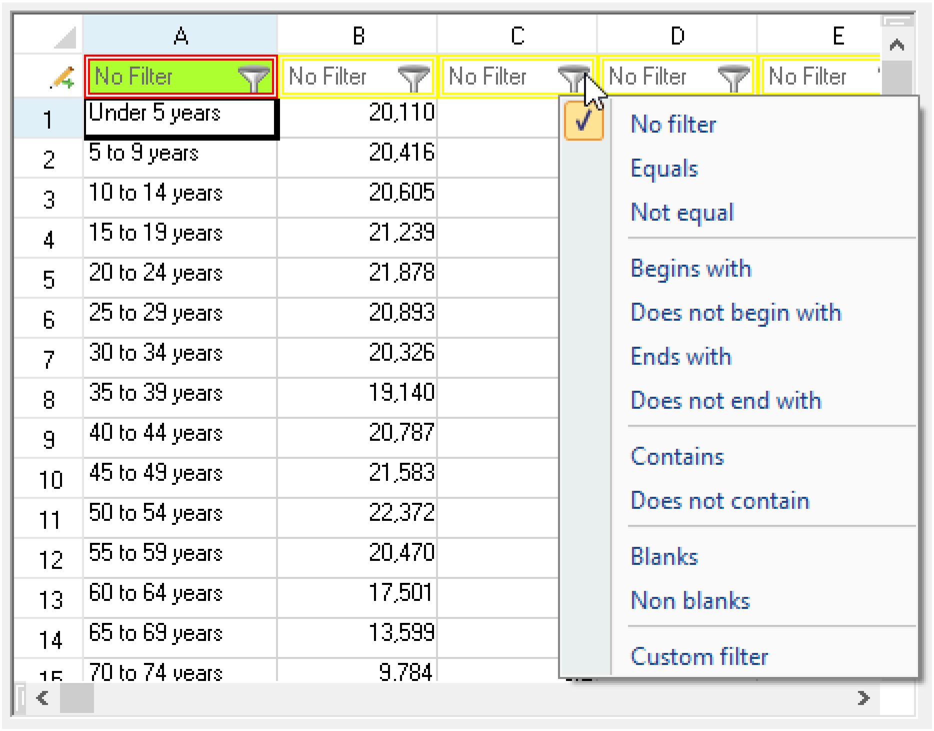
Each type of filtering provides a way for users to change data's appearance or temporarily hide data based on conditions that they specify, as shown in the following figures. This figure illustrates the simple filter.



The following figure illustrates the enhanced filter.



If you provide enhanced filtering, you can choose to offer a filter list to assist users in setting up their filter, as shown in the preceding figure, or a filter bar, as shown in the following figure.



| | A | B |
|---|---|---|
| | No Filter | No Filter |
| 1 | Under 5 years | 20,110 |
| 2 | 5 to 9 years | 20,416 |
| 3 | 10 to 14 years | 20,605 |
| 4 | 15 to 19 years | 21,239 |
| 5 | 20 to 24 years | 21,878 |
| 6 | 25 to 29 years | 20,893 |
| 7 | 30 to 34 years | 20,326 |
| 8 | 35 to 39 years | 19,140 |
| 9 | 40 to 44 years | 20,787 |
| 10 | 45 to 49 years | 21,583 |
| 11 | 50 to 54 years | 22,372 |
| 12 | 55 to 59 years | 20,470 |
| 13 | 60 to 64 years | 17,501 |
| 14 | 65 to 69 years | 13,599 |
| 15 | 70 to 74 years | 9,784 |

Specify the way users are allowed to filter data using the instructions in **Allowing the User to Filter Rows**. You can customize many features for each type of filtering, as well as the display of filtered rows, as described in the following sections.

- **Customizing Simple Filtering**
- **Customizing Enhanced Filtering**

**Understanding Simple Row Filtering** and **Understanding Enhanced Row Filtering** describe how users interact with the filter features you create.

## 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.

**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.

## 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.

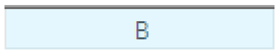
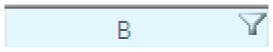
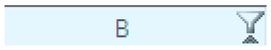
- **Understanding Simple Row Filtering**
- **Setting the Appearance of Filtered Rows**
- **Customizing the Filter List**
- **Creating a Custom Filter**
- **Setting the Appearance of Filter Indicators**

## Understanding Simple Row Filtering

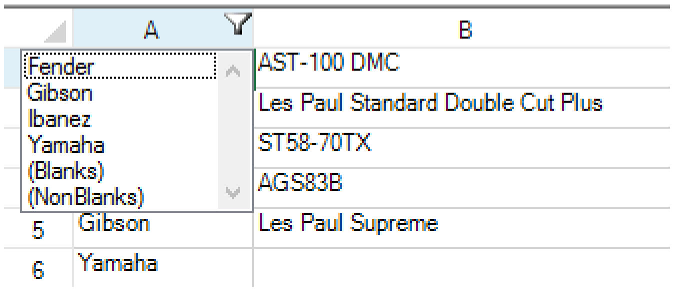
This topic summarizes how the end user interacts with the simple filter feature.

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**.

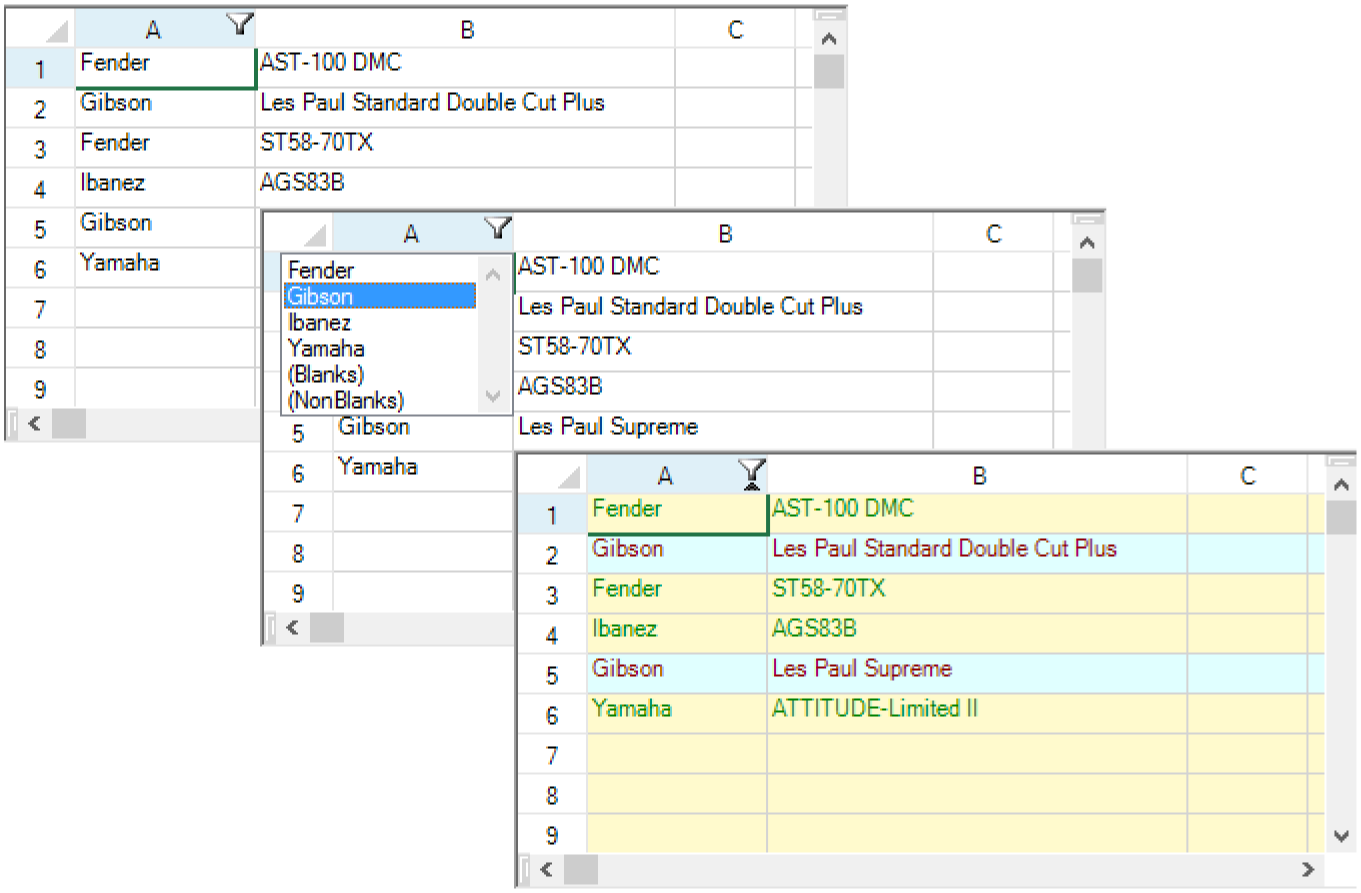
To understand the aspects of filtering that can be customized, return to the overview at **Managing Filtering of Rows of User Data**.

## 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 | | |
| 6 | Yamaha | | |
| 7 | | | |
| 8 | | | |
| 9 | | | |

| | 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 | |
| 7 | | | |
| 8 | | | |
| 9 | | | |

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)
```

## 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.NamedStyleim 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;
```

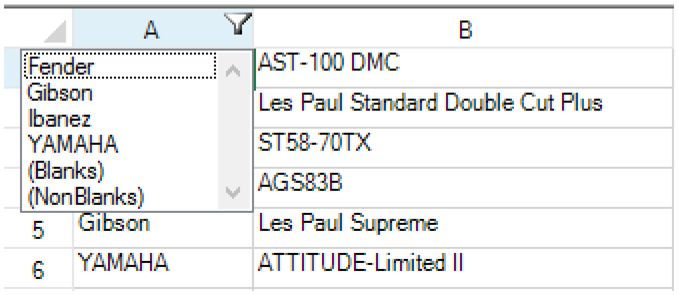
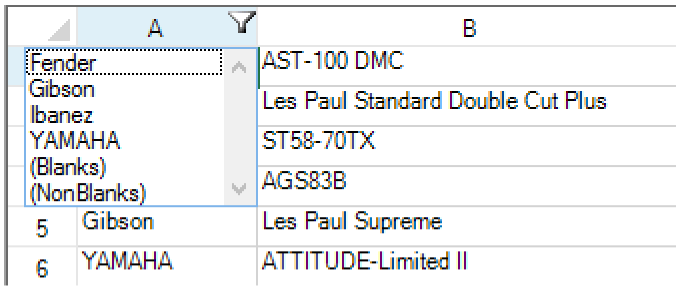
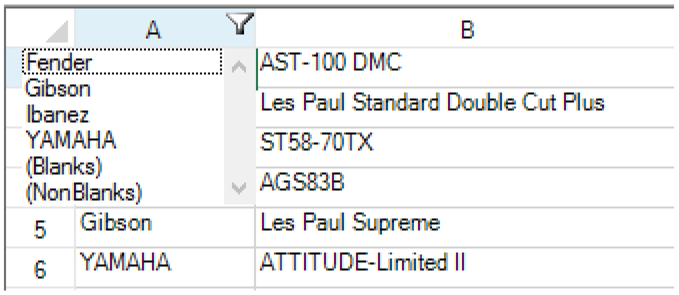
**VB**

```
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** | **Example** |
|---|---|
| Fixed, three-dimensional (default) |  A, B; Fender, Gibson, Ibanez, YAMAHA, (Blanks), (NonBlanks); AST-100 DMC; Les Paul Standard Double Cut Plus; ST58-70TX; AGS83B; 5 Gibson Les Paul Supreme; 6 YAMAHA ATTITUDE-Limited II |
| Fixed, single-line |  A, B; Fender, Gibson, Ibanez, YAMAHA, (Blanks), (NonBlanks); AST-100 DMC; Les Paul Standard Double Cut Plus; ST58-70TX; AGS83B; 5 Gibson Les Paul Supreme; 6 YAMAHA ATTITUDE-Limited II |
| None |  A, B; Fender, Gibson, Ibanez, YAMAHA, (Blanks), (NonBlanks); AST-100 DMC; Les Paul Standard Double Cut Plus; ST58-70TX; AGS83B; 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)**

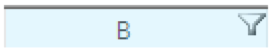
## Setting the Appearance of Filter Indicators

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 displays the filter indicator and sets the filter strings.

**C#**

```
fpSpread1.ActiveSheet.Columns[1].AllowAutoFilter = true;
fpSpread1.ActiveSheet.RowFilter.ShowFilterIndicator = true;
fpSpread1.ActiveSheet.RowFilter.AllString = "Show All";
fpSpread1.ActiveSheet.RowFilter.BlanksString = "Show The Blanks";
fpSpread1.ActiveSheet.RowFilter.NonBlanksString = "Show The Non-Blanks";
```

**VB**

```
FpSpread1.ActiveSheet.Columns(1).AllowAutoFilter = True
FpSpread1.ActiveSheet.RowFilter.ShowFilterIndicator = True
FpSpread1.ActiveSheet.RowFilter.AllString = "Show All"
FpSpread1.ActiveSheet.RowFilter.BlanksString = "Show The Blanks"
FpSpread1.ActiveSheet.RowFilter.NonBlanksString = "Show The Non-Blanks"
```

## 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.ColumnHeaderRowCount = 3;
fpSpread1.ActiveSheet.ColumnHeader.AutoFilterIndex = 1;
```

**VB**

```
FpSpread1.ActiveSheet.Columns(1).AllowAutoFilter = True
FpSpread1.ActiveSheet.ColumnHeaderRowCount = 3
FpSpread1.ActiveSheet.ColumnHeader.AutoFilterIndex = 1
```

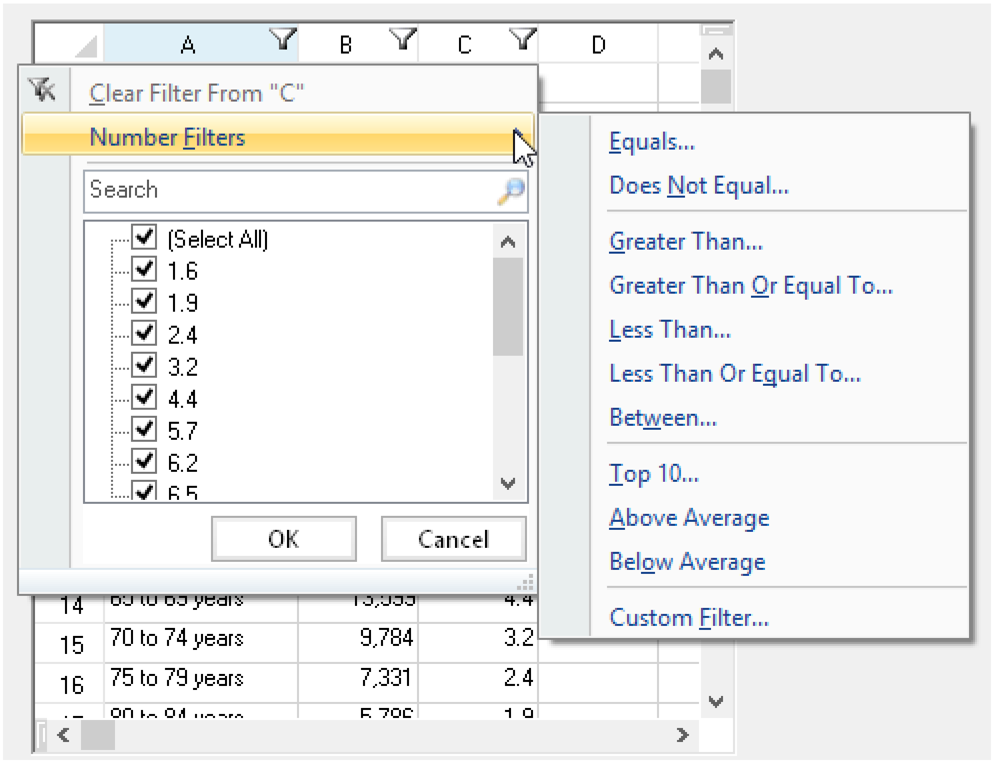
## Customizing Enhanced Filtering

You can customize enhanced filtering by specifying to use the default filter menu or to display a filter bar, as described in **Allowing the User to Filter Rows**. If you decide to have the control display a filter bar, you can customize the filter bar. You can also use an enhanced sort dialog with enhanced filtering.

- **Understanding Enhanced Row Filtering**
- **Customizing the Filter Bar**
- **Adding a Custom Sort Dialog (on-line documentation)**

## Understanding Enhanced Row 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.



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.

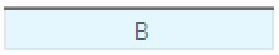
| Filter Type | Description |
|---|---|
| **Number Filters** | |
| Equals | Values in rows are equal to condition |
| Does Not Equal | Values in rows do not equal condition |
| Greater Than | Values in rows are greater than condition |
| Greater Than Or Equal To | Values in rows are greater than or equal 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 |

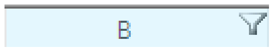
| | |
|---|---|
| Ends With | Values in rows end with the specified characters |
| Contains | Values in rows contain the specified characters |
| Does Not Contain | Values in rows do not contain the specified characters |
| Custom Filter | Values in rows that meet the conditions of a custom filter |
| **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.

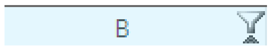
When the user chooses a filter, the control either filters the data to display only the items that match the filter criteria, or the control displays the rows that meet the criteria with one appearance, and the rows that do not meet the criteria with another appearance. For information about setting the styles for rows, see **Setting the Appearance of Filtered Rows**.

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 |

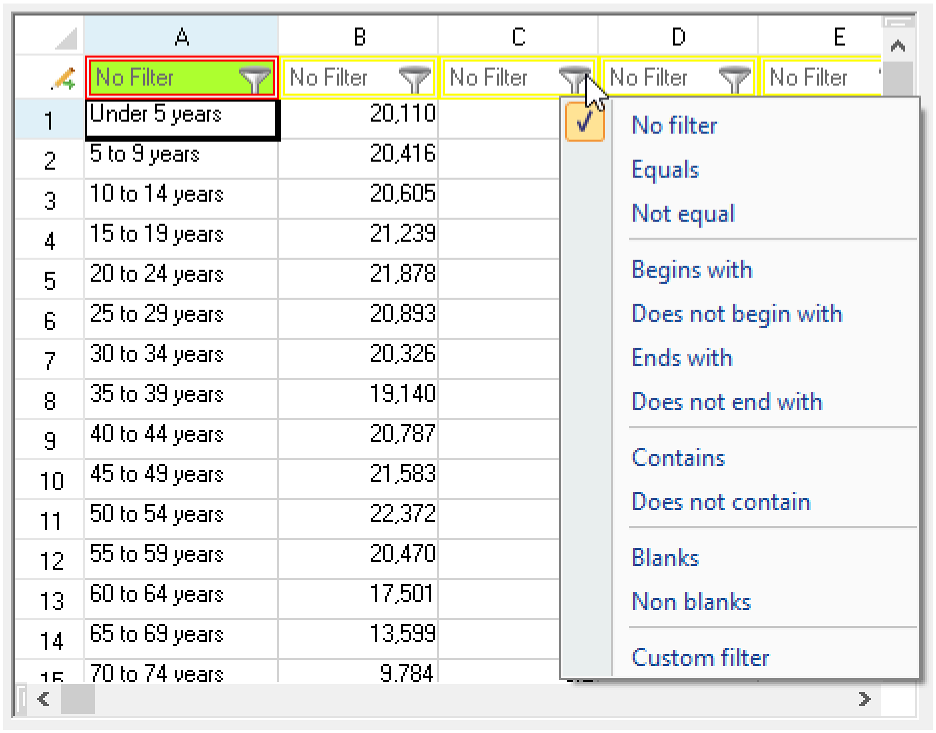


Appearance of header cell with row filtering allowed but no rows filtered



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.



| | A | B | C | D | E |
|---|---|---|---|---|---|
| | No Filter | No Filter | No Filter | No Filter | No Filter |
| 1 | Under 5 years | 20,110 | | | |
| 2 | 5 to 9 years | 20,416 | | | |
| 3 | 10 to 14 years | 20,605 | | | |
| 4 | 15 to 19 years | 21,239 | | | |
| 5 | 20 to 24 years | 21,878 | | | |
| 6 | 25 to 29 years | 20,893 | | | |
| 7 | 30 to 34 years | 20,326 | | | |
| 8 | 35 to 39 years | 19,140 | | | |
| 9 | 40 to 44 years | 20,787 | | | |
| 10 | 45 to 49 years | 21,583 | | | |
| 11 | 50 to 54 years | 22,372 | | | |
| 12 | 55 to 59 years | 20,470 | | | |
| 13 | 60 to 64 years | 17,501 | | | |
| 14 | 65 to 69 years | 13,599 | | | |
| 15 | 70 to 74 years | 9,784 | | | |

**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;
```

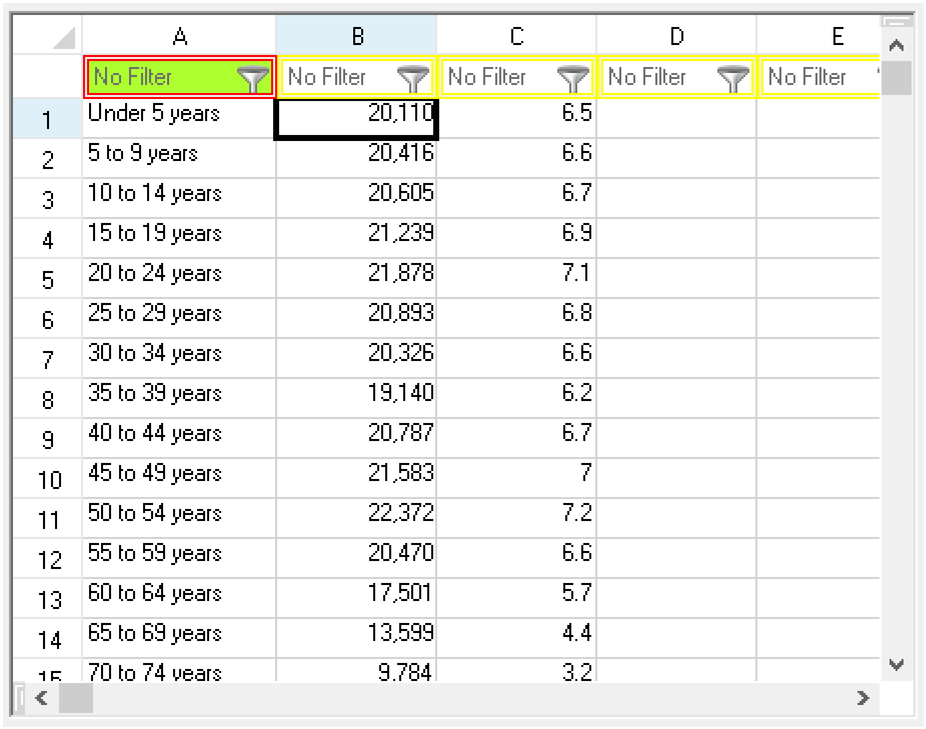
**VB**

```
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**.

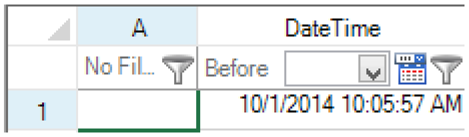
## Customizing the Filter Bar

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)
```

## Managing Grouping of Rows of User Data

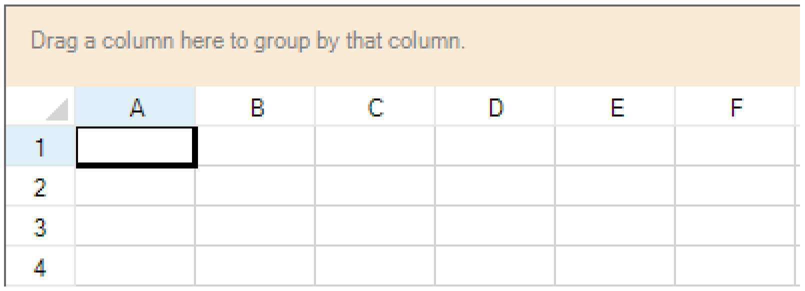
You can set the display of the spreadsheet component to allow rows to be grouped according to the column headers. You can customize the user experience for grouping data on a sheet. With grouping, you can allow the user to group rows of data according to the column headers that are dragged into the group 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.



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#**

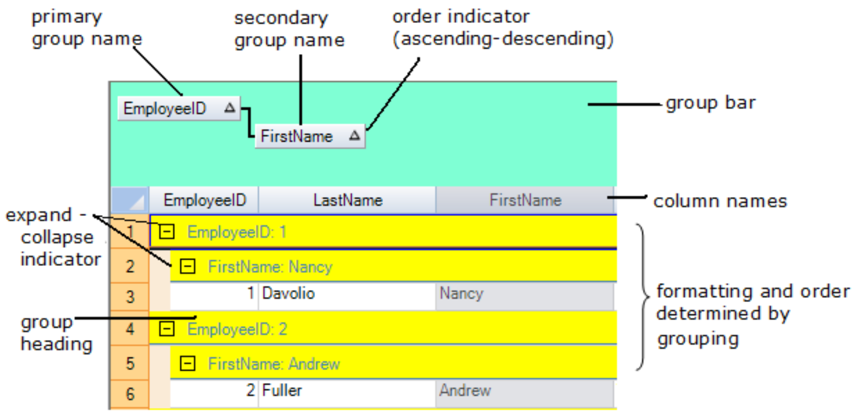
```
FpSpread1.AllowColumnMove = true;
FpSpread1.ActiveSheet.GroupBarInfo.Visible = true;
FpSpread1.ActiveSheet.AllowGroup = true;
```

**VB**

```
FpSpread1.AllowColumnMove = True
FpSpread1.ActiveSheet.GroupBarInfo.Visible = True
FpSpread1.ActiveSheet.AllowGroup = True
```

## Using Grouping

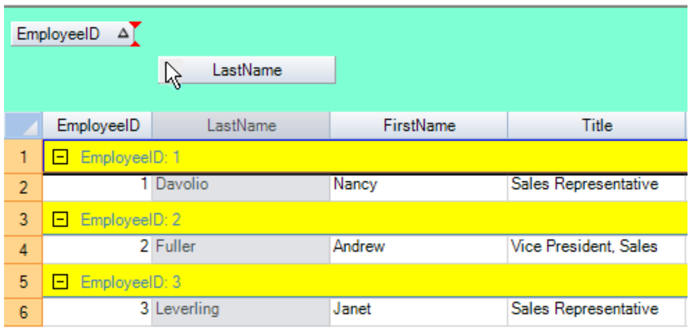
You can set up the display to allow Outlook-style grouping of rows. For large amounts of data, this is helpful to display the data in the order the user needs. The user selects columns by which to sort and the component then organizes and displays the data in a hierarchy with rows organized accordingly. 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.


primary group name
secondary group name
order indicator (ascending-descending)
group bar
column names
expand - collapse indicator
group heading
formatting and order determined by grouping
EmployeeID
FirstName

| | EmployeeID | LastName | FirstName |
|---|---|---|---|
| 1 | EmployeeID: 1 | | |
| 2 | FirstName: Nancy | | |
| 3 | 1 | Davolio | Nancy |
| 4 | EmployeeID: 2 | | |
| 5 | FirstName: Andrew | | |
| 6 | 2 | Fuller | Andrew |

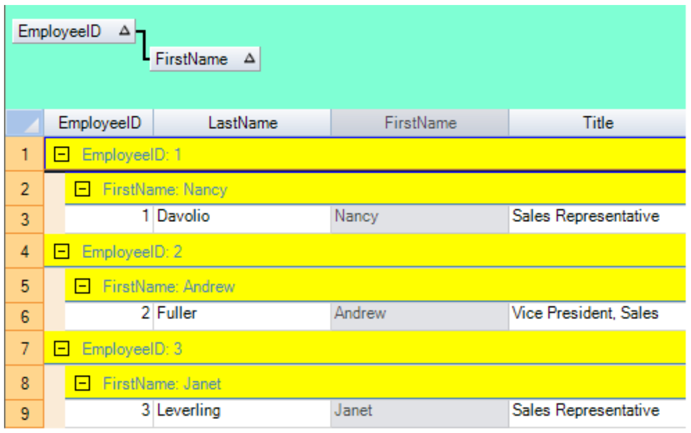
You can expand or collapse groups by clicking the expand (+) or collapse (-) indicators.

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.



| | EmployeeID | LastName | FirstName | Title |
|---|---|---|---|---|
| 1 | EmployeeID: 1 | | | |
| 2 | 1 | Davolio | Nancy | Sales Representative |
| 3 | EmployeeID: 2 | | | |
| 4 | 2 | Fuller | Andrew | Vice President, Sales |
| 5 | EmployeeID: 3 | | | |
| 6 | 3 | Leverling | Janet | Sales Representative |

Before secondary grouping: dragging the column header into the grouping bar.



| | EmployeeID | LastName | FirstName | Title |
|---|---|---|---|---|
| 1 | EmployeeID: 1 | | | |
| 2 | FirstName: Nancy | | | |
| 3 | 1 | Davolio | Nancy | Sales Representative |
| 4 | EmployeeID: 2 | | | |
| 5 | FirstName: Andrew | | | |
| 6 | 2 | Fuller | Andrew | Vice President, Sales |
| 7 | EmployeeID: 3 | | | |
| 8 | FirstName: Janet | | | |
| 9 | 3 | Leverling | Janet | Sales Representative |

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 proceeding 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 |
|---|---|

| | |
|---|---|
| **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 |
| **Grouped ('Grouped Event' in the on-line documentation)** and **Grouping ('Grouping 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 |

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

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**. You can use the **IsGroup ('IsGroup Method' in the on-line documentation)** method, which determines whether a requested row is a data row or a group header row.

For more information about group footers, refer to **Displaying a Footer for Columns or Groups**.

## 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.

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.

## 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**

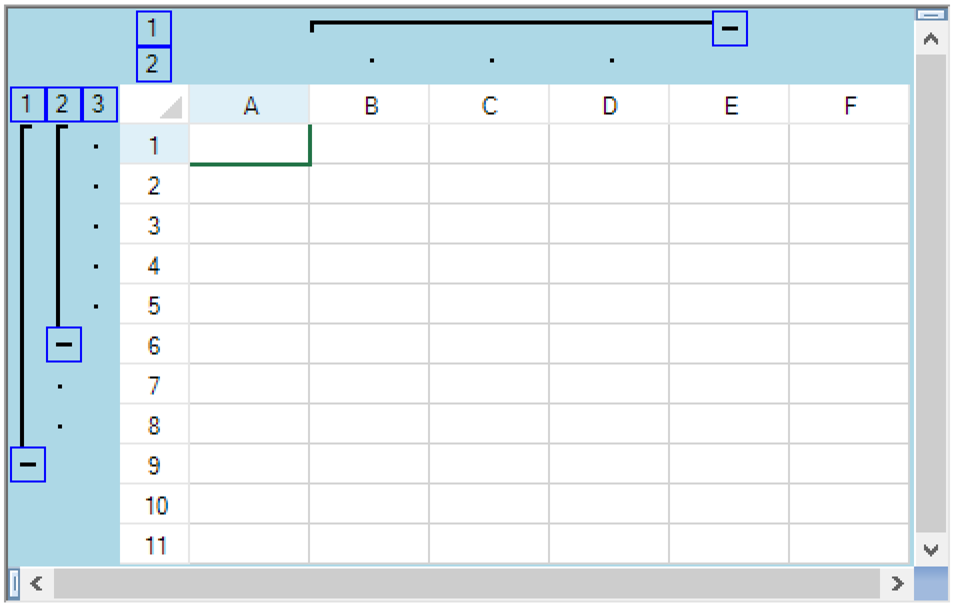
These features do not interoperate with grouping in Spread.

| 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. |

## 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)**

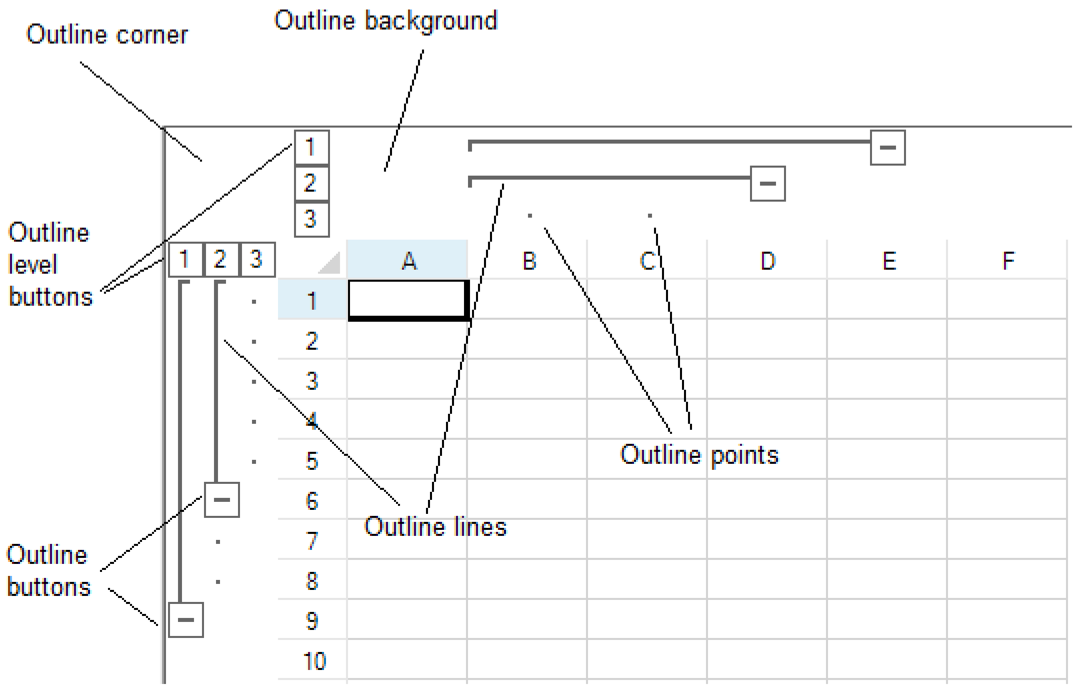
Since outlines affect the performance of other features, be sure to read **Interoperability of Outlines with Other Features**.

## 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)
```

## 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.

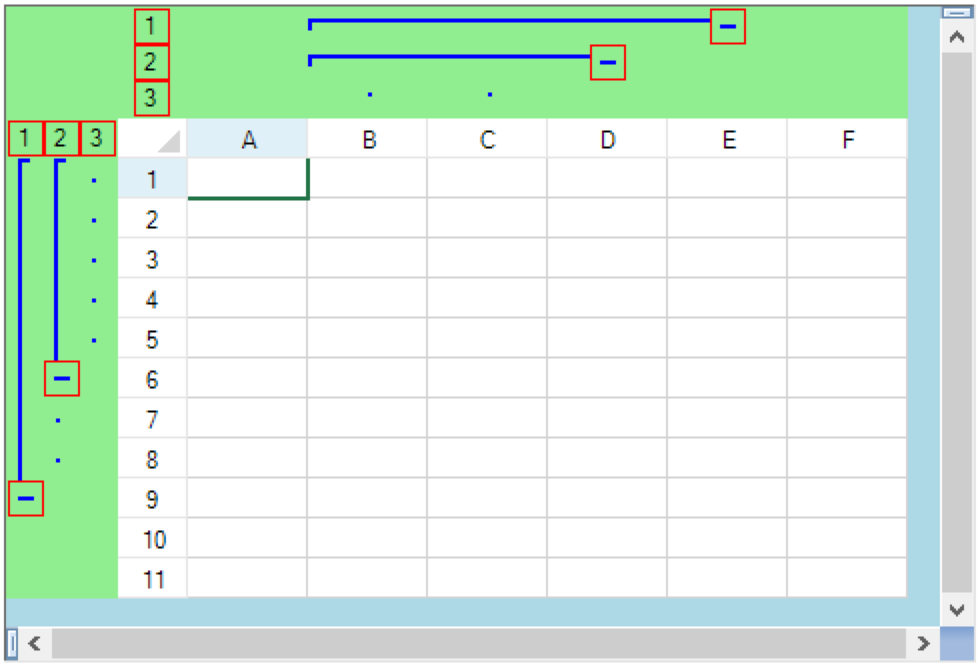
The two properties in the **SheetView** class that can be used to customize the appearance are:

- **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)**
- **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. |

## 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**

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.

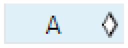
Sorting is not intended to be used when Outlook-style grouping is turned on. For more information about grouping (which is a type of sorting), refer to **Managing Grouping of Rows of User Data**.

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

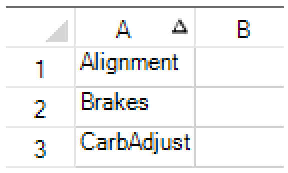
You can set the spreadsheet to allow the user to automatically sort the data when a column header is clicked. The first time the column header is clicked (selected) the unsorted icon is displayed. The second click displays the sort icon and sorts the column. If the user clicks successively on the same 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.

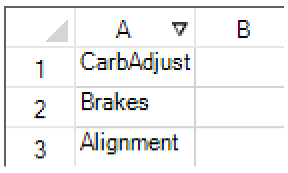
The **SetColumShowSortIndicator ('SetColumnShowSortIndicator Method' in the on-line documentation)** method or **ShowSortIndicator ('ShowSortIndicator Property' in the on-line documentation)** property can be set to display or hide the sort indicator. The sort indicator appears in the header column as 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 **SetColumShowSortIndicator ('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 **SetColumShowSortIndicator ('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#**

```csharp
fpSpread1.ActiveSheet.SetColumnAllowAutoSort(0, true);
fpSpread1.ActiveSheet.SetColumnShowSortIndicator(0, false);
fpSpread1.ActiveSheet.AutoSortColumn(0);
```

**VB**

```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.

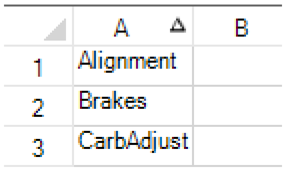
## Setting the Appearance of Sort Indicators

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
```

## Customizing Interaction with 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 How Cell Types Work**
- **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 How Cell Types Work

These topics describe how cell types work:

- **Understanding Cell Type Basics**
- **Determining the Cell Type of the Active Cell**
- **Understanding How Cell Types Display and Format Data**
- **Understanding How Cell Type Affects Model Data**

## 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.

## Determining the Cell Type of the Active Cell

You can determine the cell type of the active cell. You can use the **GetCellType ('GetCellType Method' in the on-line documentation)** method of the **SheetView ('SheetView Class' in the on-line documentation)** class.

## 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** | "123-45- | "123-45-6789" | "123456789" |

| | | | |
|---|---|---|---|
| **('GcCharMaskCellType Class' in the on-line documentation)** | 6789" | | |
| **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 (unchecked) | "False" | 0 |

| | | | |
|---|---|---|---|
| | 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 of numeric value) | 15 (actual 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 | Data Type Written to Model |
|---|---|

| | |
|---|---|
| **BarCodeCellType ('BarCodeCellType Class' in the on-line documentation)** | Value of barcode |
| **ButtonCellType ('ButtonCellType Class' in the on-line documentation)** two-state | Integer |
| **ButtonCellType ('ButtonCellType Class' in the on-line documentation)** one-state | Null |
| **CheckBoxCellType ('CheckBoxCellType Class' in the on-line documentation)** three-state | Integer (0 = false, 1= true, 2 = indeterminate) |
| **CheckBoxCellType ('CheckBoxCellType Class' in the on-line documentation)** two-state | Boolean |
| **ColorPickerCellType ('BarCodeCellType Class' in the on-line documentation)** | Null |
| **ComboBoxCellType ('ComboBoxCellType Class' in the on-line documentation)** and **MultiColumnComboBoxCellType ('MultiColumnComboBoxCellType Class' in the on-line documentation)** | Depends on value of **EditorValue** property. String, if EditorValue = String or item data. Integer, if EditorValue = index |
| **GcComboBoxCellType ('GcComboBoxCellType Class' in the on-line documentation)** | Depends on value of **EditorValue** property. String, if EditorValue = String or item data. Integer, if EditorValue = index |
| **HyperLinkCellType ('HyperLinkCellType Class' in the on-line documentation)** | Array of Boolean values (whether each link is clicked or unclicked) |
| **ImageCellType ('ImageCellType Class' in the on-line documentation)** | System.Drawing.Color or Null |
| **ListBoxCellType ('ListBoxCellType Class' in the on-line documentation)** | Array |
| **MultiOptionCellType ('MultiOptionCellType Class' in the on-line documentation)** | Depends on value of **EditorValue** property. String, if EditorValue = String or item data, Integer, if EditorValue = index |
| **ProgressCellType ('ProgressCellType Class' in the on-line documentation)** | Double |
| **RichTextCellType ('RichTextCellType Class' in the on-line documentation)** | String |
| **SliderCellType ('SliderCellType Class' in the on-line documentation)** | Integer |

## Working with Editable Cell Types

You can work with the editable cell types as described in the following topics:

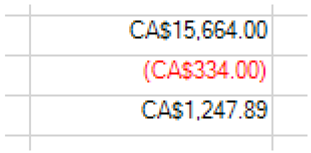
- **Setting a Currency Cell**
- **Setting a Date-Time Cell**
- **Setting a GcCharMask Cell (on-line documentation)**
- **Setting a GcDateTime Cell**
- **Setting a GcMask Cell (on-line documentation)**
- **Setting a GcNumber Cell**

- **Setting a GcTextBox Cell**
- **Setting a GcTimeSpan Cell (on-line documentation)**
- **Setting a General Cell**
- **Setting a Mask Cell**
- **Setting a Number Cell**
- **Setting a Percent Cell**
- **Setting a Regular Expression Cell**
- **Setting a Text Cell**

For other cell types, refer to **Working with Graphical Cell Types**.

## 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.



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**

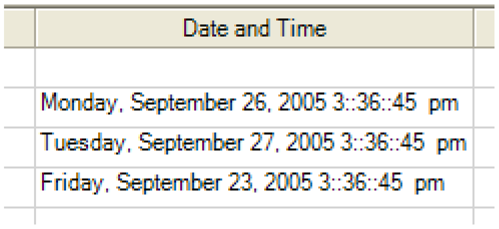
```
Dim currcell As New FarPoint.Win.Spread.CellType.CurrencyCellType()
currcell.CurrencySymbol = "US$"
currcell.DecimalSeparator = ":"
currcell.DecimalPlaces = 8
FpSpread1.ActiveSheet.Cells(1,1).CellType = currcell
```

### 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 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.

## 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.

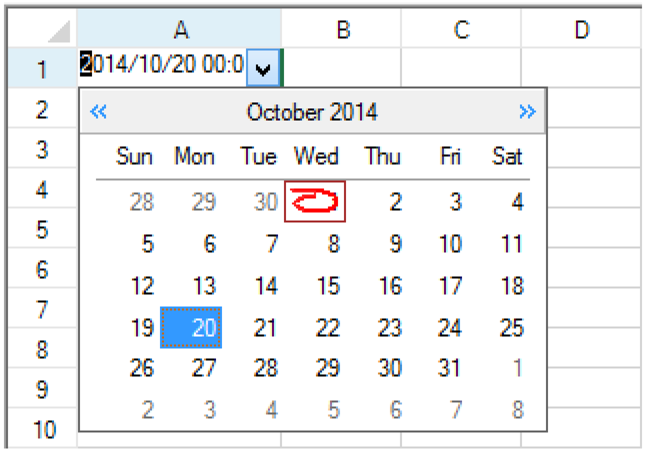
The GcDateTime cell supports different calendar styles from the DateTime cell. You can specify which fields to display with the **DisplayFields ('DisplayFields Property' in the on-line documentation)** property.

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

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 **GcDateTime** 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 **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
```

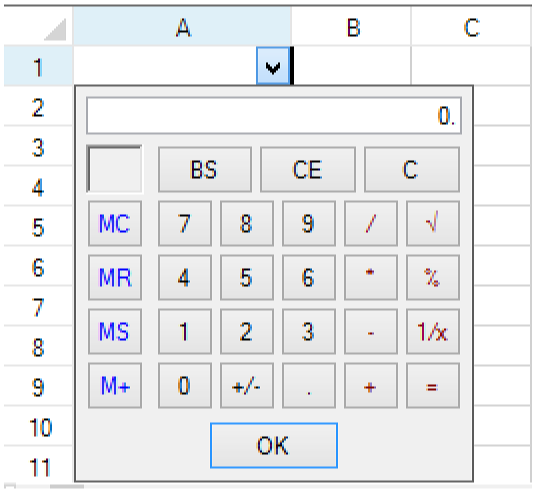
**Using the Spread Designer**

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 GcNumber Cell

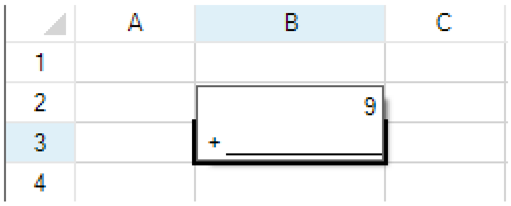
You can create a number cell that displays a side button and calculator. The **GcNumberCellType ('GcNumberCellType Class' in the on-line documentation)** cell is part of the GrapeCity.Win.PluginInputMan assembly.

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.

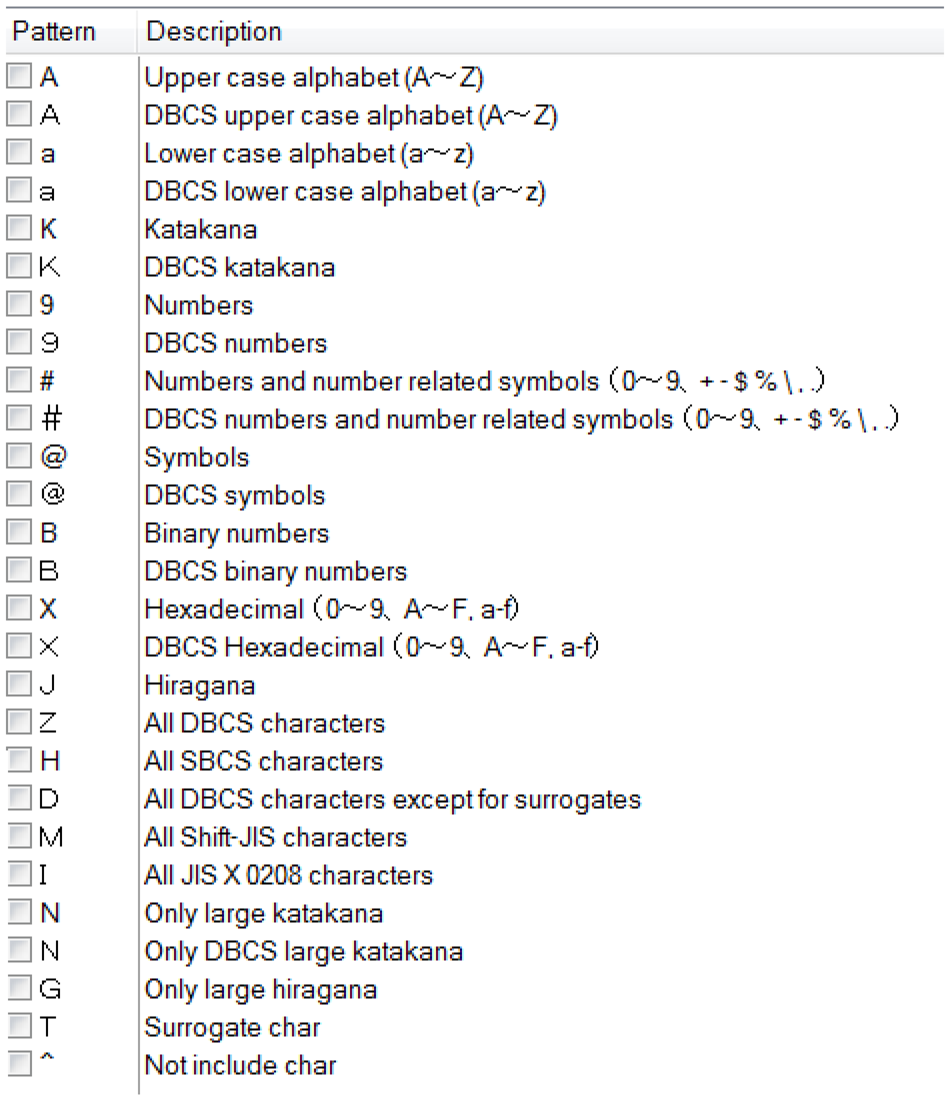
## Setting a GcTextBox Cell

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 |
|---|---|
| ☐ A | Upper case alphabet (A～Z) |
| ☐ Ａ | DBCS upper case alphabet (A～Z) |
| ☐ a | Lower case alphabet (a～z) |
| ☐ ａ | DBCS lower case alphabet (a～z) |
| ☐ K | Katakana |
| ☐ Ｋ | DBCS katakana |
| ☐ 9 | Numbers |
| ☐ ９ | DBCS numbers |
| ☐ # | Numbers and number related symbols（0～9、+ - $ % \ , .） |
| ☐ ＃ | DBCS numbers and number related symbols（0～9、+ - $ % \ , .） |
| ☐ @ | Symbols |
| ☐ ＠ | DBCS symbols |
| ☐ B | Binary numbers |
| ☐ Ｂ | DBCS binary numbers |
| ☐ X | Hexadecimal（0～9、A～F, a-f） |
| ☐ Ｘ | DBCS Hexadecimal（0～9、A～F, a-f） |
| ☐ J | Hiragana |
| ☐ Z | All DBCS characters |
| ☐ H | All SBCS characters |
| ☐ D | All DBCS characters except for surrogates |
| ☐ M | All Shift-JIS characters |
| ☐ I | All JIS X 0208 characters |
| ☐ N | Only large katakana |
| ☐ N | Only DBCS large katakana |
| ☐ G | Only large hiragana |
| ☐ T | Surrogate char |
| ☐ ^ | 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 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 gnrlcell As New FarPoint.Win.Spread.GeneralCellType
gnrlcell.FormatString = "#,###.00"
FpSpread1.Sheets(0).Cells(1, 1).CellType = gnrlcell
```

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
```

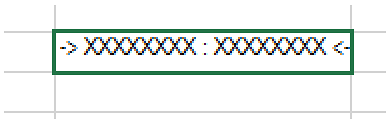
**Using the Spread Designer**

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 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 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** | Sets whether values are represented as fractions. |

| | |
|---|---|
| **documentation)** | |
| **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#**

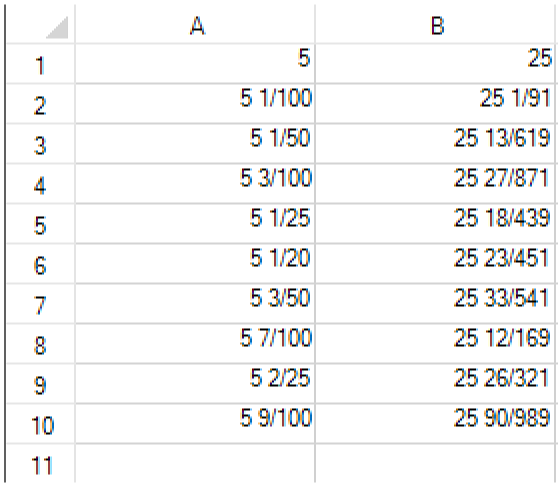
```
fpSpread1.ActiveSheet.Columns[0, 9].Width = 120;
FarPoint.Win.Spread.CellType.NumberCellType frac = new
FarPoint.Win.Spread.CellType.NumberCellType();
frac.FractionMode = true;
frac.FractionConvertWholeNumbers = false;
frac.FractionDenominatorPrecision =
FarPoint.Win.Spread.CellType.FractionDenominatorPrecision.Custom;
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.



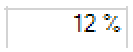
| | A | B |
|---|---|---|
| 1 | 5 | 25 |
| 2 | 5 1/100 | 25 1/91 |
| 3 | 5 1/50 | 25 13/619 |
| 4 | 5 3/100 | 25 27/871 |
| 5 | 5 1/25 | 25 18/439 |
| 6 | 5 1/20 | 25 23/451 |
| 7 | 5 3/50 | 25 33/541 |
| 8 | 5 7/100 | 25 12/169 |
| 9 | 5 2/25 | 25 26/321 |
| 10 | 5 9/100 | 25 90/989 |
| 11 | | |

**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 Percent Cell

You can use a percent cell for displaying values as percentages and restricting inputs to percentage numeric values.



12 %

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 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.

## Working with Graphical Cell Types

You can work with the graphical cell types as described in the following topics:

- **Setting a Barcode Cell**
- **Setting a Button Cell**
- **Setting a Check Box Cell**
- **Setting a Color Picker Cell**
- **Setting a Combo Box Cell**
- **Setting a GcComboBox Cell (on-line documentation)**

- **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 Rich Text Cell**
- **Setting a Slider Cell**

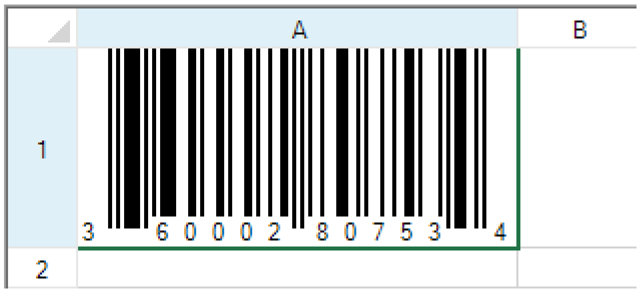
The graphical cell types use a graphic or a control or form. They are based on the **BaseCellType ('BaseCellType Class' in the on-line documentation)** class and you can define a subeditor for them.

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** | Sets the height and width of the barcode. |

| | |
|---|---|
| **documentation)** | |
| **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: [http://www.gs1.org/barcodes](http://www.gs1.org/barcodes).

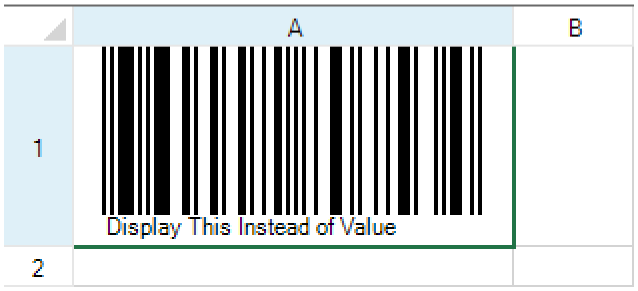
| Barcode Type | Sample Barcode Image |
|---|---|
| Code128 | 72894736 |

| | |
|---|---|
| Code39 | 72894736 |
| Code49 | 72894736 |
| Code93 | 72894736 |
| EAN128 | 72894736 |
| ITF | 72 89473 6 |
| Jan13 | 0 000072 894736 |
| Jan8 | 7 2 8 9 4 7 3 6 |
| JapanesePostal | |

| | |
|---|---|
| NW7 | a728947362a |
| PDF417 | |
| PostNet | |
| QRCode | |
| UPC | 0 00072 89473 6 |

**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)**.



**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**

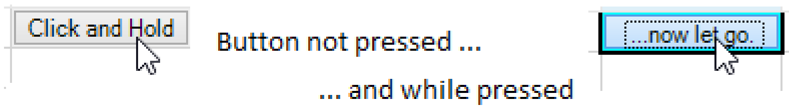
```
Dim brcdcell As New FarPoint.Win.Spread.CellType.BarCodeCellType
FpSpread1.Sheets(0).Cells(0, 0).CellType = brcdcell
FpSpread1.Sheets(0).Cells(0, 0).Value = "12345"
```

**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.



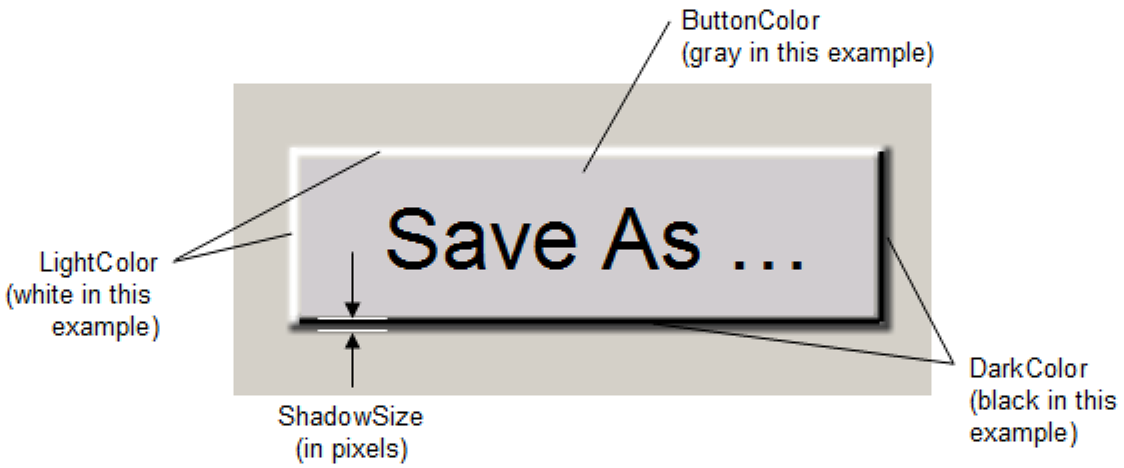
To create a cell that acts like a button, use the **ButtonCellType ('ButtonCellType Class' in the on-line documentation)** class and the settings that are summarized here. You can create a button cell using the examples shown in this topic or in the topics for the individual members of the **ButtonCellType ('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** | Sets the color of the button. |

| | |
|---|---|
| **Property' in the on-line documentation)** | |
| **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. |

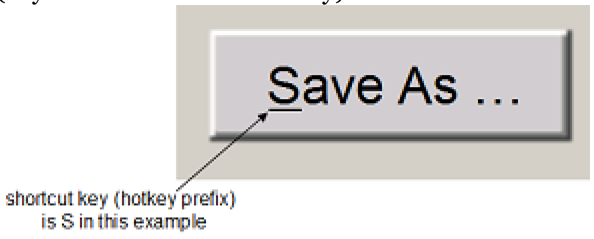


By default, the button has a single state, and changes its appearance only as long as you have the pointer over it with the mouse button pressed. For this setting, button cells behave like push buttons, which you can press by pressing your left mouse button, and which do not stay pressed when you release 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.

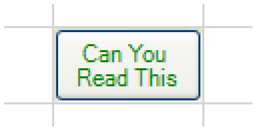
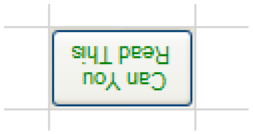
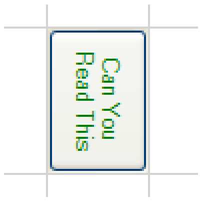
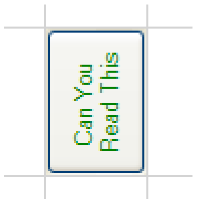
**Customizing the Text Appearance**

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.

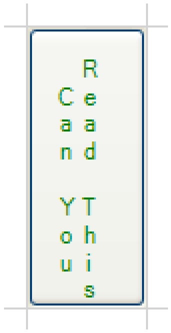
The following properties relate to the text that is displayed in the button cell.

| 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).   Save As ...<br>shortcut key (hotkey prefix) is S in this example |
| **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. |

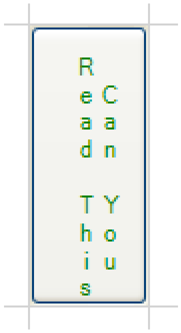
Here are the results of different settings of the **TextOrientation ('TextOrientation Property' in the on-line documentation)** property.

| Text Orientation | Example Button | Text Orientation | Example Button |
|---|---|---|---|
| Text Horizontal |  | Text Horizontal Flipped |  |
| Text Vertical |  | Text Vertical Flipped |  |

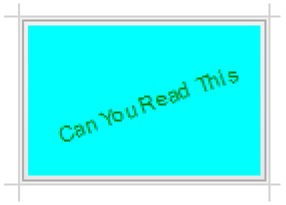
Text TopDown



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.

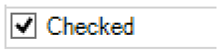
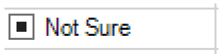
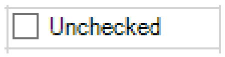
## Setting a Check Box Cell

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 |
|---|---|
|  Checked | True (checked) |
|  Not Sure | Indeterminate (grayed) |
|  Unchecked | 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 ckbxcell = new
FarPoint.Win.Spread.CellType.CheckBoxCellType();
ckbxcell.ThreeState = true;
ckbxcell.TextTrue ="Checked";
ckbxcell.TextFalse ="Unchecked";
ckbxcell.TextIndeterminate ="Not Sure";
fpSpread1.ActiveSheet.Cells[0, 0].CellType = ckbxcell;
```

**VB**

```
Dim ckbxcell As New FarPoint.Win.Spread.CellType.CheckBoxCellType()
ckbxcell.ThreeState = true
ckbxcell.TextTrue ="Checked"
ckbxcell.TextFalse ="Unchecked"
ckbxcell.TextIndeterminate ="Not Sure"
FpSpread1.ActiveSheet.Cells(0, 0).CellType = ckbxcell
```

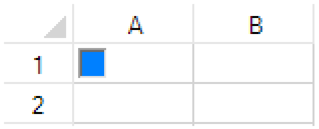
**Using the Spread Designer**

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.

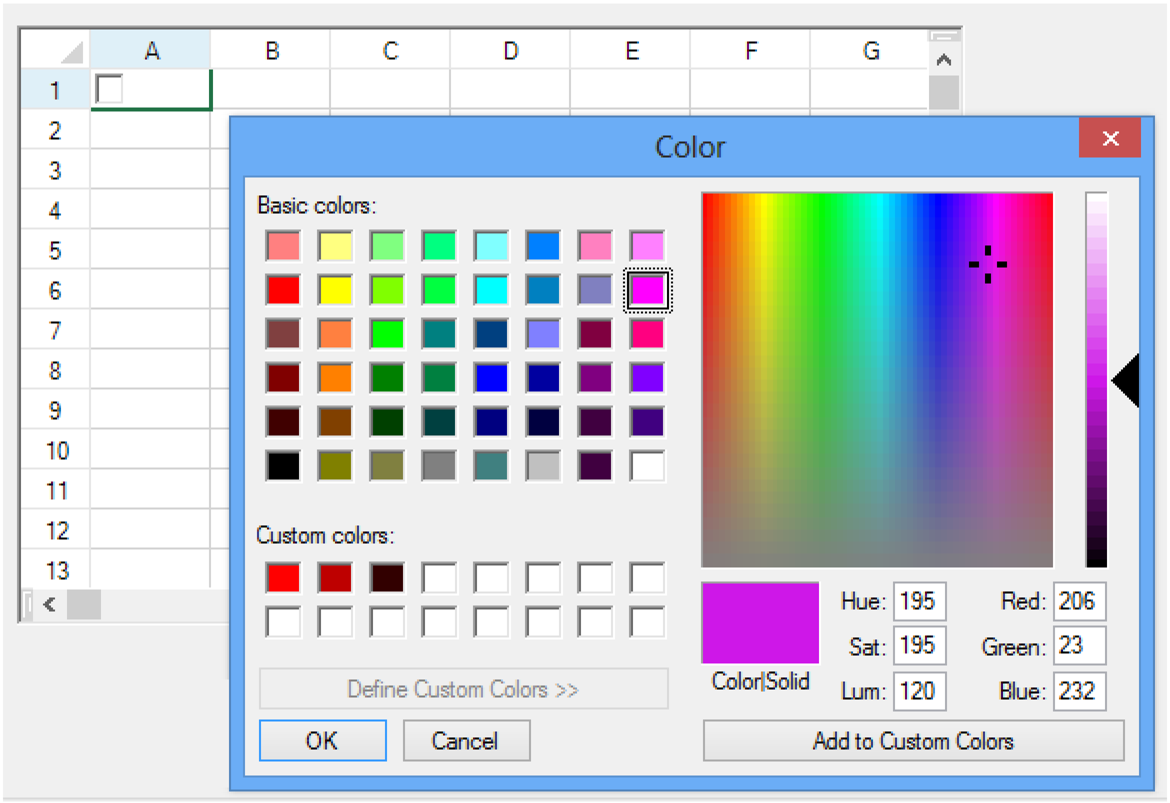
## Setting a Color Picker Cell

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.

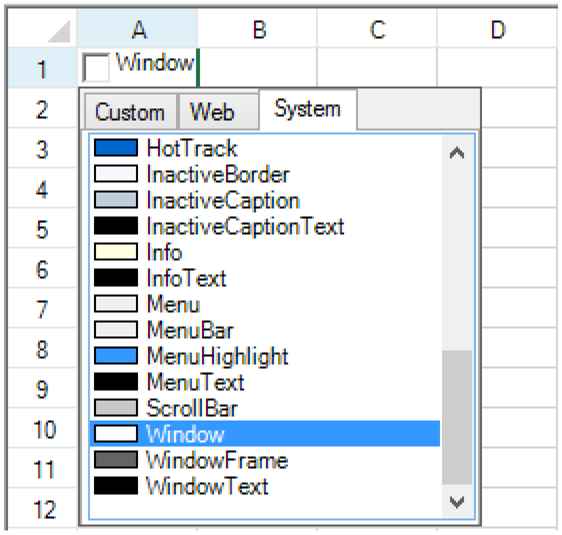


When the cell is double-clicked, either the drop-down color picker is displayed or the pop-up color dialog is displayed. There are several options for the display of the color dialog. The following figure shows the pop-up color dialog:



Hue: 195
Sat: 195
Lum: 120
Red: 206
Green: 23
Blue: 232

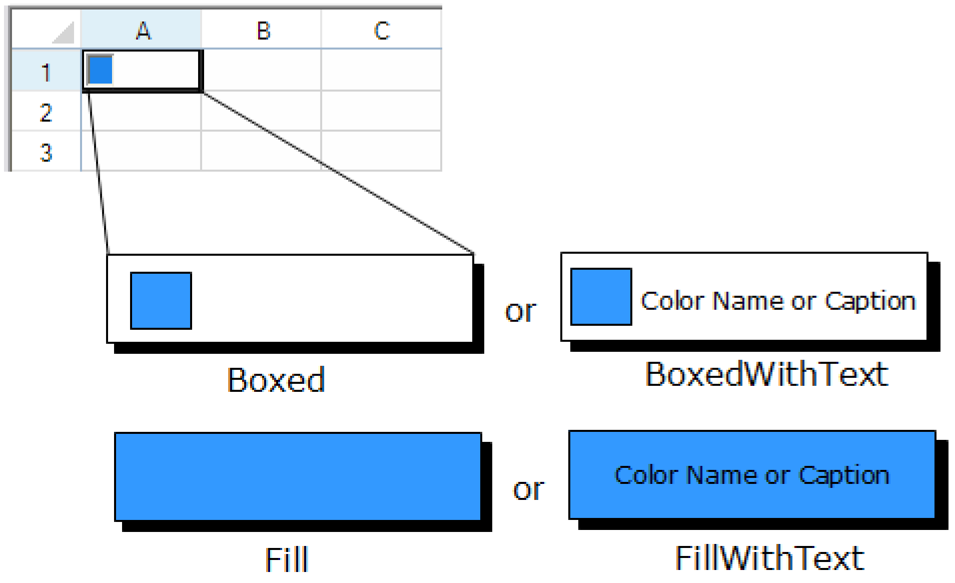
The following figure shows the drop-down color picker:



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.

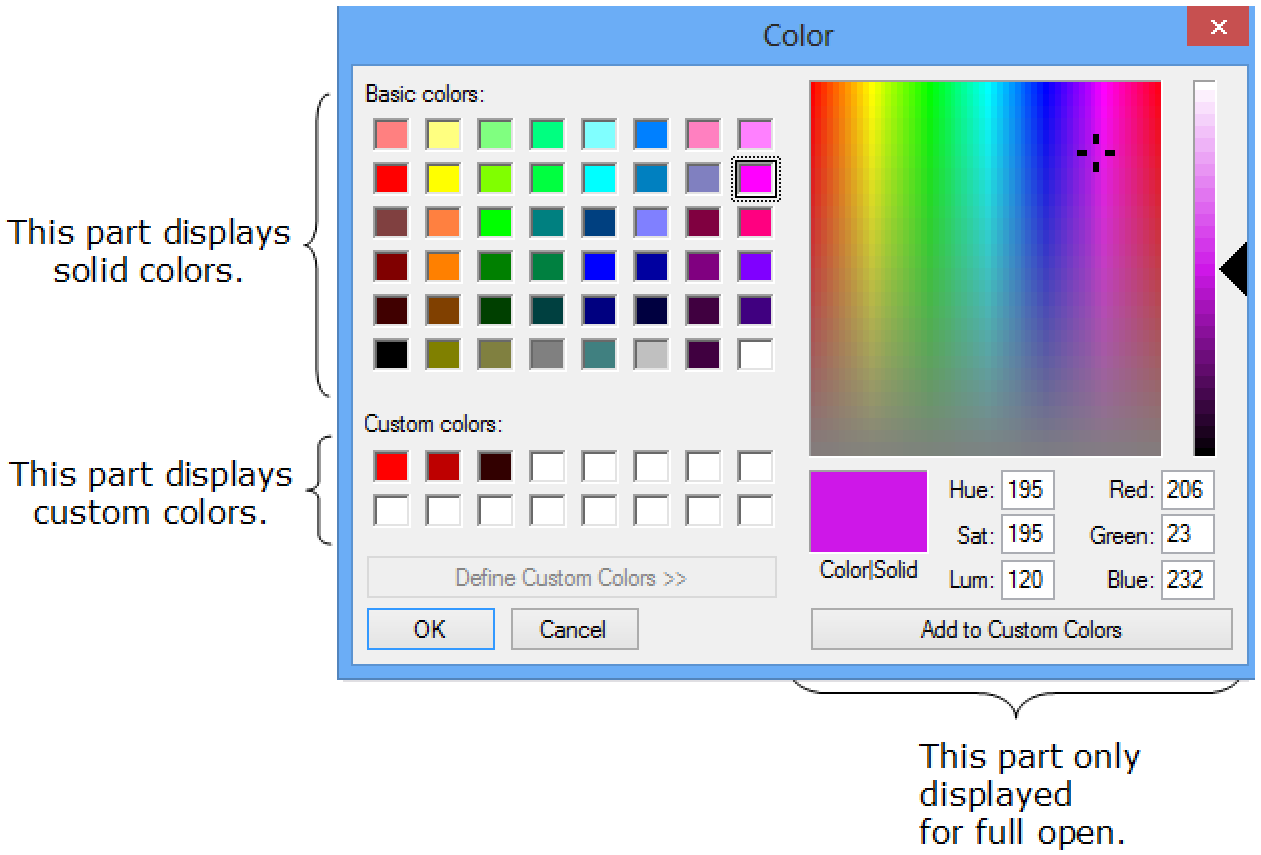


**Customizing the Color Dialog**

The color picker cell allows these customizations of the color dialog.

| Property | Description |
|---|---|
| **AllowFullOpen ('AllowFullOpen Property' in the on-line documentation)** | Sets whether to allow the color dialog to open fully to show the custom color selector. |
| **AnyColor ('AnyColor Property' in the on-line documentation)** | Sets whether the color dialog displays all available colors in the set of basic colors. |
| **Caption ('Caption Property' in the on-line documentation)** | Sets the text that appears in the cell (if any). |
| **CustomColors ('CustomColors Property' in the on-line documentation)** | Sets the custom colors shown in the color dialog. |
| **DialogShowing ('DialogShowing Property' in the on-line documentation)** | Sets whether to display the color dialog automatically. |
| **DropDown ('DropDown Property' in the on-line documentation)** | Sets whether to use the drop-down color picker (not the pop-up color dialog). |
| **FullOpen ('FullOpen Property' in the on-line documentation)** | Sets whether the color dialog opens fully to show controls used to create custom colors. |
| **SolidColorOnly ('SolidColorOnly Property' in the on-line documentation)** | Sets whether the color dialog restricts users to selecting solid colors only. |
| **Style ('Style Property' in the on-line documentation)** | Sets the style of the color dialog. |
| **UnknownText ('UnknownText Property' in the on-line documentation)** | Sets the text for an unknown color. |
| **UnknownTextStyle ('UnknownTextStyle Property' in the on-line documentation)** | 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
```

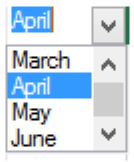
**Using the Spread Designer**

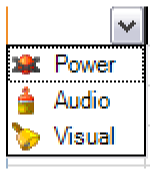
1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the **ColorPicker** cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Select and 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** | 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 | Description |
|---|---|
| **AutoCompleteCustomSource ('AutoCompleteCustomSource Property' in the on-line documentation)** | 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.

## 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:

- http://www.componentone.com
- www.clubFarPoint.com
- mailto:support@FarPointSpread.com?Subject=Spread Cell Test

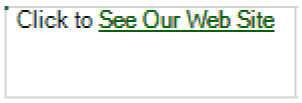
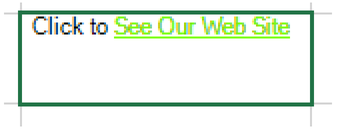
**Customizing Links**

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 and customize the color of the link that has been followed (visited or clicked).

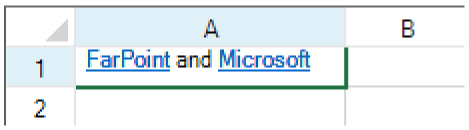
| Property | Customization |
|---|---|
| **BackgroundImage ('BackgroundImage Property' in the on-line documentation)** | Sets the background graphic image. |
| **Link ('Link Property' in the on-line documentation)** | Sets the destination URL. |
| **LinkArea ('LinkArea Property' in the on-line documentation)** | Sets the area of the text that is the hyperlink. |
| **LinkAreas ('LinkAreas Property' in the on-line documentation)** | Sets the area of the text that is the hyperlink. |
| **LinkColor ('LinkColor Property' in the on-line documentation)** | Sets the color of links (before they are followed). |
| **Links ('Links Property' in the on-line documentation)** | Sets the hyperlinks. |
| **Text ('Text Property' in the on-line documentation)** | Sets the label of the hyperlink, that is, what appears in the cell. |
| **VisitedLinkColor ('VisitedLinkColor Property' in the on-line documentation)** | Sets the color of followed links. |

**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 |
|---|---|
|  Click to See Our Web Site |  Click to See Our Web Site |

The following figure displays a hyperlink cell with multiple links.



FarPoint and Microsoft

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.

### 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.

### Example

This example sets the size of the cell (by column and row), creates a hyperlink button, and specifies the destination URL.

#### 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://www.componentone.com";
hlnkcell.LinkArea = new LinkArea(9,16);
hlnkcell.LinkColor = Color.DarkGreen;
hlnkcell.VisitedLinkColor = Color.Chartreuse;
fpSpread1.ActiveSheet.Cells[1, 1].CellType = hlnkcell;
```

#### 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://www.componentone.com"
hlnkcell.LinkArea = new LinkArea(9,16)
hlnkcell.LinkColor = Color.DarkGreen
hlnkcell.VisitedLinkColor = Color.Chartreuse
FpSpread1.ActiveSheet.Cells(1, 1).CellType = hlnkcell
```

**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.
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.

**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 = "FarPoint and Microsoft";
string[] s = new string[]{"www.fpoint.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 = "FarPoint and Microsoft"
Dim s() As String = New String() {"www.fpoint.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**

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 **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.

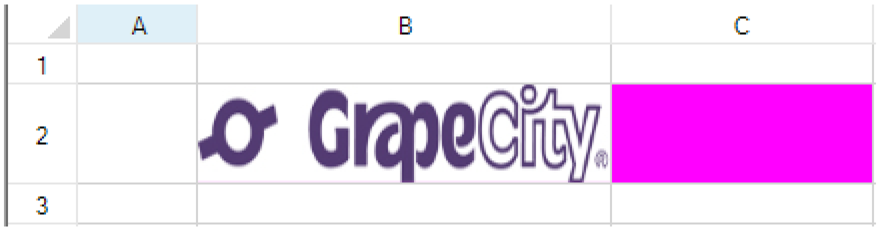
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

## 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.



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**

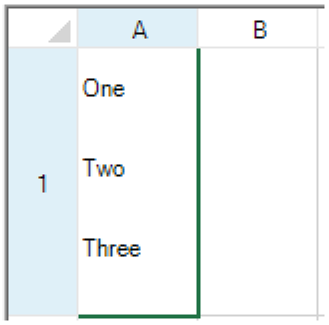
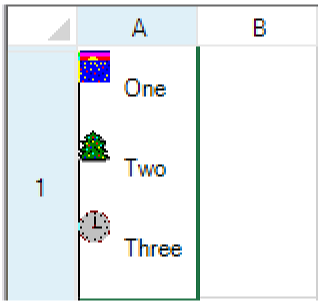
```
Dim imgct As New FarPoint.Win.Spread.CellType.ImageCellType()
Dim image As System.Drawing.Image = System.Drawing.Image.FromFile("D:\Logos\logo.jpg")
imgct.Style = FarPoint.Win.RenderStyle.Stretch
imgct.TransparencyColor = Color.Black
imgct.TransparencyTolerance = 20

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
```

**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**

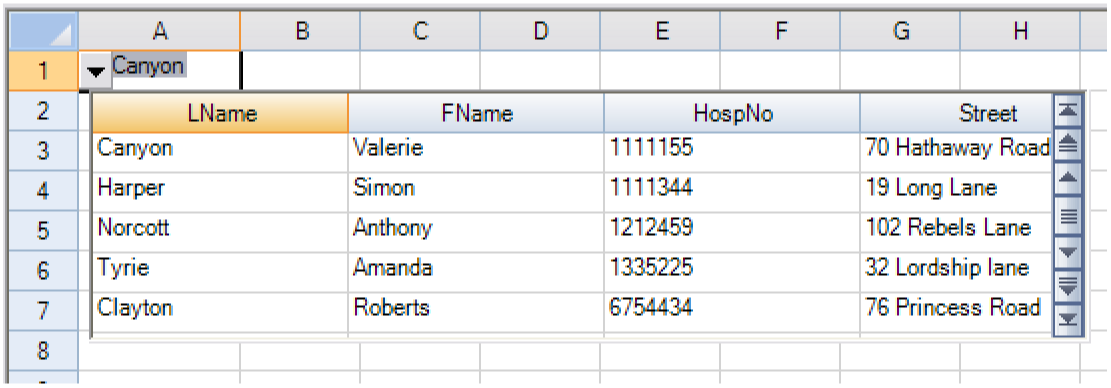
```
Dim listcell As New FarPoint.Win.Spread.CellType.ListBoxCellType()
listcell.ImageList = ImageList1
listcell.ItemData = New String() {"One", "Two", "Three"}
listcell.Items = New String() {"One", "Two", "Three"}
listcell.ItemHeight = 40
FpSpread1.ActiveSheet.Cells(0, 0).CellType = listcell
FpSpread1.ActiveSheet.Rows(0).Height = 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.



Canyon

| LName | FName | HospNo | Street |
|---|---|---|---|
| Canyon | Valerie | 1111155 | 70 Hathaway Road |
| Harper | Simon | 1111344 | 19 Long Lane |
| Norcott | Anthony | 1212459 | 102 Rebels Lane |
| Tyrie | Amanda | 1335225 | 32 Lordship lane |
| Clayton | Roberts | 6754434 | 76 Princess Road |

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 | Description |
|---|---|
| **BackgroundImage ('BackgroundImage Property' in the on-line documentation)** | Sets the background image in the cell. |
| **ButtonAlign ('ButtonAlign Property' in the on-line documentation)** | Sets where the buttons are displayed. |
| **ColumnEdit ('ColumnEdit Property' in the on-line documentation)** | Sets the column of the list to use for the edit portion. |
| **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. |
| **ListAlignment ('ListAlignment Property' in the on-line documentation)** | Sets which side of the editor the list aligns to. |
| **ListOffset ('ListOffset Property' in the on-line documentation)** | Sets how much the list offsets from the editor. |
| **ListWidth ('ListWidth Property' in the on-line documentation)** | Sets the width of the list. |
| **MaxDrop ('MaxDrop Property' in the on-line documentation)** | Sets the maximum number of items to display in the list at one time. |
| **StringTrim ('StringTrim Property' in the on-line documentation)** | Sets how to trim characters that do not fit in the cell. |
| **SubEditor ('SubEditor Property' in the on-line documentation)** | 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**

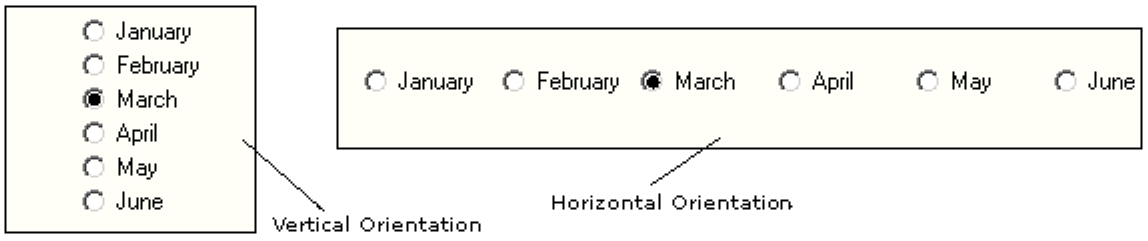
```
Dim conStr As String = "Provider=Microsoft.JET.OLEDB.4.0;data
source=C:\SpreadStudio\Common\Patients2000.mdb"
Dim sqlStr As String = "SELECT * FROM Patients"
Dim conn As New System.Data.OleDb.OleDbConnection(conStr)
Dim ds As DataSet = New DataSet()
Dim da As New System.Data.OleDb.OleDbDataAdapter(sqlStr, 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 | Description |
|---|---|
| **BackgroundImage ('BackgroundImage Property' in the on-line documentation)** | Sets the background image for the cell. |
| **EditorValue ('EditorValue Property' in the on-line documentation)** | Sets which value is written to the underlying data model. |
| **ItemData ('ItemData Property' in the on-line documentation)** | Sets the ItemData to use for the list. |
| **Items ('Items Property' in the on-line documentation)** | Creates the list to use for the option buttons. |
| **Orientation ('Orientation Property' in the on-line documentation)** | Sets the orientation of the option buttons. |
| **Picture ('Picture Property' in the on-line documentation)** | Customizes the option button images. |
| **TextAlign ('TextAlign Property' in the on-line documentation)** | Sets how text aligns in the cell. |
| **UseMnemonic ('UseMnemonic Property' in the on-line documentation)** | 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**

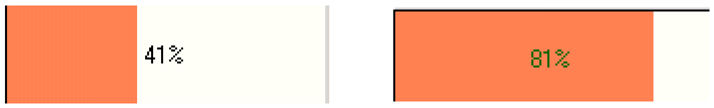
```
Dim multcell As New FarPoint.Win.Spread.CellType.MultiOptionCellType()
multcell.Items = new String() {"Carbon", "Oxygen", "Hydrogen"}
multcell.Orientation = FarPoint.Win.RadioOrientation.Horizontal
FpSpread1.ActiveSheet.Cells(0, 0).CellType = multcell
FpSpread1.ActiveSheet.Columns(0).Width = 220
```

**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

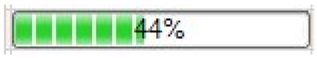
A progress indicator cell displays a progress indicator control across the entire cell. You can specify the color of the fill, the text to display, the color of the text and other properties.



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 | Description |
|---|---|
| **BackgroundImage ('BackgroundImage Property' in the on-line documentation)** | Sets the background image for the cell. |
| **FillColor ('FillColor Property' in the on-line documentation)** | Sets the color to use for the filled part of the progress indicator. |
| **FillColor2 ('FillColor2 Property' in the on-line documentation)** | Sets the second fill color to use for the gradient part of the progress indicator. |
| **FillTextColor ('FillTextColor Property' in the on-line documentation)** | Sets the color to use for the text in the filled part of the indicator. |
| **GradientMode ('GradientMode Property' in the on-line documentation)** | Sets the gradient mode for a gradient style progress indicator. |
| **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 progress bar. |
| **Picture ('Picture Property' in the on-line documentation)** | Sets the image to use for the progress bar when the style is set to Picture. |
| **ShowText ('ShowText Property' in the on-line documentation)** | Sets whether the percent filled string is displayed. |
| **Style ('Style Property' in the on-line documentation)** | Sets the style of the progress bar(s). |
| **Text ('Text Property' in the on-line documentation)** | Sets the string to use when **TextStyle** is set to Custom. |
| **TextStyle ('TextStyle Property' in the on-line documentation)** | 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

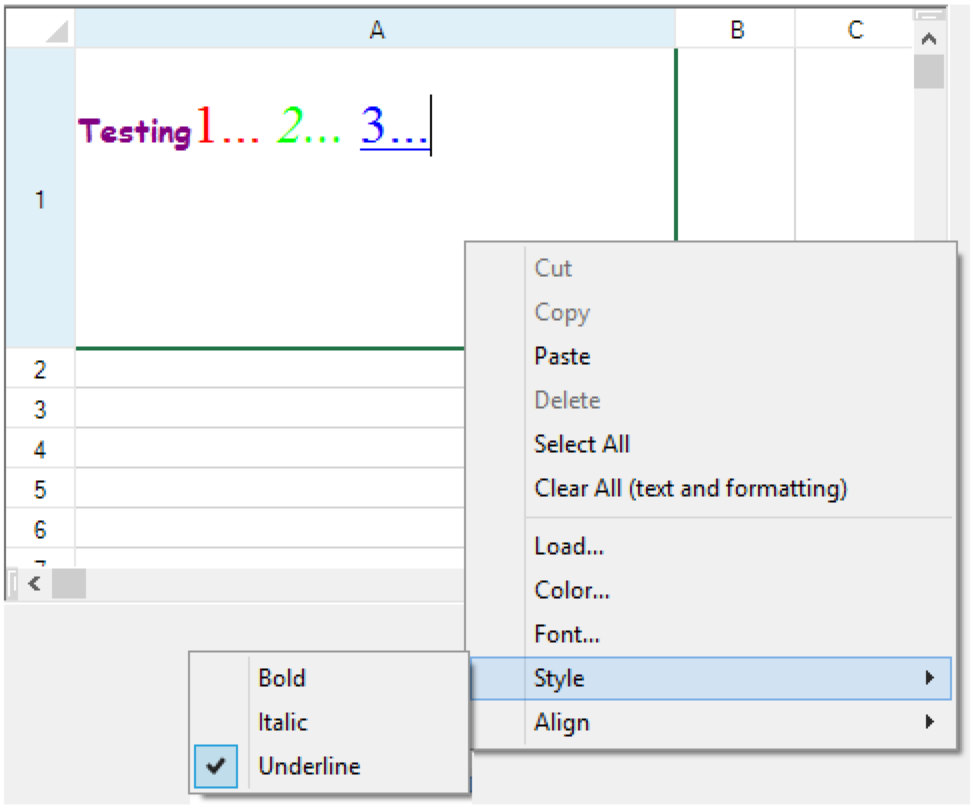
1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the **Progress** cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Select and set 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**.

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

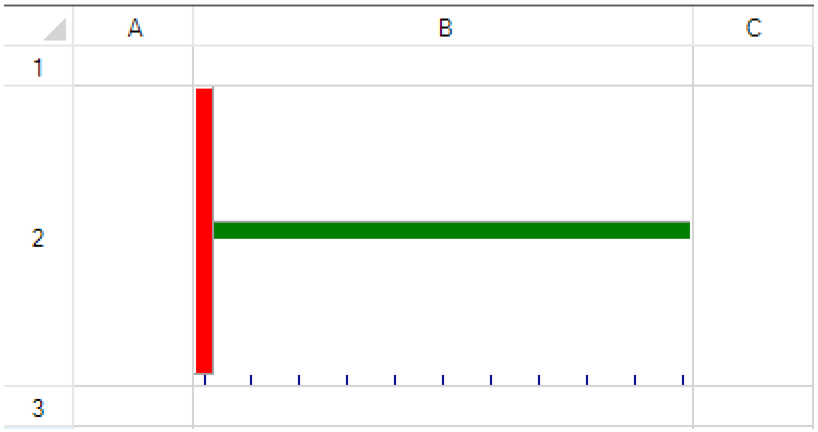
```
Dim f as System.IO.TextReader = System.IO.File.OpenText("your_file.rtf")
Dim bits As String
bits = f.ReadToEnd()
f.Close()
FpSpread1.ActiveSheet.Cells(0, 0).CellType = New
FarPoint.Win.Spread.CellType.RichTextCellType()
FpSpread1.ActiveSheet.Cells(0, 0).Value = bits
```

### 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.

## Setting a Slider Cell

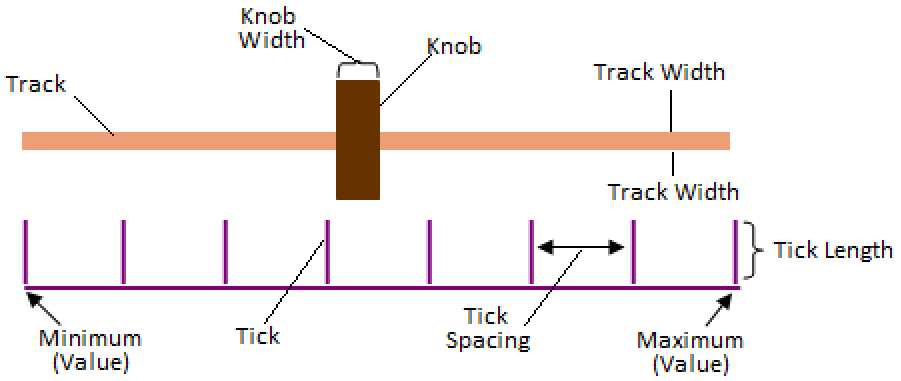
A slider cell displays a slider control in the cell.



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 | Customization |
|---|---|
| **BackgroundImage ('BackgroundImage Property' in the on-line documentation)** | Sets the background image for the cell. |
| **ChangeOnFocus ('ChangeOnFocus Property' in the on-line documentation)** | Sets whether the slider moves with the initial click. |
| **KnobColor ('KnobColor Property' in the on-line documentation)** | Sets the color of the slider knob. |
| **KnobPicture ('KnobPicture Property' in the on-line documentation)** | 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** | Sets the orientation of the slider. |

| | |
|---|---|
| **documentation)** | |
| **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 miminum 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.Picture(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

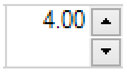
You can specify the cell type for individual cells, columns, rows, a range of cells, or an entire sheet. For any cell type there are properties of a cell that can be set. In general, working with cell types includes defining the cell type, setting the properties, and applying that cell type to cells.

The following topics provide additional information about customizing cells based on cell types:

- **Displaying Spin Buttons**
- **Allowing a Combo Box Cell to Handle a Double Click**
- **Limiting Values for a Numeric Cell**
- **Customizing the Pop-Up Date-Time Control**
- **Customizing the Pop-Up Calculator Control**
- **Customizing Automatic Completion (Type Ahead)**
- **Working with a SubEditor**
- **Creating a Custom Cell Type**

## 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 = 443.3482
```

### 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.

## 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
```

## 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).

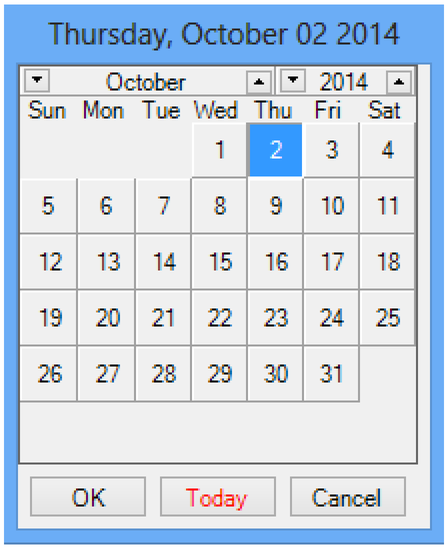
From the drop-down list, choose **Currency** or other numeric cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type.

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.

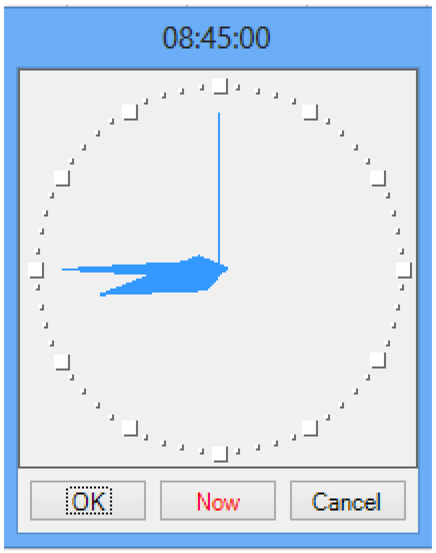
## Customizing the Pop-Up Date-Time Control

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**

```
Dim datecell As New FarPoint.Win.Spread.CellType.DateTimeCellType()
Dim daynames() As String = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday" }
Dim months() As String = {"January", "February", "March", "April", "May", "June",
"July", "August", "September", "October", "November", "December"}
Dim dayabbrev() As String = {"Su","My","Ty","Wy","Th","Fy","Sy"}
Dim mthabbrev() As String =
{"Jy","Fy","Mh","Al","My","Jn","Jl","At","Sr","Or","Nr","Dr"}
Dim okbuttn As String = "Fine"
Dim cancelb As String = "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

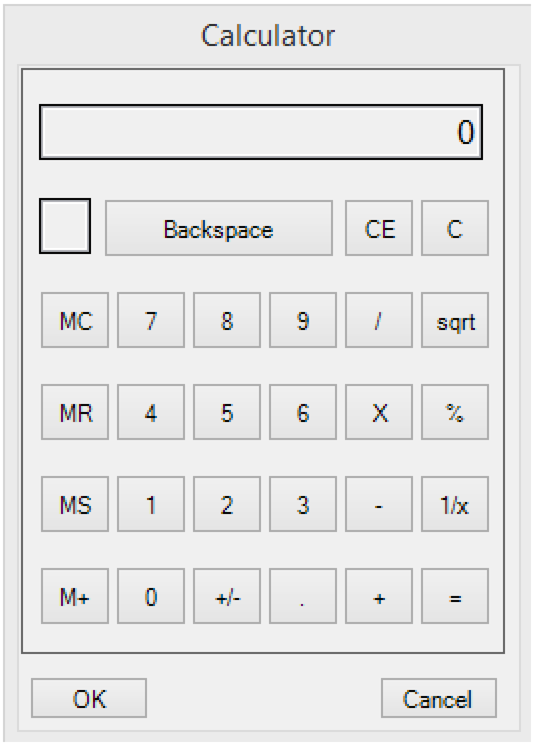
1. Select the cell or cells in the work area.
2. In the property list, in the **Misc** category, select **CellType**. From the drop-down list, choose the **DateTime** cell type. Now expand the **CellType** property and various properties are available that are specific to this cell type. Set properties 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.

## 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#**

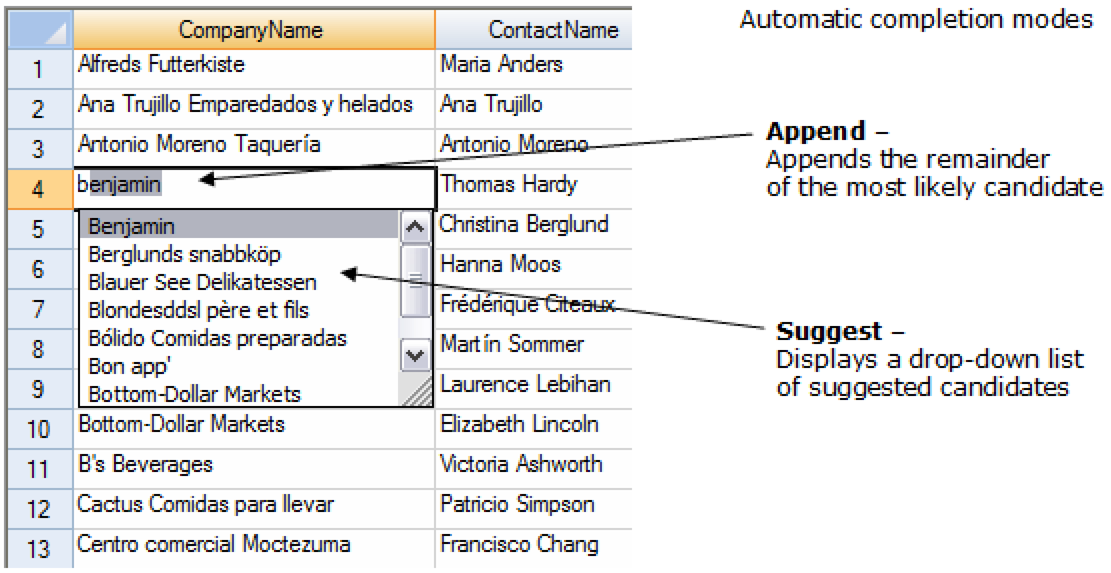
```
FarPoint.Win.Spread.CellType.CurrencyCellType ctest = new
FarPoint.Win.Spread.CellType.CurrencyCellType();
ctest.SetCalculatorText("Accept", "Cancel");
fpSpread1.Sheets[0].Cells[0, 0].CellType = ctest;
```

**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.



Automatic completion modes

| | 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 | Martín Sommer |
| 9 | Bólido Comidas preparadas | Laurence Lebihan |
| | Bon app' | |
| | Bottom-Dollar Markets | |
| 10 | Bottom-Dollar Markets | Elizabeth Lincoln |
| 11 | B's Beverages | Victoria Ashworth |
| 12 | Cactus Comidas para llevar | Patricio Simpson |
| 13 | Centro comercial Moctezuma | Francisco Chang |

**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 http://www.componentone.com/SuperProducts/SpreadStudio/Demos/.

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.

For 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;
}
```

**VB**

```
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

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
```

## 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**
- **Using Drag Operations to Fill Cells**
- **Using Visible Indicators in the Cell**
- **Using Conditional Formatting of Cells**
- **Managing Formulas in Cells**
- **Using the Additional Spread Controls**

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 **Customizing Interaction with Cell Types**.

You can also determine how the user interacts with the keyboard. Refer to **Managing Keyboard Interaction**.

## Using Edit Mode and Focus

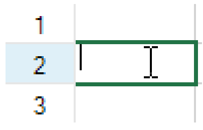
Important aspects of working with cells, and with user interaction at a cell level, include the use of edit mode and focus. The following topics describe how you can customize interaction including edit mode and focus:

- **Understanding Edit Mode in a Cell**
- **Locking a Cell**
- **Allowing the Display of Buttons in a Cell**
- **Customizing the Focus Indicator for a Cell**

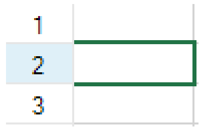
## Understanding Edit Mode in a Cell

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 ('Locked Property' in the on-line documentation)** 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.

## 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.

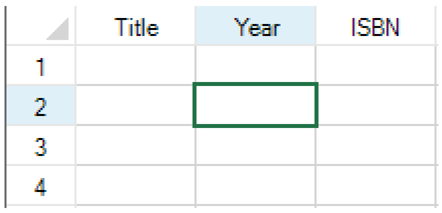
For details on setting the display of primary and secondary buttons as part of a cell’s appearance, see the **Appearance ('Appearance Class' in the on-line documentation)** class **DrawPrimaryButton ('DrawPrimaryButton Property' in the on-line documentation)** property and **DrawSecondaryButton ('DrawSecondaryButton Property' in the on-line documentation)** property.

You can also make use of the FpSpread **ButtonClicked ('ButtonClicked Event' in the on-line documentation)** event.

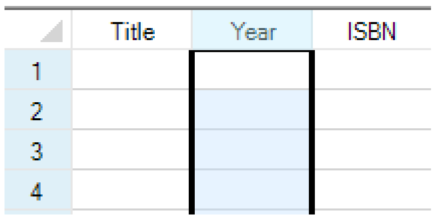
For more information about cell types that allow buttons to be drawn in a cell, refer to **Working with Graphical Cell Types**. For more information about cell types that can display spin buttons, see **Displaying Spin Buttons**.

## 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.



Focus indicator for selected individual cell



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
```

## 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.

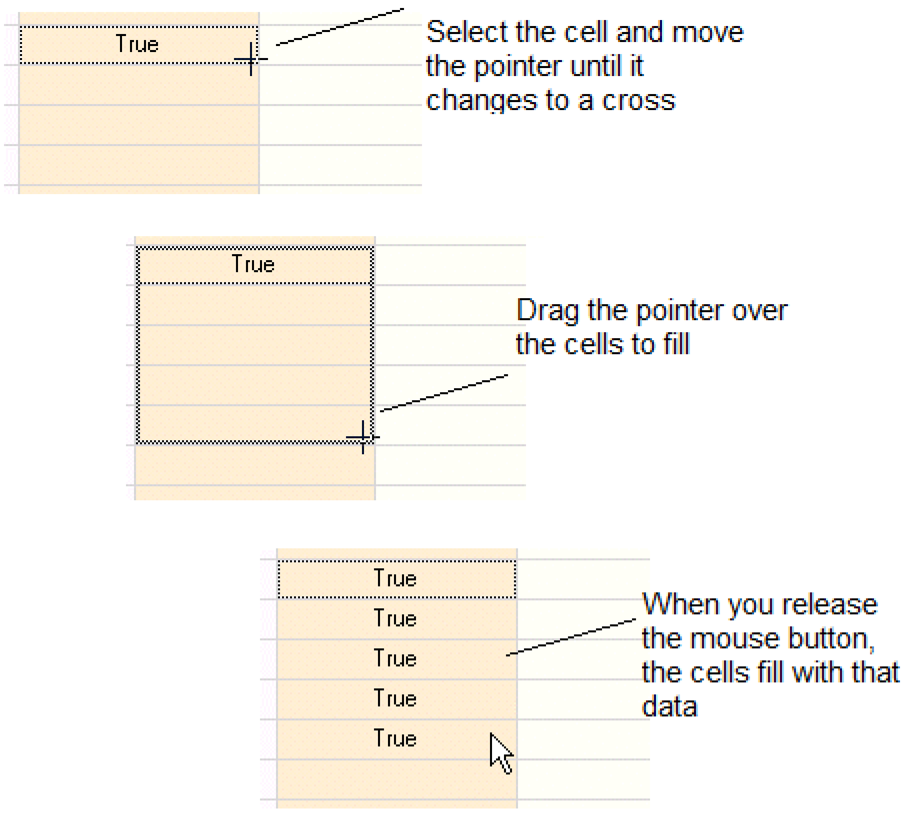
## Filling Cells with Drag and Fill

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).

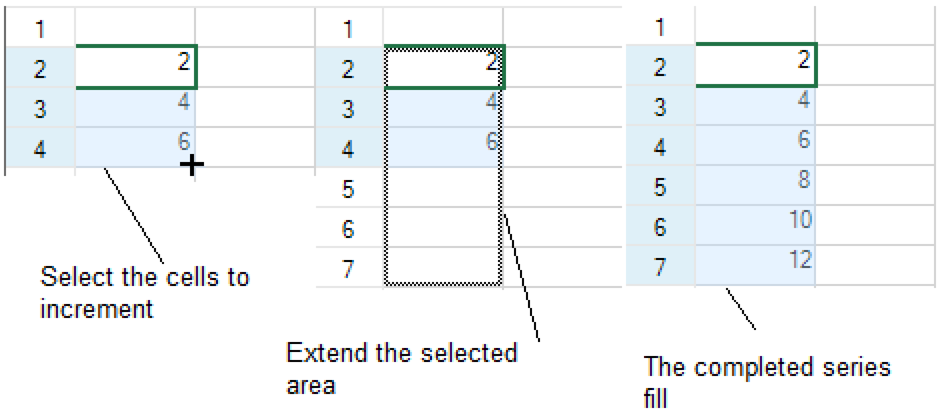
You can also specify whether the fill is a copy or a series. The copy option copies the data to the selected range. The series option increments or decrements the values based on the series. Drag down or to the right to increment the values. Drag up or to the left to decrement the values. Use the **RangeDragFillMode ('RangeDragFillMode Property' in the on-line documentation)** property to specify the type of fill.

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 example shown here uses the copy option to fill several cells in a column from the originally selected cell.


True
Select the cell and move the pointer until it changes to a cross
True
Drag the pointer over the cells to fill
True
True
True
True
True
When you release the mouse button, the cells fill with that data

This example uses the series fill option.



You can customize the direction of the fill using the **FillDirection ('FillDirection Enumeration' in the on-line documentation)** enumeration.

For more information, refer to the **AllowDragFill ('AllowDragFill 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 **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.

**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.

**Example**

This example code sets the component to allow the drag-fill feature.

**C#**

```
fpSpread1.AllowDragFill = true;
```

**VB**

```
FpSpread1.AllowDragFill = True
```

**Using Code**

Allow the drag-and-drop feature by setting the **AllowDragFill ('AllowDragFill 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.

**C#**

```
FarPoint.Win.Spread.SpreadView sv = fpSpread1.GetRootWorkbook();
sv.AllowDragFill = true;
```

**VB**

```
Dim sv As FarPoint.Win.Spread.SpreadView = FpSpread1.GetRootWorkbook
sv.AllowDragFill = 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 fill 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 **AllowDragFill** 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 Move

You can drag and move entire rows or columns of cells within a given sheet.

For more information, refer to the **Allowing the User to Move 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 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:

- **Displaying Text Tips in a Cell**
- **Adding a Note to a Cell**
- **Preventing a Cell from Receiving Focus**
- **Returning Information for a Clicked Cell**
- **Displaying Error Icons in Cells or Rows**

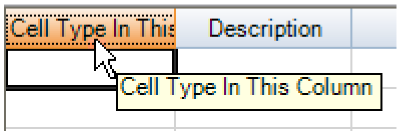
As with most spreadsheets, Spread does not allow in-cell editing of the cells in the row and column headers.

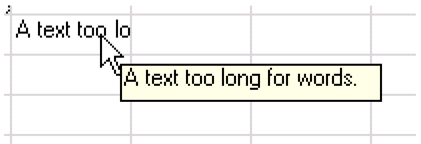
For more information about cells and sheet interactions, refer to the following topics:

| Description | Refer to |
|---|---|
| For information on adding a formula to a cell | **Placing a Formula in Cells** |
| For information on allowing a user to enter a formula in a cell | **Allowing the User to Enter Formulas** |
| For information on how to change the appearance of cells | **Customizing the Appearance of a Cell** |
| For information on setting the cell type | **Customizing Interaction with Cell Types** |
| For information about customizing focus indicators | **Customizing the Focus Indicator for a Cell** |
| For information on customizing the interaction at a sheet level | **Customizing Interaction with a Sheet** |

## 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 of the Component**.

### 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")
```

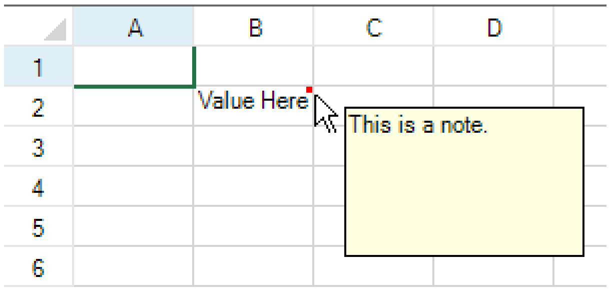
```
FpSpread1.TextTipPolicy = FarPoint.Win.Spread.TextTipPolicy.Floating
FpSpread1.TextTipDelay = 1000
MessageBox.Show("Place the pointer over the text to see the text tip.", "",
MessageBoxButtons.OK)
```

**Using the Spread Designer**

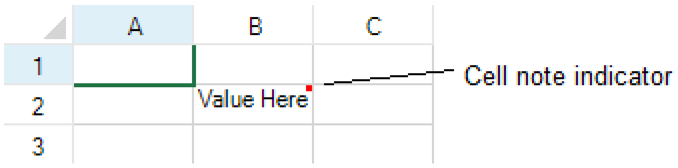
1. Select the Spread component (or select **Spread** from the pull-down menu).
2. In the property list, in the **Behavior** category, select the **TextTipPolicy** property.
3. Click the drop-down arrow to display the choices and choose a value.
4. If you want to set the delay, select the **TextTipDelay** 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.

## 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.

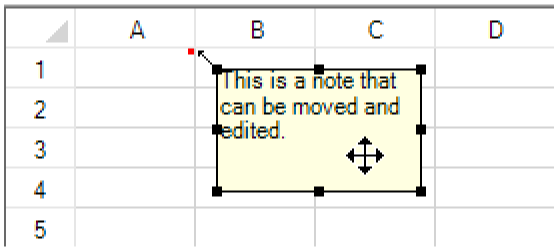


The red square in the upper right corner of the cell indicates that a note is available for that cell. An example is shown in the following figure. You can use the **CellNoteIndicatorVisible ('CellNoteIndicatorVisible Property' in the on-line documentation)** property to hide the cell note indicator 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.

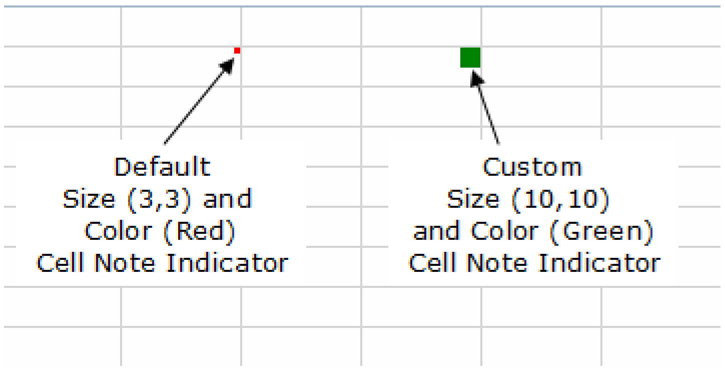
**Understanding Limitations**

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.

## 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.

## 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.

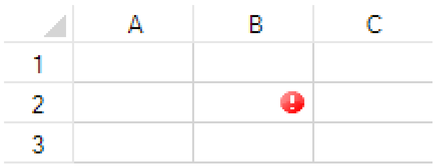
For headers, you can get row and column index information of clicked header cells by accessing **CellClick** event parameter *e* in the **CellClickEventArgs** class. You can detect whether the headers have been clicked. You can get *x*- and *y*-coordinates as well from this parameter. You can implement 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.

## 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.



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 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 ('ConditionalFormatting Class' in the on-line documentation)** class.

The following topics provide additional information about specific conditional formatting rules.

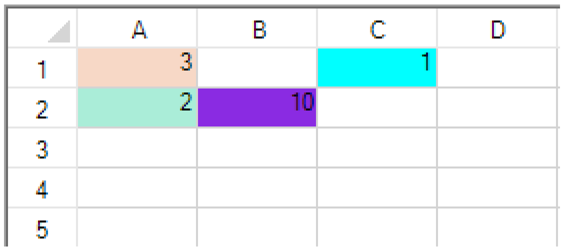
- **Color Scale Rules**
- **Data Bar Rule**
- **Highlighting Rules**
- **Icon Set Rule**
- **Top, Bottom, or Average Rules**

## 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
```

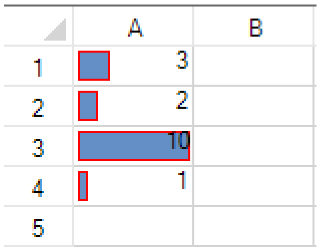
**Using the Spread Designer**

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 Value | The maximum value in the range of cells that the conditional formatting rule applies to. |
| 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.

## 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

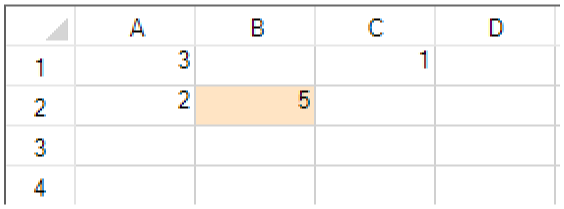
After you choose one of the options above, enter a value or formula against which each cell is compared. If the cell data satisfies that criteria, then the formatting is applied.

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)**

- **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

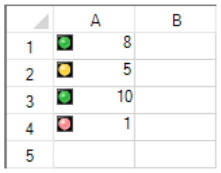
1. In the work area, select the cell or cells for which you want to set the conditional format.
2. Under the **Home** menu, select the **Conditional Formatting** icon in the **Style** section, then select the **Highlight Cells Rules** option, and then choose the condition.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

## 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
```

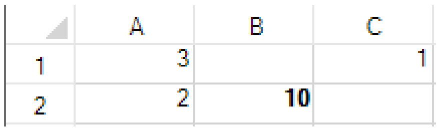
**Using the Spread Designer**

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.

## 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 use a named style to 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 a complete list of operations, refer to the **ComparisonOperator ('ComparisonOperator Enumeration' in the on-line documentation)** enumeration. For a list of the types of expressions, refer to the **CalcEngine.Expression** object. For more information about the possible style settings, refer to **Creating and Applying a Style for Cells**.

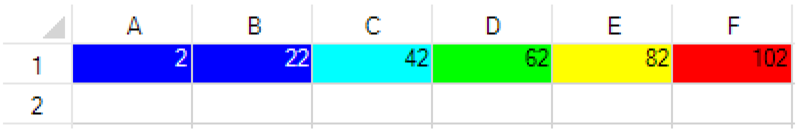
If two conditional formats are set to the same cell, the second conditional format takes effect.

The conditional formatting can be done using the **ConditionalFormat ('ConditionalFormat Class' in the on-line documentation)** class and any of these members of the **SheetView ('SpreadView Class' in the on-line documentation)** class:

- **GetConditionalFormats** method
- **SetConditionalFormat** methods
- **ClearConditionalFormats** method

When you use the **GetConditionalFormat** methods, the conditions, operator, and style information are returned as a **ConditionalFormat** 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 a **NamedStyle** 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.

You can also use these members of the **IConditionalFormatSupport ('IConditionalFormatSupport Interface' in the on-line documentation)** interface:

- **ConditionalFormatIsRowUsed ('ConditionalFormatIsRowUsed Method' in the on-line documentation)** method
- **ConditionalFormatNextNonEmptyColumnInRow ('ConditionalFormatNextNonEmptyColumnInRow Method' in the on-line documentation)** method

Use the **ClearConditionalFormats** method to clear only the conditional formats of a cell without affecting the other formatting or the contents of the cell.

Use the **Model.DefaultSheetStyleModel.ConditionalFormatIsRowUsed** method to determine whether a row in the style model contains style settings.

Use the **Model.DefaultSheetStyleModel.ConditionalFormatNextNonEmptyColumnInRow** method to return the index of the next non-empty column in a row in the model.Use the **Model.DefaultSheetStyleModel.GetValidConditionalFormat** method to return the style information for the first valid condition for the cell at the specified row and column in the model.

### Using Code

1. Define styles.
2. Set a conditional format for a cell.

### Example

This example code sets the conditional format for a cell based on the numeric value of the data. It changes the coloring of the cell based on the value of temperature data. To see how it works, type a number in cell B2, then either change cells or leave edit mode to see the formatting 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
```

## Managing Formulas in Cells

## 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 $C$2+$D$2, 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"
```

**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.

## 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 ("B0"). The component treats the value in the cell "B0" 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 "B0". 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 does not support range references where the start row and end row consist of different reference types (for example, one absolute coordinate and one relative coordinate).

Either both row coordinates must be absolute or both row coordinates must be relative. For example:

| Reference | Whether Supported |
|---|---|
| R1C[-1]:R5C[-1] | supported (absolute row : absolute row) |
| R1C[-1]:RC[-1] | not supported (absolute row : relative row) |
| RC[-1]:R5C[-1] | not supported (relative row : absolute row) |
| R[-5]c[-1]:RC[-1] | supported (relative row : relative row) |

Develop your formulas so that either both row coordinates are absolute,

```
for (int i = 0; i < n; i++)
fpspread1.Sheets[0].Cells[i,column].Formula = "SUM(R1C[-1]:R" +       (i+1).ToString() + "C[-1])"
```

or both row coordinates are relative,

```
for (int i = 0; i < n; i++)
fpspread1.Sheets[0].Cells[i,column].Formula = "SUM(R[" +       (-i).ToString() + "]C[-1]:RC[-1])"
```

The same restrictions apply to start column and end column coordinates.

For more information on cell 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 is False, each worksheet functions as a separate calculation unit with each worksheet having its own calculation settings (auto calculations, iterations, custom functions, custom names, and so on). Changing a calculation setting affects a single worksheet. For this setting of **EnableCrossSheetReference ('EnableCrossSheetReference Property' in the on-line documentation)**, formulas can only reference cells on the same worksheet.

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"
```

## 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. 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 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. The exact order in which the circular formulas are evaluated during a given iteration cannot be assumed by the application. As with most spreadsheet products (including Excel and OpenOffice), circular formulas in Spread are intended to be used in scenarios where the iterations converge to the desired solution regardless of the order of evaluation within a given iteration.

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")
```

## 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)** and **RecalculateAll ('RecalculateAll Method' in the on-line documentation)** methods 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 **RecalculateAll ('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.RecalculateAll();
```

**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.RecalculateAll()
```

## 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

In many of the cell types, users can type in a formula by simply starting with an equals sign (=). You do not need to set a property to allow this. How the result is displayed 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. For currency and date cells, specific formatting can be defined. For more information on 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 **Setting up the Formula Text Box**.

**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 a Shortcut**

Allow the user to enter formulas by setting the **AllowUserFormulas ('AllowUserFormulas Property' in the on-line documentation)** property for the component or the sheet.

**Example**

This example code sets the Spread component to allow users to enter formulas.

**C#**

```
fpSpread1.AllowUserFormulas = true;
```

**VB**

```
FpSpread1.AllowUserFormulas = True
```

**Using Code**

Allow the user to enter formulas by setting the **AllowUserFormulas ('AllowUserFormulas Property' in the on-line documentation)** property for the component or the sheet.

**Example**

This example code sets the child sheet to allow users to enter formulas.

**C#**

```
FarPoint.Win.Spread.SpreadView sv = fpSpread1.GetRootWorkbook();
sv.AllowUserFormulas = true;
```

**VB**

```
Dim sv As FarPoint.Win.Spread.SpreadView = FpSpread1.GetRootWorkbook
sv.AllowUserFormulas = True
```

**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**.
3. 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. Use the **AddModelScopeCustomName ('AddModelScopeCustomName Method' in the on-line documentation)** method to add sheet level custom names.

Avoid using custom names that start with C# or R# patterns (# stands for any number).

**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#

```
FarPoint.Win.Spread.Model.DefaultSheetDataModel d = new
FarPoint.Win.Spread.Model.DefaultSheetDataModel();
d.AddCustomName("test", "$B$1", 0, 0);
```

### VB

```
Dim d As FarPoint.Win.Spread.Model.DefaultSheetDataModel = New
FarPoint.Win.Spread.Model.DefaultSheetDataModel()
d.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#

```
FarPoint.Win.Spread.Model.DefaultSheetDataModel d;
d = (FarPoint.Win.Spread.Model.DefaultSheetDataModel)FpSpread1.Sheets[0].Models.Data;
d.AddCustomName("alpha", "101", 0, 0);
```

### VB

```
Dim d As FarPoint.Win.Spread.Model.DefaultSheetDataModel
d = DirectCast(FpSpread1.Sheets(0).Models.Data,
FarPoint.Win.Spread.Model.DefaultSheetDataModel)
d.AddCustomName("alpha", "101", 0, 0)
```

The following example adds a name that is a range reference.

### C#

```
FarPoint.Win.Spread.Model.DefaultSheetDataModel d;
d = (FarPoint.Win.Spread.Model.DefaultSheetDataModel)FpSpread1.Sheets[0].Models.Data;
d.AddCustomName("Sales", "Sheet1!$F$20:$F$50", 0, 0);
```

### VB

```
Dim d As New FarPoint.Win.Spread.Model.DefaultSheetDataModel
d = DirectCast(FpSpread1.Sheets(0).Models.Data,
FarPoint.Win.Spread.Model.DefaultSheetDataModel)
d.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 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)** and **MaxArgs ('MaxArgs Property' in the on-line documentation)** properties 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 **AcceptsError ('AcceptsError Method' in the on-line documentation)** method of the function returns False for that parameter, then the **Evaluate** method of the function is skipped and the error value is used as the result.

The custom function's **Evaluate ('Evaluate Method' in the on-line documentation)** method does not receive any information regarding the location (or context) in which the formula is being evaluated. If your custom function needs the row and column in which it is being evaluated then you must add extra parameters to your custom function and manually pass the row and column coordinates in the extra parameters.

### 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 create three functions: a cube mathematical function, an XOR logical function, and a null string function. The following code implements the custom functions.

The CUBE custom function raises a number to the third power. That is, CUBE(x) is equivalent to POWER($x^3$).

**C#**

```
public class CubeFunctionInfo : FarPoint.CalcEngine.FunctionInfo
{
public override string Name { get { return "CUBE"; } }
public override int MinArgs { get { return 1; } }
public override int MaxArgs { get { return 1; } }
public override object Evaluate (object[] args)
{
double num = FarPoint.CalcEngine.CalcConvert.ToDouble(args[0]);
return num * num * num;
}
}
```

The XOR custom function performs an exclusive OR operation on two Boolean values. This is similar to the "^" operator in C or the Xor operator in VB.

**C#**

```
public class XorFunctionInfo : FunctionInfo
{
public override string Name { get { return "XOR"; } }
public override int MinArgs { get { return 2; } }
public override int MaxArgs { get { return 2; } }
public override object Evaluate (object[] args)
{
bool arg0 = CalcConvert.ToBool(args[0]);
bool arg1 = CalcConvert.ToBool(args[1]);
return (arg0 || arg1) && (arg0 != arg1);
}
}
```

The NULL function returns the constant value null similar to how the FALSE() function returns the constant value false.

**C#**

```
public class NullFunctionInfo : FunctionInfo
{
public override string Name { get { return "NULL"; } }
public override int MinArgs { get { return 0; } }
public override int MaxArgs { get { return 0; } }
public override object Evaluate (object[] args)
{
return null;
}
}
```

The following code registers the custom functions.

**C#**

```
fpSpread1.ActiveSheet.AddCustomFunction(new CubeFunctionInfo());
fpSpread1.ActiveSheet.AddCustomFunction(new XorFunctionInfo());
fpSpread1.ActiveSheet.AddCustomFunction(new NullFunctionInfo());
```

The following code uses the customs in formulas.

**C#**

```
fpSpread1.ActiveSheet.SetFormula(0, 0, "CUBE(5)");
fpSpread1.ActiveSheet.SetFormula(1, 0, "XOR(FALSE,FALSE)");
fpSpread1.ActiveSheet.SetFormula(1, 1, "XOR(TRUE,FALSE)");
fpSpread1.ActiveSheet.SetFormula(1, 2, "XOR(FALSE,TRUE)");
fpSpread1.ActiveSheet.SetFormula(1, 3, "XOR(TRUE,TRUE)");
fpSpread1.ActiveSheet.SetFormula(2, 0, "CHOOSE(1,100,NULL(),300)");
fpSpread1.ActiveSheet.SetFormula(2, 1, "CHOOSE(2,100,NULL(),300)");
fpSpread1.ActiveSheet.SetFormula(2, 2, "CHOOSE(3,100,NULL(),300)");
```

**Parameters in Custom Functions**

By default, parameters are passed by value. 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). A cell range is passed as an instance of the **CalcArray ('CalcArray Class' in the on-line documentation)** class. The **CalcArray ('CalcArray Class' in the on-line documentation)** class has **RowCount** and **ColumnCount** properties for determining the number of rows and columns in the two dimensional array. The **CalcArray** class has a **GetValue ('GetValue Method' in the on-line documentation)** method for getting a single value from of the array. The row and column indexes to the **GetValue** method are zero based.

If you want a parameter passed by reference, then you must override the **AcceptsReference ('AcceptsReference Method' in the on-line documentation)** method in the **FunctionInfo ('FunctionInfo Class' in the on-line**

**documentation)** class. When the **AcceptsReference** method returns True for a parameter, a single cell or a cell range is passed as an instance of the **CalcReference ('CalcReference Class' in the on-line documentation)** class. The **CalcReference ('CalcReference Class' in the on-line documentation)** class has **Row** and **Column** properties for determining the first row and column in the reference. The **CalcReference** class has **RowCount** and **ColumnCount** properties for determining the number of rows and columns in the reference. The **CalcArray** class has a **GetValue** method for getting a single value from the reference. The row and column indexes for the **GetValue** method start at the row and column.

**Example**

In this example, a function counts the number of cells in a range that are less than a given criteria.

**C#**

```
class CountIfLessThanFunctionInfo : FunctionInfo
  {
    public override string Name
    {
        get { return "COUNTIFLESSTHAN"; }
    }
    public override int MinArgs
    {
        get { return 2; }
    }
    public override int MaxArgs
    {
        get { return 2; }
    }
    public override bool AcceptsReference(int i)
    {
        return i == 0;
    }
    public override object Evaluate(object[] args)
    {
        CalcReference range = args[0] as CalcReference;
        double criteria = CalcConvert.ToDouble(args[1]);
        double count = 0.0;
        if (range == null)
            return CalcError.Value;
        for (int i = range.Row; i < range.Row + range.RowCount; i++)
        {
            for (int j = range.Column; j < range.Column +
range.ColumnCount; j++)

      double cellValue = CalcConvert.ToDouble(range.GetValue(i, j));
            if (cellValue < criteria)
                    count++;
        }
    }
        return count;
  }
}
```

## Using the Additional Spread Controls

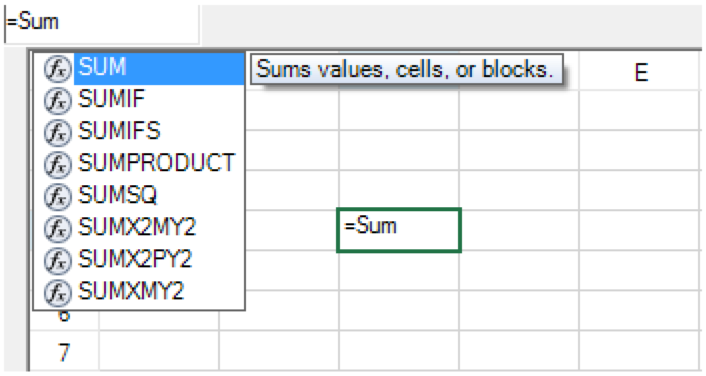
You can extend the functionality provided in Spread to end users through other controls on the form. The Formula Provider extends the formula and function capability to other controls on the form. The Formula Text Box can be used to make formula bars or other edit controls in your application. The Name Box can be used to add custom names in the application. These controls extend spreadsheet capability to other parts of your application.

- **Setting up the Formula Text Box**
- **Setting up the Formula Provider**
- **Setting up the Name Box**

For more information on formulas, refer to **Managing Formulas in Cells** and the Formula Reference.

## Setting up the Formula Text Box

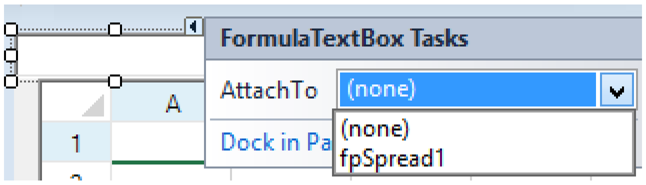
You can set up a floating formula bar that end users can use to add formulas. The formula bar is similar to the formula editor available to the developer and has the appearance of a text box. The formula bar provides a list of calculation functions. It also provides a visual method of selecting cell ranges for the formula.



Refer to the following example to see how to create the formula text box.

### Setting up the Formula Text Box

To set up the formula bar at run time, 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.

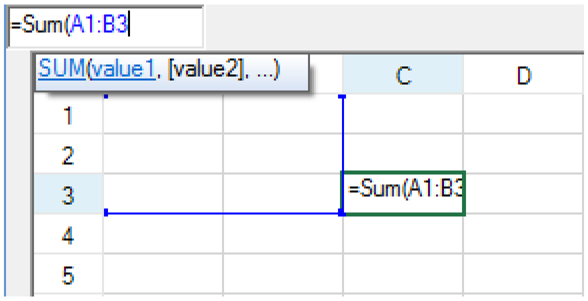


The **AllowUserFormulas ('AllowUserFormulas Property' in the on-line documentation)** property allows the user to type formulas in the cell in the Spread control.

If you set the **AllowUserFormulas** property to True, then the formulas that are typed in a cell 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.



For more information on formulas, refer to **Managing Formulas in Cells** and the Formula Reference.

**Using Code**

Create the formula editor and attach it to the control.

**Example**

This example code creates the floating formula bar.

**C#**

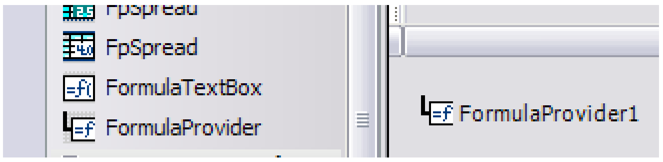
```
FarPoint.Win.Spread.FormulaTextBox editor = new FarPoint.Win.Spread.FormulaTextBox();
editor.Location = new Point(0, 0);
editor.Size = new Size(80, 20);
this.Controls.Add(editor);
editor.Attach(fpSpread1);
// This line will disconnect the formula bar from the control
// editor.Detach();
```

**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()
```

## Setting up the Formula Provider

You can add a formula provider control to the form. The provider control allows you to add values from other controls using formulas and put the result in a control on the form. Custom formula functions can be used (**AddCustomFunctionInfo ('AddCustomFunctionInfo Method' in the on-line documentation)** method) as well as custom names (**AddQueryValueName ('AddQueryValueName Method' in the on-line documentation)** method). 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.



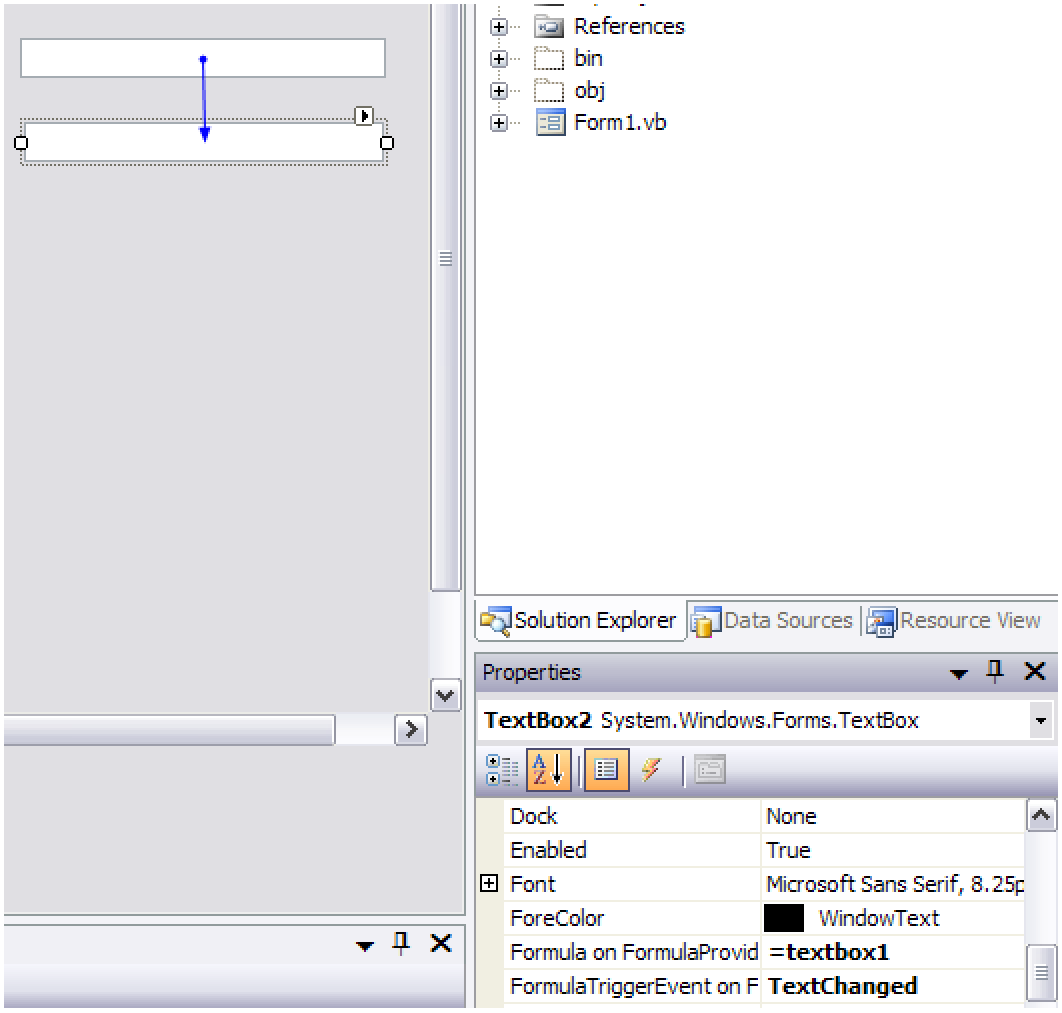
### Setting up the Formula Provider Control

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.



| | |
|---|---|
| Dock | None |
| Enabled | True |
| Font | Microsoft Sans Serif, 8.25p |
| ForeColor | WindowText |
| Formula on FormulaProvid | =textbox1 |
| FormulaTriggerEvent on F | TextChanged |

For more information on formulas, refer to **Managing 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 **SetFormulaTriggerEvent ('SetFormulaTriggerEvent Method' in the on-line documentation)** 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**

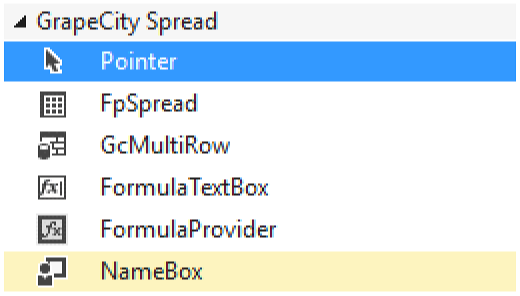
```
FormulaProvider1.SetFormula(TextBox2, "=TextBox1")
FormulaProvider1.SetFormulaTriggerEvent(TextBox1, "TextChanged")
```

## 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**

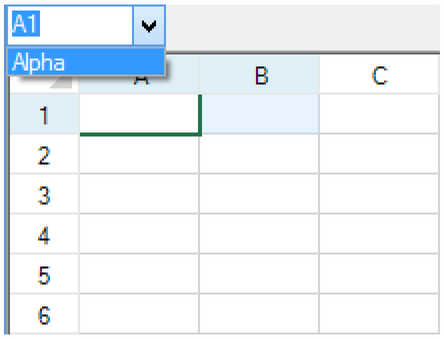
Select the name box control from the **Toolbox** and draw it on the form or use the **NameBox ('NameBox Class' in the on-line documentation)** class to create the control at runtime. Then attach the control to Spread. The following image displays the NameBox control in the toolbox.



**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)
```

## Managing 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**
  - **Customizing Column and Field Binding**
  - **Binding a Combo Box to a DataReader**
- **Adding to Bound Data**
  - **Adding a Row to a Bound Sheet**
  - **Adding an Unbound Row to a Bound Sheet**
  - **Adding an Unbound Column to a Bound Sheet**
- **Customizing Data Binding**
  - **Customizing Column Headers for Bound Sheets**
  - **Customizing Cell Types for Bound Sheets**
  - **Working with Hierarchical Data Display**
  - **Creating a Hierarchical Display Manually**
  - **Creating Custom Hierarchy Icons**

If you would like to step through an example, see one of the brief tutorials:

- **Tutorial: Binding to a Corporate Database (Visual Studio 2013) (on-line documentation)**
- **Tutorial: Binding to a Corporate Database (Older Visual Studio)**

## 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**
- **Customizing Column and Field Binding**
- **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) (on-line documentation)**.

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");
```

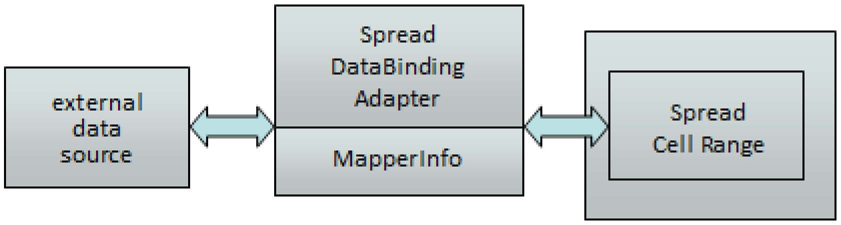
**VB**

```
' 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 | Cell Type |
|---|---|
| Boolean | Check box cell |
| DateTime | Date time cell |
| Double, Single, Decimal | Number cell |
| Int16, Int32, and so on | Number cell |
| String | Text cell |
| Other | General cell |

For more information, refer to the **SpreadDataBindingAdpater ('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();
```

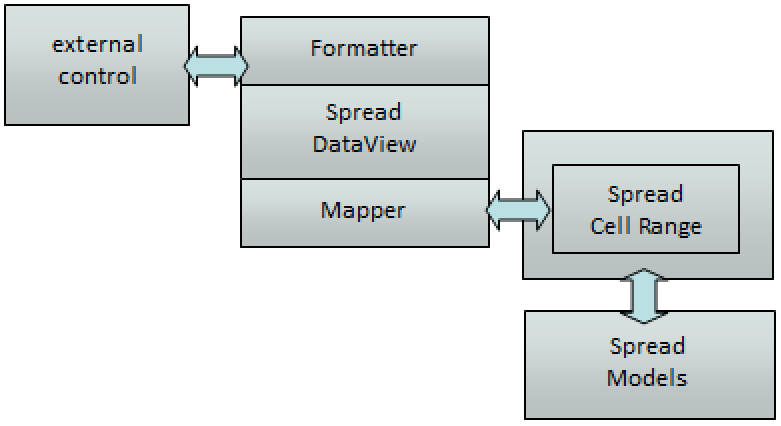
```
// Assign the datasource to a data table
data.DataSource = dt;
data.Spread = fpSpread1;
data.SheetName = "Sheet1";
data.MapperInfo = new FarPoint.Win.Spread.Data.MapperInfo(3, 2, 1, 1);
data.FillSpreadDataByDataSource();
```

**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 ('SpreadDataView 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()
```

## 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.

## 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()
```

## Adding to Bound Data

You can add rows or columns to bound data.

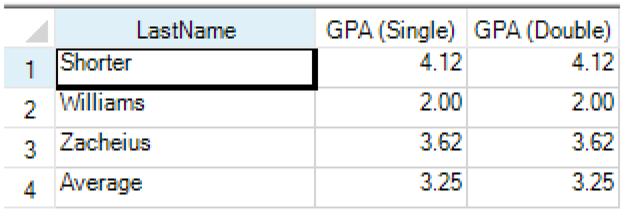
The tasks you can perform to add rows or columns include:

- **Adding a Row to a Bound Sheet**
- **Adding an Unbound Row to a Bound Sheet**
- **Adding an Unbound Column to a Bound 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);
```

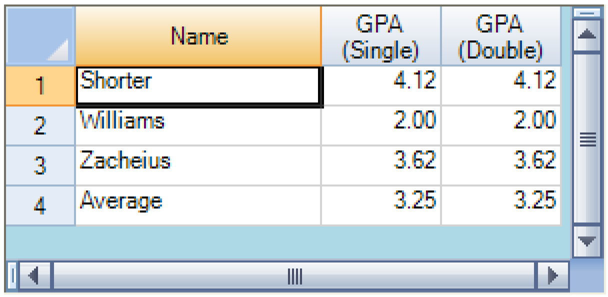
**VB**

```
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
```

## 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 Data Binding

You can customize the display including headers, cell types, and hierarchical display.

The tasks you can perform for customizing the display of bound data include:

- **Customizing Column Headers for Bound Sheets**
- **Customizing Cell Types for Bound Sheets**
- **Working with Hierarchical Data Display**
- **Creating a Hierarchical Display Manually**
- **Creating Custom Hierarchy Icons**

## 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.
4. 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

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 for 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 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.

## 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**

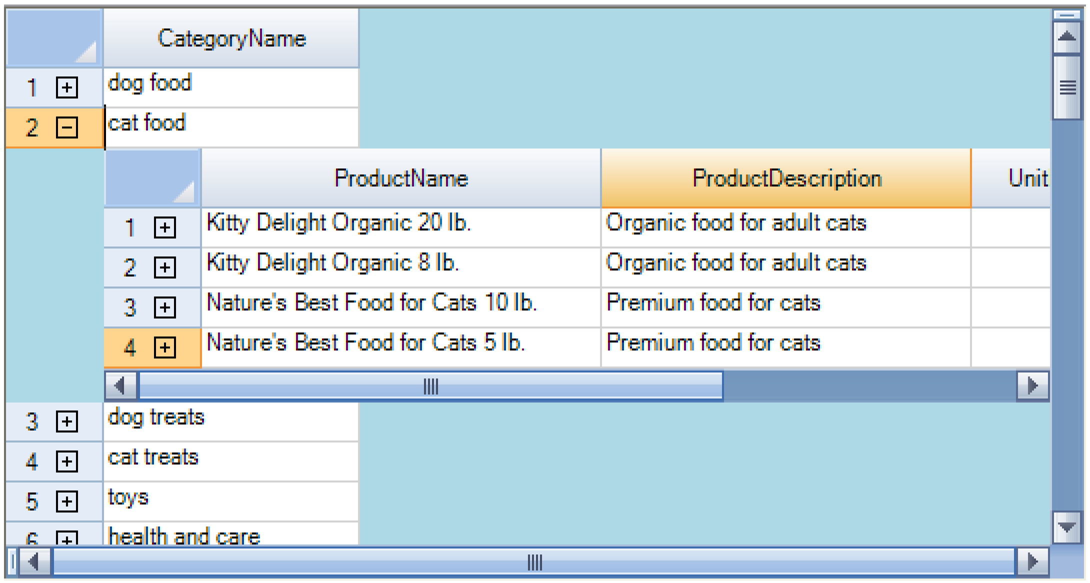
```
' Create a new SheetView object.
Dim newsheet As New FarPoint.Win.Spread.SheetView()
' Bind the SheetView object to the data set.
newsheet.DataSource = DataSet1
' Change the column header text in the first header cell.
newsheet.ColumnHeader.Cells(0, 0).Text = "Student ID"
' Assign the SheetView object 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.

## 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 | |

| | CategoryName |
|---|---|
| 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**.

For a small example see the **Tutorial: Binding to a Corporate Database (Older Visual Studio)**.

### 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\GrapeCity\Spread Studio 9\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 Studio 9\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 classn as String
Private grade As String
Public Property ClassName() As String
Get
Return classn
End Get
Set(ByVal Value As String)
classn = Value
End Set
End Property

Public Property Grades() As String
Get
Return grade
End Get
Set(ByVal Value As String)
grade = Value
End Set
End Property
End Class

Public Class student
Private name As String
Private id As String
Private sco As ArrayList = New ArrayList()
Public Property names() As String
Get
Return name
End Get
Set(ByVal Value As String)
name = Value
End Set
End Property

Public Property ids() As String
Get
Return id
End Get
Set(ByVal Value As String)
id = Value
End Set
End Property
```

```
Public ReadOnly Property score() As ArrayList
Get
Return sco
End Get
End Property
End Class

'Form Load
Dim list As ArrayList = New ArrayList()
Dim s As student = New student()
s.names = "John Smith"
s.ids = "100001"
Dim sc As Score = New Score()
sc.ClassName = "math"
sc.Grades = "A"
s.Score.Add(sc)
sc = New Score()
sc.ClassName = "English"
sc.Grades = "A"
s.Score.Add(sc)
list.Add(s)
s = New student()
s.names = "David Black"
s.ids = "100002"
sc = New Score()
sc.ClassName = "math"
sc.Grades = "B"
s.Score.Add(sc)
sc = New Score()
sc.ClassName = "English"
sc.Grades = "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**

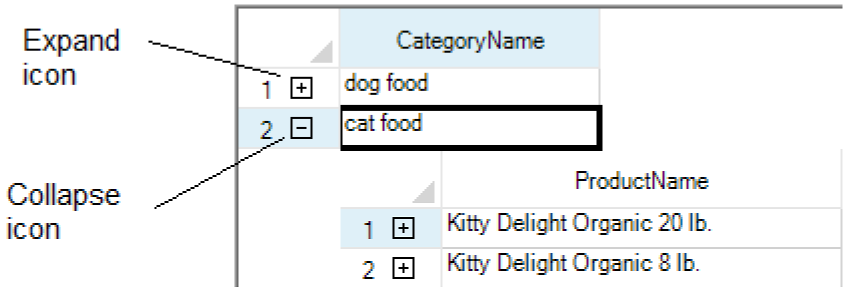
```
...
Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
MyBase.Load
     FpSpread1.Sheets.Clear()
     FpSpread1.Sheets.Add(New customSheet)
     FpSpread1.Sheets(0).RowCount = 10
     Dim i As Integer
     For i = 0 To 9
          If i <> 1 Then
               FpSpread1.Sheets(0).SetRowExpandable(i, 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 (Older Visual Studio)

The following tutorial walks you through creating a project and binding the Spread control to a database.

In this tutorial, the major steps are:

- **Step 1: Adding Spread to a Data Binding Project**
- **Step 2: Setting up the Database Connection**
- **Step 3: Specifying the Data to Use**
- **Step 4: Creating the Data Set**
- **Step 5: Binding Spread to the Database**
- **Step 6: Improving the Display by Changing the Cell Type**

## Step 1: Adding Spread to a Data Binding Project

1. Start a new Visual Studio .NET project.
2. Name the project databind.
3. Name the form file in the project binding.cs (or .vb).
4. 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, complete the steps in **Adding a Component to a Project**

## Step 2: Setting up the Database Connection

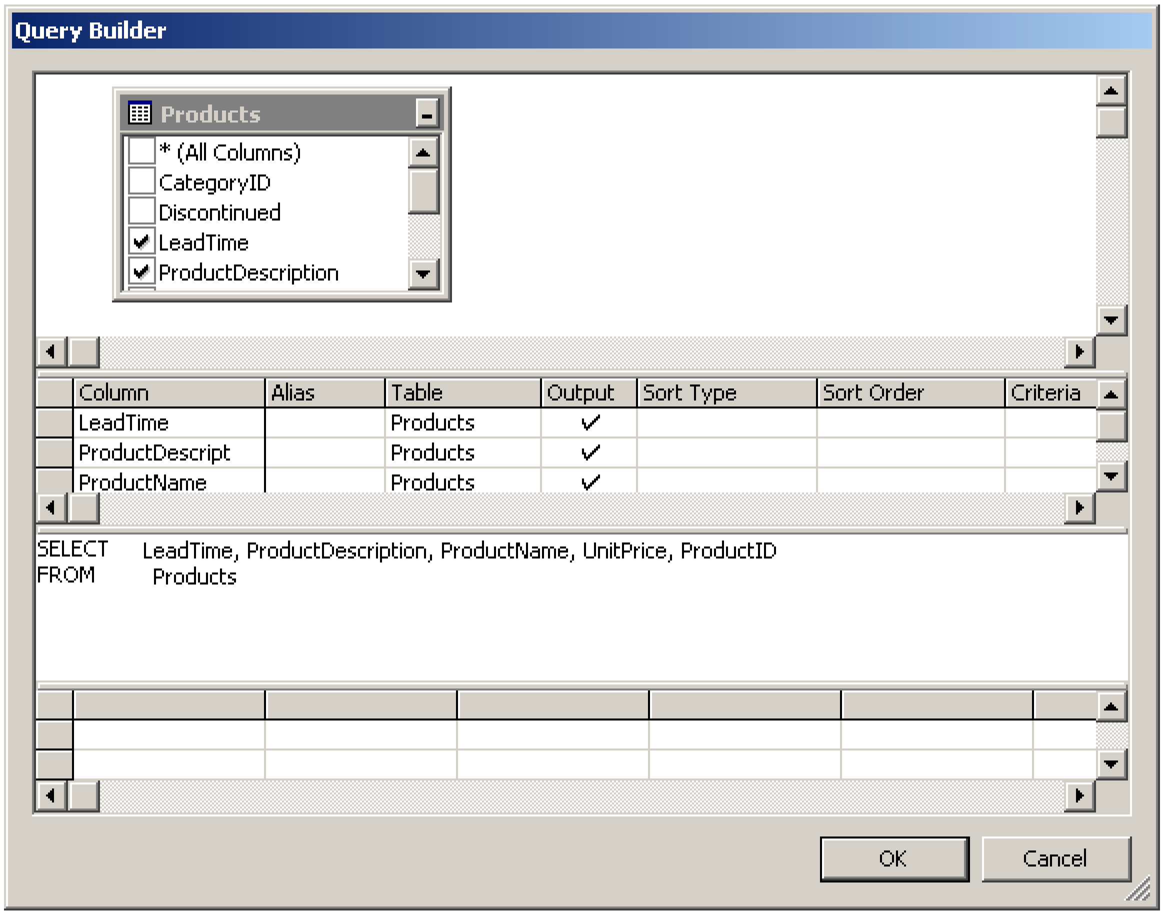
You must tell the project which database you want to use. In this step, you will add a OleDbConnection control to your form, and tell it the name of the database to use.

1. If the Toolbox is not displayed, from the **View** menu choose **Toolbox**.
2. Click the **Data** tab to display the available data controls.
3. Double-click the OleDbConnection control to add it to your form.
   The OleDbConnection control is added to your form, in a new area created below the visible area of the form. The data controls you create in this tutorial will all be placed in this area, instead of in the visible area of the form.
4. Press F4 to display the **Properties** window for the OleDbConnection control.
5. In the **Properties** window, change the name of the control to dbConnect.
6. In the **Properties** window, click the **ConnectionString** property.
7. Click the down arrow displayed on the right side of the setting area, then select **New Connection** from the drop-down list.
   The **Data Link Properties** dialog is displayed.
8. Click the **Provider** tab, and then select **Microsoft Jet 4.0 OLE DB Provider** from the list.
9. Click **Next**.
10. Next to the Select or enter a database name box, click the **Browse** button.
11. Browse to C:\Program Files (x86)\GrapeCity\Spread Studio 9\Docs\Windows Forms\TutorialFiles\databind.mdb and then choose **Open**.
12. Click the **Test Connection** button.
13. If you do not receive a message stating the "Test connection succeeded" retry steps 6 through 12.
14. If you received the message "Test connection succeeded," your connection is complete. Click **OK** to close the **Data Link Properties** dialog.

## Step 3: Specifying the Data to Use

Now that you have specified the database to use, you need to retrieve the records from the database table you want to display in your Spread control. To do this, you will use the OleDbDataAdapter control.

1. If the Toolbox is not displayed, from the **View** menu choose **Toolbox**.
2. Click the **Data** tab to display the available data controls.
3. Double-click the OleDbDataAdapter control to add it to your form.
   The OleDbDataAdapter control is added in the area below the visible area of the form. The **Data Adapter Configuration Wizard** appears.
4. Choose **Next** to begin completing the wizard.
5. In the **Choose Your Data Connection** dialog, under **Which data connection should the data adapter use?** select the connection you created in Step 2 from the drop-down list. Then choose **Next**.
6. In the **Choose a Query Type** dialog, select **Use SQL statements** and then choose **Next**.
7. In the **Generate the SQL statements** dialog, choose **Query Builder**.
   The **Add Table** dialog appears to let you specify the table to use in the database.
8. Select the **Products** table from the list and choose **Add**, then choose **Close**.
9. In the **Query Builder** dialog, the **Product** table appears in a window, with a list of the available fields in the table. Select the following fields:
   - **LeadTime**
   - **ProductDescription**
   - **ProductName**
   - **UnitPrice**
   - **ProductID**
10. The Query Builder creates your SQL query in the status box. Your dialog should look like this:



| Column | Alias | Table | Output | Sort Type | Sort Order | Criteria |
|---|---|---|---|---|---|---|
| LeadTime | | Products | ✓ | | | |
| ProductDescript | | Products | ✓ | | | |
| ProductName | | Products | ✓ | | | |

```
SELECT    LeadTime, ProductDescription, ProductName, UnitPrice, ProductID
FROM       Products
```

11. Choose **OK** to close the **Query Builder** dialog, then choose **Next** in the wizard.
12. The wizard summarizes your choices. Choose **Finish** to complete the wizard.
13. Press F4 to display the **Properties** window for the OleDbDataAdapter control.
14. In the **Properties** window, change the name of the control to dbAdapt.

## Step 4: Creating the Data Set

Now that you have specified the database and the data to use from the database, you will create a data set to contain the data for your Spread control.

**Using Code**

1. Select the dbAdapt OleDBDataAdapter control on the form.
2. Press F4 to display the **Properties** window for the control, if it is not already displayed.
3. Click the **Generate Dataset** verb at the bottom of the **Properties** window.
4. The **Generate Dataset** dialog appears.
5. Click **OK** to close the **Generate Dataset** dialog.
   The new data set control is added to your form.
6. Press F4 to display the **Properties** window for the new DataSet control, if it is not already displayed.
7. In the **Properties** window, change the name of the control to dbDataSet.
8. Double-click on the form in your project to open the code window.
9. Type the following code in the **Form_Load** event:

**Example**

**C#**

```
DataSet ds;
ds = dbDataSet;
dbAdapt.Fill(ds);
```

**VB**

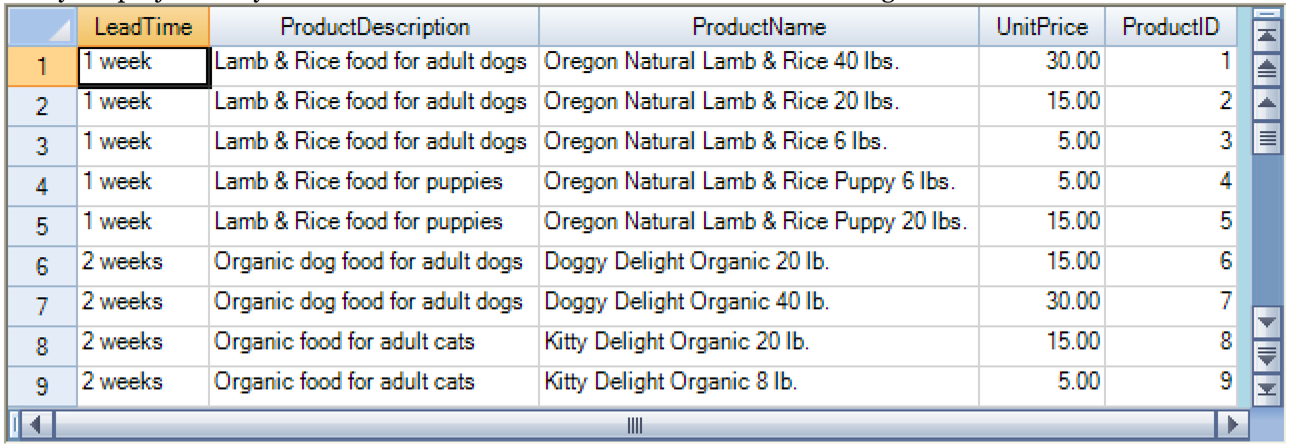
```
Dim ds As DataSet
ds = dbDataSet
dbAdapt.Fill(ds)
```

This fills the data set with the data from the database you specified, using the fields you specified when setting up the OleDbDataAdapter control.

## Step 5: Binding Spread to the Database

Your data set is ready, now you need to provide code to bind the Spread control to the data set.

1. Press F4 to display the **Properties** window for the Spread control, if it is not already displayed.
2. In the **Properties** window, set the **DataSource** property to the name of your data set, dbDataSet. Notice that the column headers in the Spread control change to be the field names from the **Products** table in your database.
3. Save your project.
4. Run your project and you should see a form that looks similar to the following:



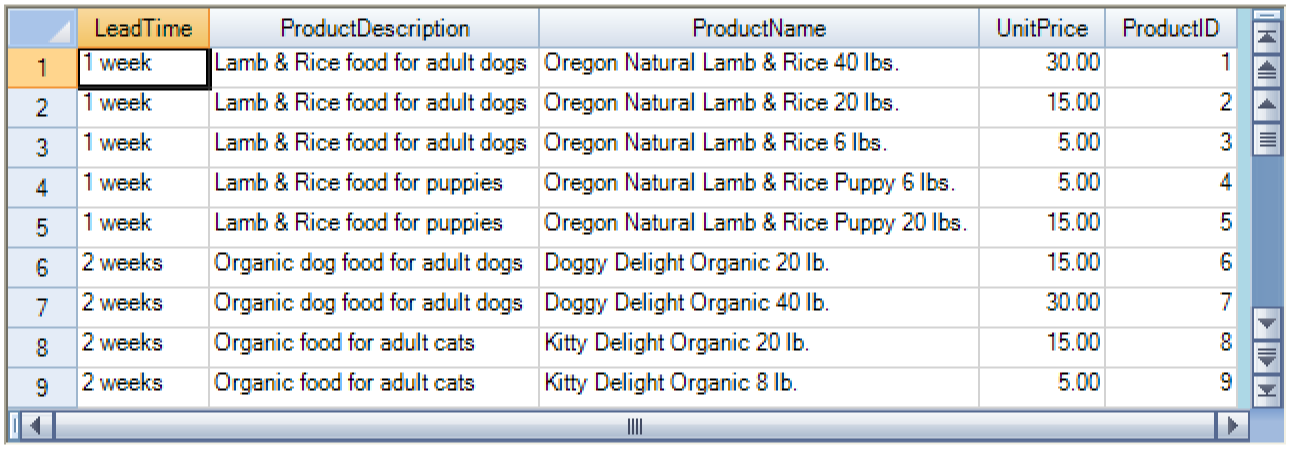
| | LeadTime | ProductDescription | ProductName | UnitPrice | ProductID |
|---|---|---|---|---|---|
| 1 | 1 week | Lamb & Rice food for adult dogs | Oregon Natural Lamb & Rice 40 lbs. | 30.00 | 1 |
| 2 | 1 week | Lamb & Rice food for adult dogs | Oregon Natural Lamb & Rice 20 lbs. | 15.00 | 2 |
| 3 | 1 week | Lamb & Rice food for adult dogs | Oregon Natural Lamb & Rice 6 lbs. | 5.00 | 3 |
| 4 | 1 week | Lamb & Rice food for puppies | Oregon Natural Lamb & Rice Puppy 6 lbs. | 5.00 | 4 |
| 5 | 1 week | Lamb & Rice food for puppies | Oregon Natural Lamb & Rice Puppy 20 lbs. | 15.00 | 5 |
| 6 | 2 weeks | Organic dog food for adult dogs | Doggy Delight Organic 20 lb. | 15.00 | 6 |
| 7 | 2 weeks | Organic dog food for adult dogs | Doggy Delight Organic 40 lb. | 30.00 | 7 |
| 8 | 2 weeks | Organic food for adult cats | Kitty Delight Organic 20 lb. | 15.00 | 8 |
| 9 | 2 weeks | Organic food for adult cats | Kitty Delight Organic 8 lb. | 5.00 | 9 |

5. If your form does not look similar to this form, adjust the size of your Spread control, and re-check the steps you have performed so far.
6. Stop the project.

## Step 6: Improving the Display by Changing the Cell Type

In this step, you will change the cell type for one of the columns to better display the data from the database.

Run your project and you should see a form that looks similar to the following:



| | LeadTime | ProductDescription | ProductName | UnitPrice | ProductID |
|---|---|---|---|---|---|
| 1 | 1 week | Lamb & Rice food for adult dogs | Oregon Natural Lamb & Rice 40 lbs. | 30.00 | 1 |
| 2 | 1 week | Lamb & Rice food for adult dogs | Oregon Natural Lamb & Rice 20 lbs. | 15.00 | 2 |
| 3 | 1 week | Lamb & Rice food for adult dogs | Oregon Natural Lamb & Rice 6 lbs. | 5.00 | 3 |
| 4 | 1 week | Lamb & Rice food for puppies | Oregon Natural Lamb & Rice Puppy 6 lbs. | 5.00 | 4 |
| 5 | 1 week | Lamb & Rice food for puppies | Oregon Natural Lamb & Rice Puppy 20 lbs. | 15.00 | 5 |
| 6 | 2 weeks | Organic dog food for adult dogs | Doggy Delight Organic 20 lb. | 15.00 | 6 |
| 7 | 2 weeks | Organic dog food for adult dogs | Doggy Delight Organic 40 lb. | 30.00 | 7 |
| 8 | 2 weeks | Organic food for adult cats | Kitty Delight Organic 20 lb. | 15.00 | 8 |
| 9 | 2 weeks | Organic food for adult cats | Kitty Delight Organic 8 lb. | 5.00 | 9 |

1. Double-click on the form to open the code window.
2. Set the cell type for the **UnitPrice** column by adding the code in the following example after the code you have already added:
3. Save your project.

**Example**

**C#**

```
FarPoint.Win.Spread.CellType.CurrencyCellType CurrCell = new
FarPoint.Win.Spread.CellType.CurrencyCellType();
CurrCell.DecimalPlaces = 2;
CurrCell.CurrencySymbol = "US$";
fpSpread1.Sheets[0].Columns[3].CellType = CurrCell;
```

**VB**

```
Dim CurrCell As New FarPoint.Win.Spread.CellType.CurrencyCellType()
CurrCell.DecimalPlaces = 2
CurrCell.CurrencySymbol = "US$"
FpSpread1.Sheets(0).Columns(3).CellType = CurrCell
```

You have managed data binding to a corporate database using Spread. You have completed this tutorial.

Review the list of steps for **Tutorial: Binding to a Corporate Database (Older Visual Studio)**.

## 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**
- **Validating User Input**
- **Rearranging Data on a Sheet**
- **Removing Data from a Sheet**
- **Improving Performance by Suspending the Layout**

## 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**
- **Repeatedly Filling a Range of Cells with Copied Cells**

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)**<br>**SetText ('SetText Method' in the on-line documentation)** |
| | Range of cells | **GetClip ('GetClip Method' in the on-line documentation)**<br>**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)**<br>**SetValue ('SetValue Method' in the on-line documentation)** |
| | Range of cells | **GetClipValue ('GetClipValue Method' in the on-** |

| | | |
|---|---|---|
| | | **line documentation)**<br>**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)**<br>**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 | Cell Property |
|---|---|
| As a string with formatting (for example "$1,234.56") | **Text ('Text Property' in the on-line documentation)** |
| As a string without formatting (for example "1234.45") | **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))
```

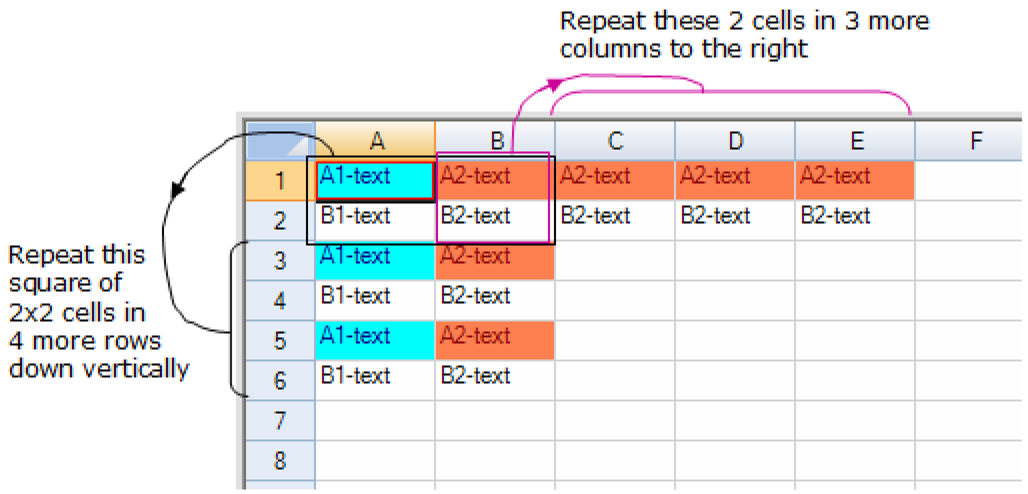
**Note:** Empty cells return a null value (Nothing in VB) that causes the cast to fail. If there is a possibility that a cell is empty or contains a value of unknown data type then your code should check the data type prior to performing the cast or should provide an exception handler to catch the exception thrown by the failed cast.

## 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)


Repeat these 2 cells in 3 more columns to the right
Repeat this square of 2x2 cells in 4 more rows down vertically

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | A1-text | A2-text | A2-text | A2-text | A2-text | |
| 2 | B1-text | B2-text | B2-text | B2-text | B2-text | |
| 3 | A1-text | A2-text | | | | |
| 4 | B1-text | B2-text | | | | |
| 5 | A1-text | A2-text | | | | |
| 6 | B1-text | B2-text | | | | |
| 7 | | | | | | |
| 8 | | | | | | |

**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)
```

## 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. 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<br>$1,234.56<br>£45 |
| DateTime | Date and time data, which can include separator characters | 8/16/2002,<br>Monday, August 05, 2002 4:40 PM |
| GcDateTime | Date and time data, which can include separator characters | 8/16/2002,<br>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 decimal symbol | 3.45<br>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<br>1,234.56 |
| Percent | Numeric data, which can include characters for the percent symbol, a separator, and a decimal symbol | 0.5<br>1,234% |
| Text | Any character is accepted. | |

For more information about differences between these cell types, refer to the **Customizing Interaction with 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)** |
| 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 **Managing 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.

## Rearranging Data on a Sheet

You can rearrange data on a sheet from cell to cell in many ways.

- **Copying Data on a Sheet**
- **Moving Data on a Sheet**
- **Swapping Data on a Sheet**

## 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**.

## 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
```

## 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 ('SwapColumns Method' in the on-line documentation)** Method
- Model Namespace: **DefaultSheetSpanMode**l Class: **SwapRows ('SwapRows Method' in the on-line documentation)** 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 only data
- **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)**

## Improving Performance by Suspending the Layout

One way to improve the performance of the component, if there are changes to many cells, is to hold or suspend 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 component can save a lot of time and still deliver a refreshed interface to the user.

**Layout Objects**

A layout is an object that stores calculated values (mostly widths and heights of cells, spans, and viewports) used for painting the component 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 component by storing the calculated layout values used during painting and reusing them each time the component repaints instead of recalculating them each time. When a change is tracked that requires the layout object to be regenerated, it is 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.NoteStyle = 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 ('SuspendLayout Method' in the on-line documentation)** 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."

**Example**

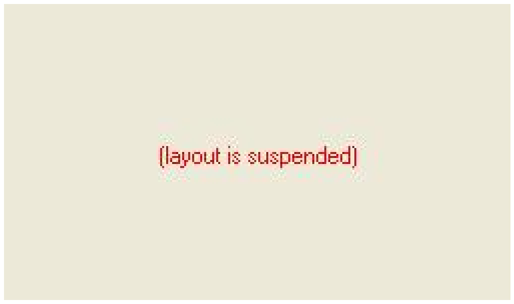
This example suspends and resumes the layout.

**C#**

```
Dim st As System.DateTime = System.DateTime.Now
With FpSpread1
.SuspendLayout()
.Sheets(0).ActiveSkin = FarPoint.Win.Spread.SheetSkin.Load("d:\temp\skin.skn")
.ResumeLayout()
End With
MsgBox("Duration (ticks)=" & System.DateTime.Now.Ticks - st.Ticks)

private void PerformInitialSetup()
    {
      // (0) used for property labels.
      _propertyLabelOrdinal = 0;
      // Set up the spread
      spread.SuspendLayout();

      SheetView sheet = spread.ActiveSheet;
      sheet.Models.ColumnHeaderData = new HeaderDataModel(spread, _orientation);
      sheet.Models.Style = new SheetStyleModel(_orientation);
      spread.NamedStyles = _config.Styles.NamedStyles;

      // Insert initial data
      sheet.Columns.Count = 2;
      foreach (Block block in _config.Blocks)
      {
        // Insert any leading blank rows
        if (block.SpaceBefore > 0) sheet.Rows.Add(sheet.Rows.Count, block.SpaceBefore);
        if (block.DealProperties != null)
        {
          int rowIndex = sheet.Rows.Count;
          sheet.Rows.Add(rowIndex, block.DealProperties.Count);
          foreach (DealProperty property in block.DealProperties)          {
            sheet.Cells[rowIndex, _propertyLabelOrdinal].Value = property;
            sheet.Rows[rowIndex].StyleName = block.StyleName;
            rowIndex++;
           }
        }

        // Insert any trailing blank rows
        if (block.SpaceAfter > 0) sheet.Rows.Add(sheet.Rows.Count, block.SpaceAfter);
     }

     // Set initial styles
    sheet.Columns[_propertyLabelOrdinal].StyleName = "dealPropertyLabels";

    spread.ResumeLayout();
   }
```

**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.


(layout is suspended)

This should only happen when the layout is suspended with **SuspendLayout ('SuspendLayout Method' in the on-line documentation)**, 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 setting text only does it under certain circumstances, for example, when you have **AllowCellOverflow** turned on. 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.

**Other Performance Improvements**

Beyond the suspending (and subsequent resuming) of the layout logic, there are few other things you can consider that may improve performance.

If you are not using sticky notes, then you can set **AutoUpdateNotes** to false to prevent the component from checking for sticky notes that need to be made visible or hidden or moved. If you are using **AllowCellOverflow**, turning that off increases the performance of the layout calculations, because that feature requires multiple text width calculations on each change to the data in a cell. If you are using formulas, setting **AutoCalculation** to false before your updates and then setting it back to true and calling **Recalculate** afterwards eliminates redundant intermediate recalculations of your formulas.

Some other things to consider are to reduce the size of the control or display fewer columns and rows at once (the layout objects only calculate the visible portion of the sheet), or implement your own sheet model objects (like your own data model object implementing **ISheetDataModel**) which remove features that you do not require (for example, if you do not require data binding, the data binding interfaces do not need to be implemented).

**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.

**Example**

This example shows the two methods used together within a specific scope around code that changes the cells. This suspends the repainting of the Spread component while changing the color of the cells and then resumes the repainting.

**C#**

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

**VB**

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

## Managing 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 **WhenAncestorOfFocused** input map for processing keystrokes that occur while the component or one of its child controls has focus. The **WhenAncestorOfFocused** 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 **WhenAncestorOfFocus** 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 **WhenAncestorOfFocused** 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 **WhenAncestorOfFocused**, 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 + Page Up | Extends selection left one page of columns | **ExtendToPreviousPageOfColumns ('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 ('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 | Moves to previous item in the list | **MoveToPreviousItem ('MoveToPreviousItem** |

| | | |
|---|---|---|
| Arrow | | **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 Field' in the on-line documentation)** |
| Page Down | Selects the next page of items in the list | **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 on in a date cell, spreadsheet displays a pop-up calendar to let you choose a date | **ShowSubEditor ('ShowSubEditor Field' 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)** |
| 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 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)** |

## 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 RowMode and WhenFocused

The default map for this operation mode and input map mode is summarized in this table.

| Key Code | Action Name |
|---|---|
| F2 | **ClearCell ('ClearCell Field' in the on-line documentation)** |
| Alt+Up Arrow | **ComboShowList ('ComboShowList Field' in the on-line documentation)** |
| Alt+Down Arrow | **ComboShowList ('ComboShowList Field' in the on-line documentation)** |
| F3 | **DateTimeNow ('DateTimeNow Field' in the on-line documentation)** |
| F4 | **ShowSubEditor ('ShowSubEditor 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 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 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 | **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 | Action Name |
|---|---|

| Key | Code |
|---|---|
| Right | **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)** |
| 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(newKeystroke(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)
```

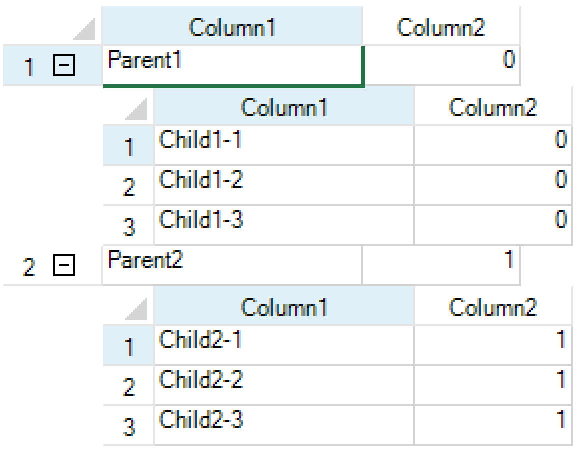
```
' Create another InputMap object.
Dim inputmap2 As New FarPoint.Win.Spread.InputMap()
' Assign this InputMap object to the existing map.
inputmap2 = FpSpread1.GetInputMap(FarPoint.Win.Spread.InputMapMode.WhenFocused)
' Map the Enter key.
inputmap2.Put(New FarPoint.Win.Spread.Keystroke(Keys.Enter, 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
```

## 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.

## 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.

## Managing 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 | • MouseDown<br>• Enter<br>• GotFocus<br>• **CellClick ('CellClick Event' in the on-line documentation)**<br>• **LeaveCell ('LeaveCell Event' in the on-line documentation)**<br>• **EnterCell ('EnterCell Event' in the on-line documentation)**<br>• Paint<br>• MouseUp<br>• MouseCaptureChanged<br>• **SelectionChanged ('SelectionChanged Event' in the on-line documentation)**<br>• Paint |
| Click on a combo box cell and select an item | • **ComboDropDown ('ComboDropDown Event' in the on-line documentation)**<br>• **ComboSelChange ('ComboSelChange Event' in the on-line documentation)**<br>• **EditChange ('EditChange Event' in the on-line documentation)**<br>• **ComboCloseUp ('ComboCloseUp Event' in the on-line documentation)**<br>• Paint |
| Click on a multiple option cell | • MouseDown |

| | |
|---|---|
| and select an option | • **CellClick ('CellClick Event' in the on-line documentation)**<br>• **LeaveCell ('LeaveCell Event' in the on-line documentation)**<br>• **EnterCell ('EnterCell Event' in the on-line documentation)**<br>• **EditModeStarting ('EditModeStarting Event' in the on-line documentation)**<br>• MouseCaptureChanged<br>• ControlAdded<br>• **EditModeOn ('EditModeOn Event' in the on-line documentation)**<br>• LostFocus<br>• Paint<br>• **ButtonClicked ('ButtonClicked Event' in the on-line documentation)** |
| Double-click a general cell (go into edit mode) | • MouseDown<br>• **CellClick ('CellClick Event' in the on-line documentation)**<br>• **LeaveCell ('LeaveCell Event' in the on-line documentation)**<br>• **EnterCell ('EnterCell Event' in the on-line documentation)**<br>• Paint<br>• MouseUp<br>• MouseCaptureChanged<br>• **SelectionChanged ('SelectionChanged Event' in the on-line documentation)**<br>• Paint<br>• MouseDown<br>• **CellDoubleClick ('CellDoubleClick Event' in the on-line documentation)**<br>• **EditModeStarting ('EditModeStarting Event' in the on-line documentation)**<br>• MouseCaptureChanged<br>• Layout<br>• ControlAdded<br>• **EditModeOn ('EditModeOn Event' in the on-line documentation)**<br>• LostFocus<br>• Paint |
| Expanding a hierarchy (the first time) | • MouseDown<br>• **CellClick ('CellClick Event' in the on-line documentation)**<br>• **Expand ('Expand Event' in the on-line documentation)**<br>• Layout<br>• ControlAdded<br>• Layout<br>• **DataColumnConfigure ('DataColumnConfigure Event' in the on-line documentation)**<br>• ... **DataColumnConfigure ('DataColumnConfigure Event' in the on-line documentation)** (repeat for each column in expanded row)<br>• Layout<br>• ControlAdded |

| | |
|---|---|
| | • Layout<br>• ... Layout (repeat for each column in expanded row)<br>• ControlAdded<br>• Layout<br>• ... Layout (repeat for each expanded row)<br>• **ChildWorkbookCreated ('ChildWorkbookCreated Event' in the on-line documentation)**<br>• **ChildViewCreated ('ChildViewCreated Event' in the on-line documentation)**<br>• Layout<br>• ControlRemoved<br>• Layout<br>• ControlRemoved<br>• Layout<br>• ControlAdded<br>• Layout<br>• ... Layout (repeat for each expanded column)<br>• ControlAdded<br>• Layout<br>• Paint<br>• MouseUp<br>• MouseCaptureChanged<br>• Paint |
| Expanding a hierarchy (subsequent times) | • MouseDown<br>• **CellClick ('CellClick Event' in the on-line documentation)**<br>• **Expand ('Expand Event' in the on-line documentation)**<br>• Layout<br>• ControlAdded<br>• Layout<br>• ... Layout (repeated)<br>• ControlAdded<br>• Layout<br>• Paint<br>• MouseUp<br>• MouseCaptureChanged<br>• Paint |
| Collapsing a hierarchy (the first time or subsequent times) | • MouseDown<br>• **CellClick ('CellClick Event' in the on-line documentation)**<br>• **Expand ('Expand Event' in the on-line documentation)**<br>• Layout<br>• ControlRemoved<br>• Layout<br>• ControlRemoved<br>• Paint<br>• MouseUp<br>• MouseUp |

- Paint

## Selecting Actions

Actions such as selecting cells and working with selections in Spread result in these events:

| User Action | List of Events |
|---|---|
| Select a cell - click a general (default) cell | • MouseDown<br>• Enter<br>• GotFocus<br>• **CellClick ('CellClick Event' in the on-line documentation)**<br>• MouseUp<br>• MouseCaptureChanged<br>• **SelectionChanged ('SelectionChanged Event' in the on-line documentation)**<br>• Paint |
| Select range of cells - click a general (default) cell and drag to another cell | • MouseDown<br>• **CellClick ('CellClick Event' in the on-line documentation)**<br>• **LeaveCell ('LeaveCell Event' in the on-line documentation)**<br>• **EnterCell ('EnterCell Event' in the on-line documentation)**<br>• Paint<br>• **SelectionChanging ('SelectionChanging Event' in the on-line documentation)**<br>• Paint<br>• ... (repeating Paint every time you drag over to a cell in another row or column)<br>• MouseUp<br>• MouseCaptureChanged<br>• **SelectionChanged ('SelectionChanged Event' in the on-line documentation)**<br>• Paint |
| Select a row (or column) - click on header cell | • MouseDown<br>• Enter<br>• GotFocus<br>• **CellClick ('CellClick Event' in the on-line documentation)**<br>• **LeaveCell ('LeaveCell Event' in the on-line documentation)**<br>• **EnterCell ('EnterCell Event' in the on-line documentation)**<br>• MouseUp<br>• MouseCaptureChanged<br>• **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)** |

| | |
|---|---|
| | • Paint<br>• MouseUp<br>• MouseCaptureChanged<br>• Paint |
| Click on the tab of the active sheet | • Paint |
| Remove a sheet | • 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 | • MouseDown<br>• Enter<br>• GotFocus<br>• MouseUp<br>• MouseCaptureChanged<br>• **ColumnWidthChanged ('ColumnWidthChanged Event' in the on-line documentation)**<br>• 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 | • MouseDown<br>• **ShapeActivated ('ShapeActivated Event' in the on-line documentation)**<br>• Paint<br>• ... Paint (repeated for each cell moved through)<br>• MouseUp<br>• MouseCaptureChanged |
| Rotate Shape | • MouseDown<br>• Paint<br>• ... Paint (repeated for each cell moved through)<br>• MouseUp<br>• MouseCaptureChanged |

## Print Actions

This table lists the events for the various printing actions in Spread:

| Action | List of Events |
|---|---|
| Print Preview | • **PrintMessageBox ('PrintMessageBox Event' in the on-line documentation)**<br>• **PrintPreviewShowing ('PrintPreviewShowing Event' in the on-line documentation)**<br>• LostFocus<br>• **PrintBackground ('PrintBackground Event' in the on-line documentation)**<br>• **PrintAbort ('PrintAbort Event' in the on-line documentation)**<br>• **PrintMessageBox ('PrintMessageBox Event' in the on-line documentation)** |
| Print a sheet | • **PrintMessageBox ('PrintMessageBox Event' in the on-line documentation)**<br>• **PrintMessageBox ('PrintMessageBox Event' in the on-line documentation)**<br>• **PrintBackground ('PrintBackground Event' in the on-line documentation)**<br>• **PrintAbort ('PrintAbort Event' in the on-line documentation)**<br>• **PrintMessageBox ('PrintMessageBox Event' in the on-line documentation)**<br>• GotFocus<br>• Paint |

## Managing 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**

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-formatted file (BIFF8 format) or the Excel 2007 XML format (xlsx) using the **UseOOXMLFormat** option of the **ExcelSaveFlags ('ExcelSaveFlags Enumeration' in the on-line documentation)** enumeration. 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.

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.

For more details on the methods used, refer to **SaveExcel ('SaveExcel Method' in the on-line documentation)** methods of the **FpSpread ('FpSpread Class' in the on-line documentation)** class. There are many different **SaveExcel ('SaveExcel Method' in the on-line documentation)** methods. Some of the methods have a **saveFlags** option. This allows you to specify headers and other options. Headers are exported as frozen columns and rows.

The **Document** caching option in the **ExcelOpenFlags ('ExcelOpenFlags Enumeration' in the on-line documentation)** or **ExcelSaveFlags ('ExcelSaveFlags Enumeration' in the on-line documentation)** 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.

For more information about how the data is saved to an Excel-formatted file, see the **Import and Export Reference (on-line documentation)**.

**Using Code**

Use one of the **SaveExcel ('SaveExcel Method' in the on-line documentation)** methods of the **FpSpread ('FpSpread Class' in the on-line documentation)** class, providing the path and file name for the file to save, and any additional parameters depending on the particular method.

**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)
```

**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, 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 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 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) (http://www.w3.org) HTML 4.01 reference site (http://www.w3.org/TR/html4).

**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 cellspacing="0" cellpadding="0"
rules="all" 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;">0</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;">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:#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 Spread COM File**.
- **Opening a Custom Text 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

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 **ActiveShee**t 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 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.

## 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 the 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.

## 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 get 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 **IsSerializable** 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.

## 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)
```

## Managing Printing

You can print a spreadsheet, or parts of a spreadsheet, and use a variety of options to customize printing. Spread allows 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, 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 an Entire Sheet

You can print a sheet in the component with the **FpSpread**.**PrintSheet ('PrintSheet Method' in the on-line documentation)** method. Use this method to print the sheet or sheets, using the **PrintInfo ('PrintInfo Property' in the on-line documentation)** settings for each sheet. You can only print one sheet in the component at a time; each sheet can have its own **PrintInfo** object, but you can call the **PrintSheet** method once to do one or all of the sheets. If the sheet parameter is set to -1, all the sheets in the Spread component 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 when printing a sheet.

The default setting prints in black and white and automatically determines the best order in which to print pages. With the default settings, the following items print 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 **PrintSheet** with different sheet parameters one after the other but calling **PrintSheet** 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.

You can also use the Spread Designer to set properties for printing, and you can print directly from the Spread Designer. For more information, refer to **Printing a Sheet from Spread Designer (on-line documentation)**.

**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.

**C#**

```
fpSpread1.PrintSheet(1);
```

**VB**

```
FpSpread1.PrintSheet(1)
```

## 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 **Finding 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.
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);
```

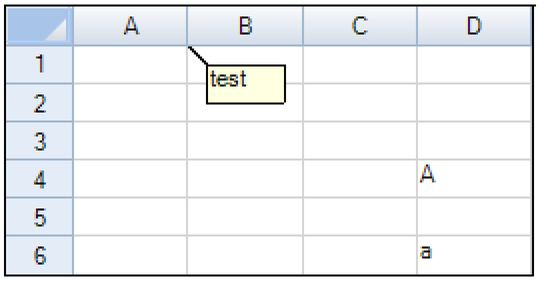
**VB**

```
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.



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
```

## 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**
- **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 | Description |
|---|---|
| **AbortMessage ('AbortMessage Property' in the on-line documentation)** | Gets or sets the message to display for the abort dialog. See **Displaying an Abort Message for the** |

| | |
|---|---|
| | **User**. |
| **BestFitCols ('BestFitCols Property' in the on-line documentation)** | Gets or sets whether column widths are adjusted to fit the longest string width for printing. See **Optimizing the Printing Using Size**. |
| **BestFitRows ('BestFitRows Property' in the on-line documentation)** | Gets or sets whether row heights are adjusted to fit the tallest string height for printing. See **Optimizing the Printing Using Size**. |
| **Centering ('Centering Property' in the on-line documentation)** | Gets or sets whether to center the print out. See **Customizing the Printed Page Layout**. |
| **Colors ('Colors Property' in the on-line documentation)** | 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**. |
| **ColStart ('ColStart Property' in the on-line documentation)** and **ColEnd ('ColEnd Property' in the on-line documentation)** | Used for printing a portion of the sheet. See **Printing a Range of Cells on a Sheet**. |
| **FirstPageNumber ('FirstPageNumber Property' in the on-line documentation)** | Gets or sets the page number to print on the first page. See **Customizing the Printed Page Layout**. |
| **Footer ('Footer Property' in the on-line documentation)** | Used for providing footers on printed pages. See **Customizing the Printed Page Header or Footer**. |
| **Header ('Header Property' in the on-line documentation)** | Used for providing headers on printed pages. See **Customizing the Printed Page Header or Footer**. |
| **Images ('Images Property' in the on-line documentation)** | 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**. |
| **JobName ('JobName Property' in the on-line documentation)** | Gets or sets the name of the print job. See **Customizing the Print Job Settings**. |
| **Margin ('Margin Property' in the on-line documentation)** | Gets or sets the margins for printing. See **Customizing the Printed Page Layout**. |
| **Opacity ('Opacity Property' in the on-line documentation)** | 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**. |
| **Orientation ('Orientation Property' in the on-line documentation)** | Gets or sets the page orientation used for printing. See **Customizing the Print Job Settings**. |
| **PageStart ('PageStart Property' in the on-line documentation)** and **PageEnd ('PageEnd Property' in the on-line documentation)** | Used for printing a page range. See **Printing Particular Pages**. |
| **PageOrder ('PageOrder Property' in the on-line documentation)** | Gets or sets the order in which pages print. See **Customizing the Print Job Settings**. |
| **PaperSize ('PaperSize Property' in the on-line documentation)** | Gets or sets the paper size to use. See **Customizing the Print Job Settings**. |
| **PaperSource ('PaperSource Property' in the on-line documentation)** | Gets or sets the paper source to use. See **Customizing the Print Job Settings**. |
| **Preview ('Preview Property' in the on-line documentation)** | Used to provide print preview. See **Providing a Preview of the Printing**. |

| | |
|---|---|
| **Printer ('Printer Property' in the on-line documentation)** | Gets or sets the name of the printer to use for printing. See **Customizing the Print Job Settings**. |
| **PrintNotes ('PrintNotes Property' in the on-line documentation)** | Gets or sets whether to print the cell notes. See **Printing a Sheet with Cell Notes**. |
| **PrintShapes ('PrintShapes Property' in the on-line documentation)** | Gets or sets whether to print floating objects. See **Printing a Sheet with Shapes**. |
| **PrintType ('PrintType Property' in the on-line documentation)** | Gets or sets what is to be printed. See **Printing Particular Pages**. |
| **RepeatColStart ('RepeatColStart Property' in the on-line documentation)** and **RepeatColEnd ('RepeatColEnd Property' in the on-line documentation)** | Gets or sets whether to print the same set of columns on each page. See **Customizing the Printed Page Header or Footer**. |
| **RepeatRowStart ('RepeatRowStart Property' in the on-line documentation)** and **RepeatRowEnd ('RepeatRowEnd Property' in the on-line documentation)** | Gets or sets whether to print the same set of rows on each page. See **Customizing the Printed Page Header or Footer**. |
| **RowStart ('RowStart Property' in the on-line documentation)** and **RowEnd ('RowEnd Property' in the on-line documentation)** | Used for printing a portion of the sheet. See **Printing a Range of Cells on a Sheet**. |
| **ShowBorder ('ShowBorder Property' in the on-line documentation)** | Gets or sets whether to print a border around the sheet. See **Customizing the Printed Page Layout**. |
| **ShowColor ('ShowColor Property' in the on-line documentation)** | Gets or sets whether to print the colors as they appear on the screen. See **Customizing the Printed Page Layout**. |
| **ShowColumnHeader ('ShowColumnHeader Property' in the on-line documentation)** and **ShowRowHeader ('ShowRowHeader Property' in the on-line documentation)** | Gets or sets whether to print the column headers and row headers. See **Customizing the Printed Page Layout**. |
| **ShowGrid ('ShowGrid Property' in the on-line documentation)** | Gets or sets whether to print the sheet grid lines. See **Customizing the Printed Page Layout**. |
| **ShowPrintDialog ('ShowPrintDialog Property' in the on-line documentation)** | 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-** | Used for turning on the rules for optimizing the |

| | |
|---|---|
| **line documentation)** | 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**. |

### 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, and page size. 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.

**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
```

## 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.

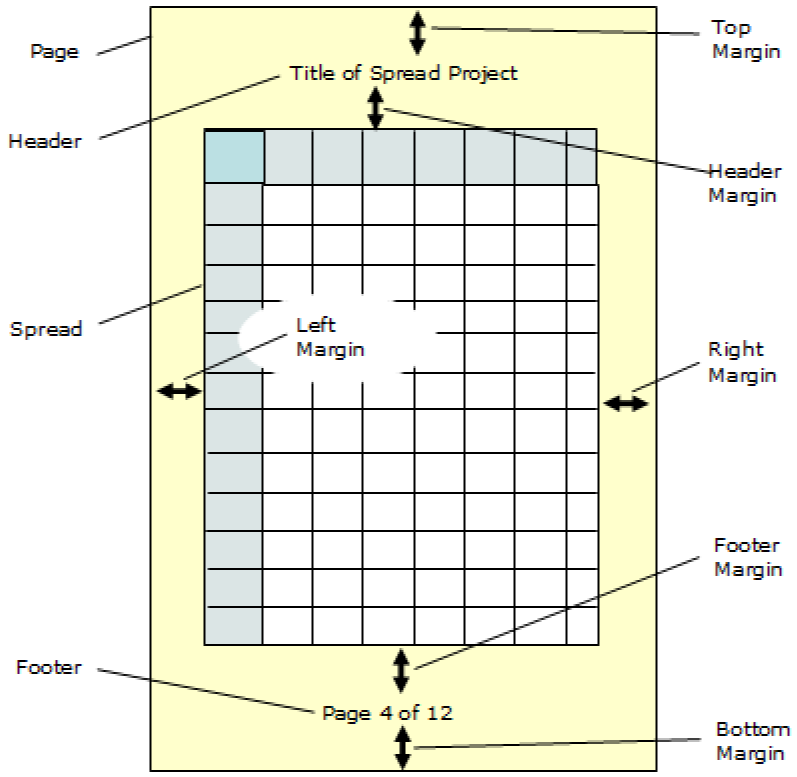
You can set the **PrintInfo Orientation ('Orientation Property' in the on-line documentation)** property.

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 print 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 | /fb0 | Turns off bold font type |
|  | /fb1 | Turns on bold font type |
| /fi | /fi0 | Turns off italics font type |
|  | /fi1 | Turns on italics font type |
| /fk | /fk0 | 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).

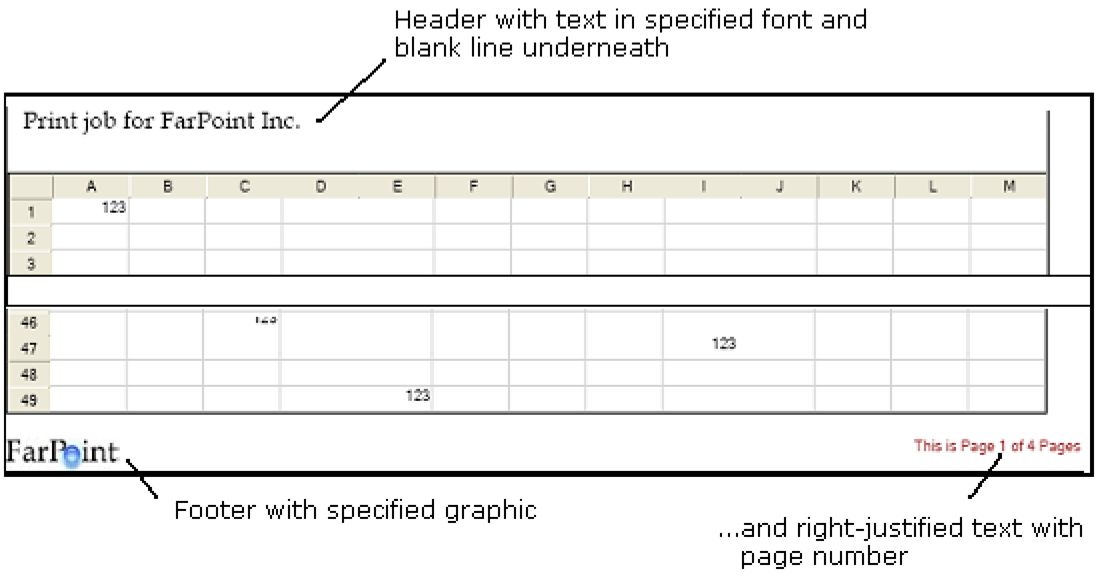
You can specify a color for the text from a list of colors if the color is previously defined in the **Colors** property.

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.


Header with text in specified font and blank line underneath
Print job for FarPoint Inc.
FarPoint
This is Page 1 of 4 Pages
Footer with specified graphic
...and right-justified text with page number

**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.

## 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.

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.

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)**.

## Working with the Chart Control

The following topics explain the basics of the Chart control and how to use the control:

- **Understanding Charts**
- **Creating Charts**

## Understanding Charts

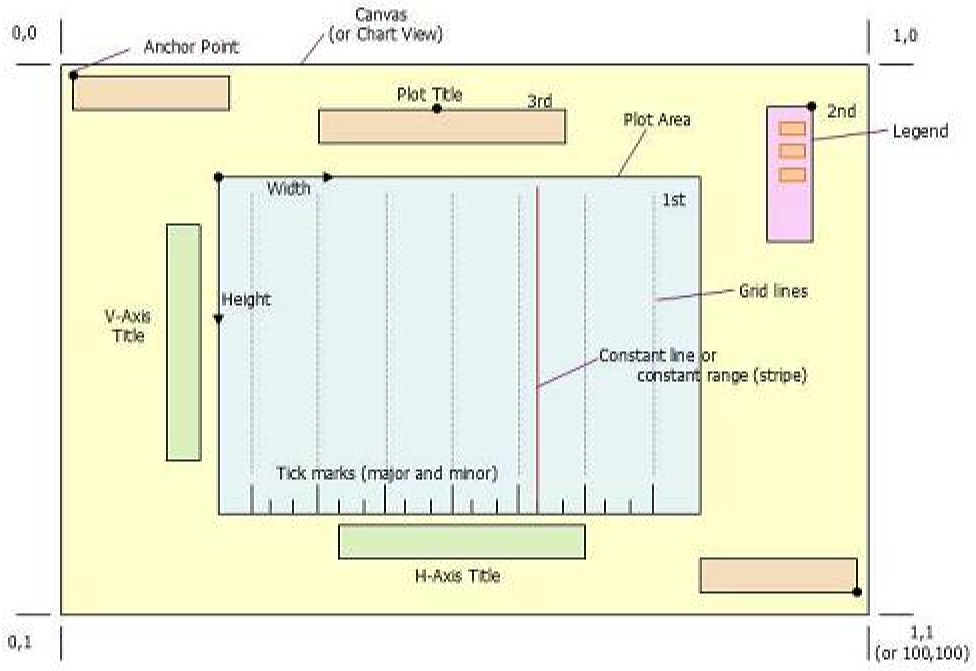
The following topics explain the various chart elements and formatting options.

- **Chart User Interface Elements**
- **Chart Object Model**
- **Chart Types and Views**
- **Plot Types**
- **Plots and Series**
- **Walls**
- **Axis and Other Lines**
- **Fill Effects**
- **Chart Line Style**
- **Elevation and Rotation**
- **Lighting, Shapes, and Borders**
- **Size - Height, Width, and Depth**
- **Labels**
- **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:

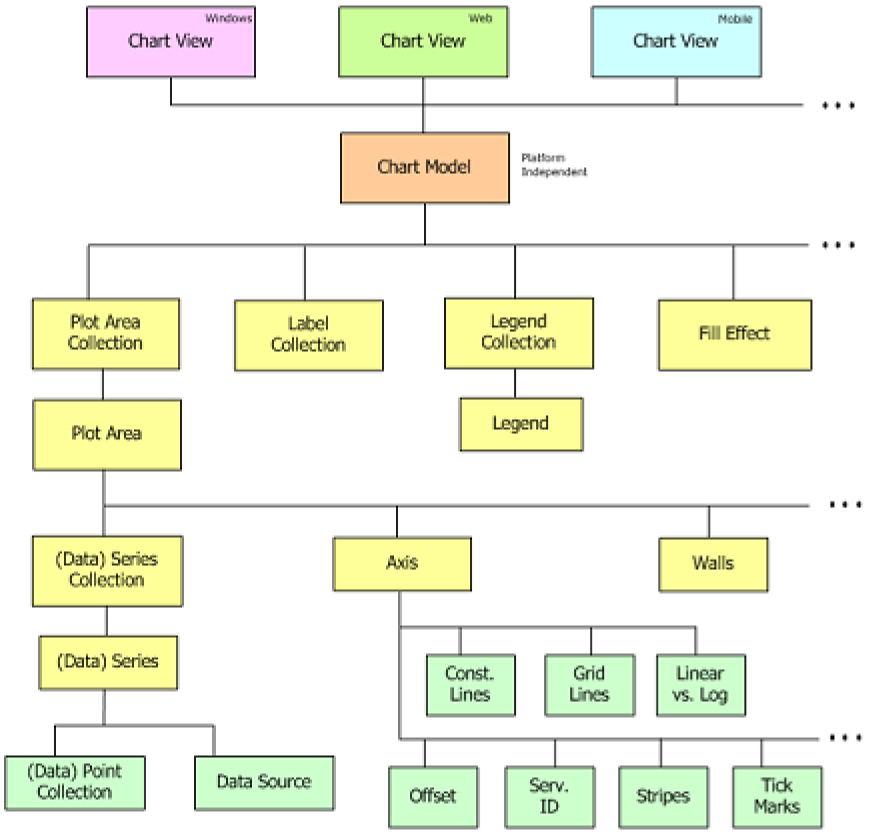
- **LabelArea ('LabelArea Class' in the on-line documentation)** - Labels contain the plot title and the axis labels.
- **LegendArea ('LegendArea Class' in the on-line documentation)** - Legends contain identifiers for each of the series of data.
- **PlotArea ('PlotArea Class' in the on-line documentation)** - 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.

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.

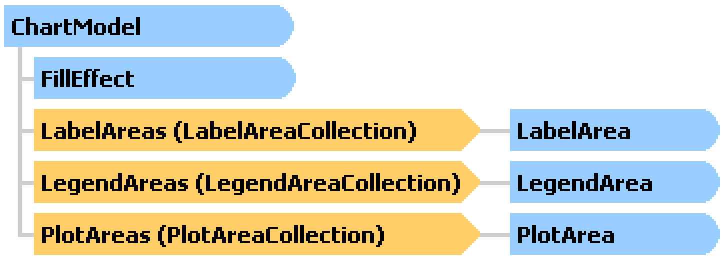
This diagram shows the object model:



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)**
- **LegendAreaCollection ('LegendAreaCollection Class' in the on-line documentation)**
- **PlotAreaCollection ('PlotAreaCollection Class' in the on-line documentation)**
- **Fill ('Fill Class' in the on-line documentation)**


ChartModel
FillEffect
LabelAreas (LabelAreaCollection)
LabelArea
LegendAreas (LegendAreaCollection)
LegendArea
PlotAreas (PlotAreaCollection)
PlotArea

## 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:

- Area
- Bar

- Box Whisker
- Bubble
- Column
- Doughnut
- Funnel
- Histogram
- Line
- Pareto
- Pie
- Polar
- Radar
- Stock
- Sunburst
- Treemap
- Waterfall
- XY
- XYZ

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.

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.

The pie type has the following types of views - 2D Pie, 3D Pie, 2D Exploded Pie, and 3D Exploded Pie.

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.

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.

The XY type has the following types of views - XY Point, XY Line, and XY Line with Marker.

The bubble type has the following types of views - 2D Bubble and 3D Bubble.

The stock type has the following types of views - High Low Close, Open High Low Close, and Candle Stick.

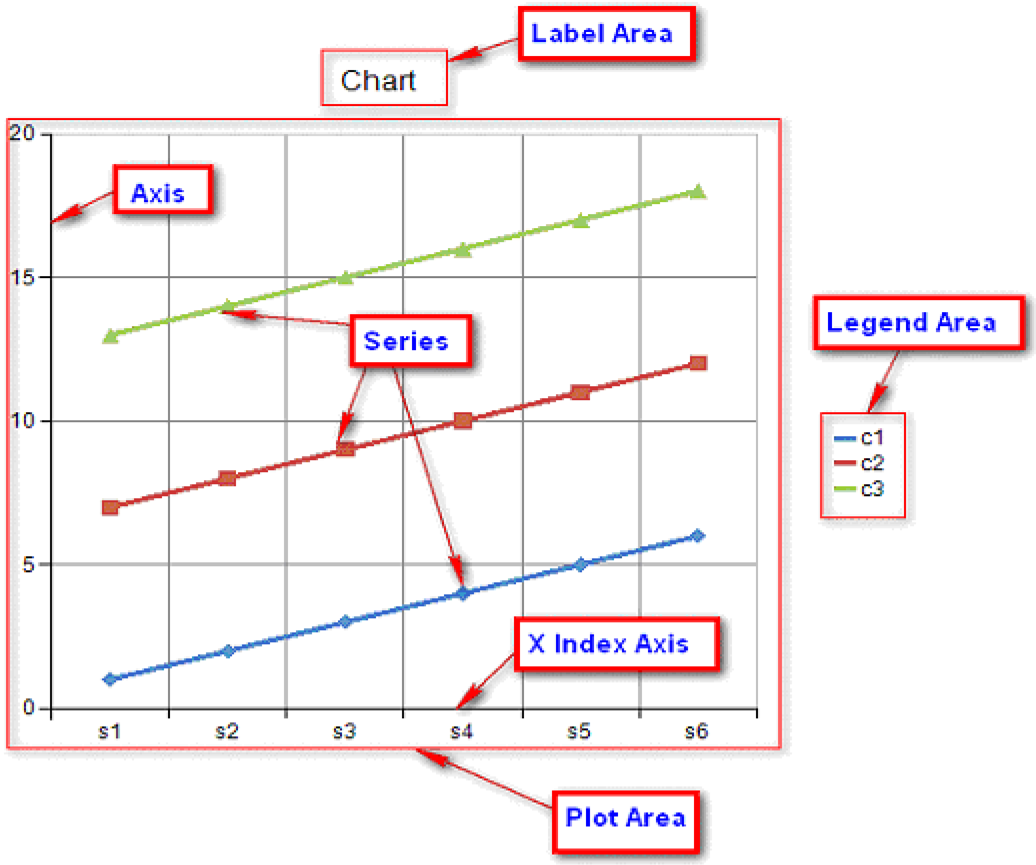
The XYZ type has the following types of views - XYZ Line, XYZ Line with Marker, XYZ Point, and Surface.

The doughnut type has the following types of views - Doughnut and Exploded Doughnut.

The radar type has the following types of views - Radar Line, Radar Line with Marker, Radar Point, and Radar Area.

The polar type has the following types of views - Polar Line, Polar Line with Marker, Polar Point, and Polar Area.

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**
- **Data Plot Types**

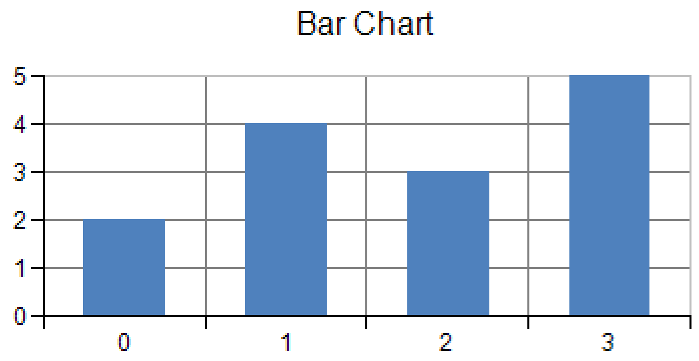
## Y Plot Types

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.

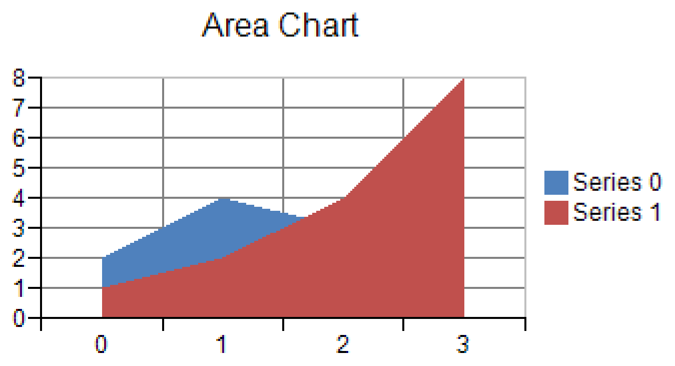


You can have any of these types of Y plots.

- **Area Charts**
- **Bar Charts**
- **Box Whisker Charts (on-line documentation)**
- **Funnel Charts (on-line documentation)**
- **Histogram Charts (on-line documentation)**
- **Line Charts**
- **Market Data (High-Low) Charts**
- **Pareto Charts (on-line documentation)**
- **Point Charts**
- **Stripe Charts**
- **Waterfall Charts (on-line documentation)**

## Area Charts

The area chart can be a basic one-dimensional Cartesian plot such as the one shown in this figure.



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

**Area Charts**

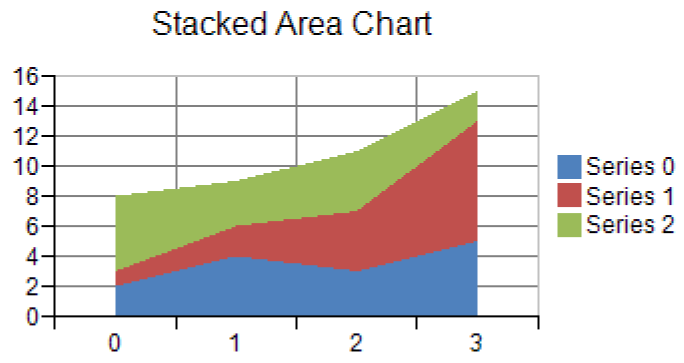
Each point in an area series contains a single data value. The data value is visualized as a point on an area.

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.

**Stacked Area Charts**

The stacked area chart shows the points vertically stacked:



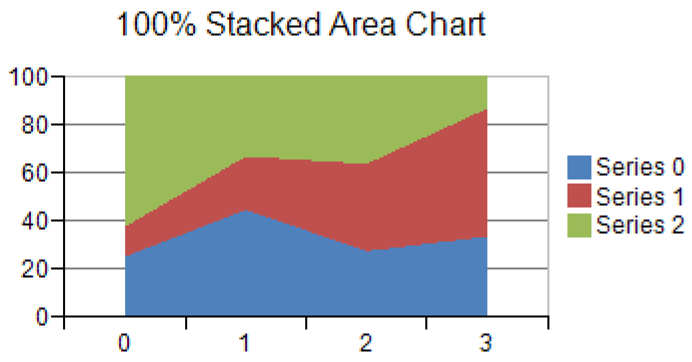
A stacked area series is a composite series that groups together two or more area series.

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.

**Stacked 100% Area Charts**

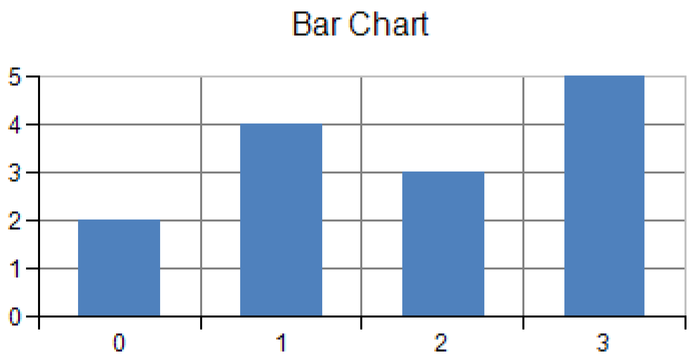
The stacked 100% area chart shows the points vertically stacked and spread all the way to the top (100%) but spread proportionately; otherwise, it is similar to a stacked area chart:



For more information on the area series object in the API, refer to the **AreaSeries ('AreaSeries 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.



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

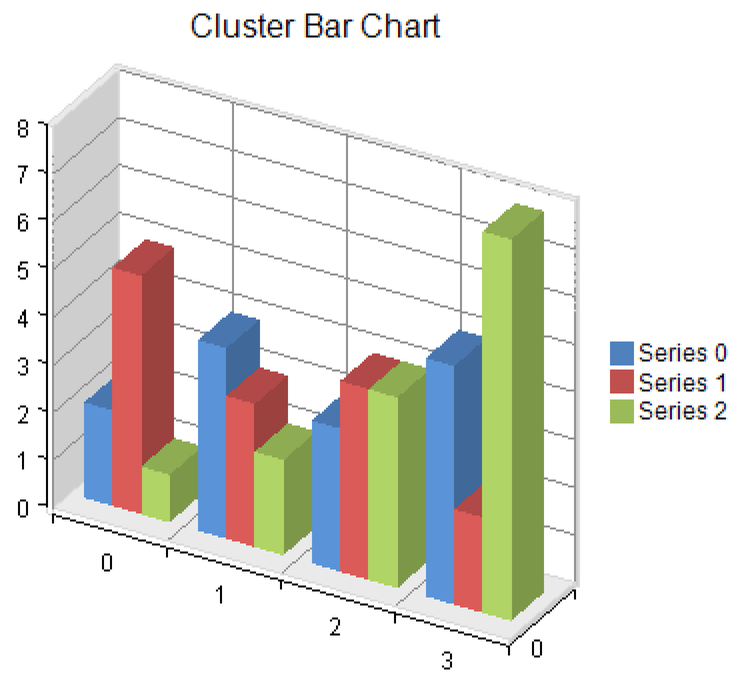
**Bar Charts**

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.

You can also display any of the bar charts as column charts by setting the **Vertical** property in the **YPlotArea ('YPlotArea Class' in the on-line documentation)** class to False.

**Cluster Bar Charts**

The cluster bar shows the bars alongside each other horizontally, clustered by series:



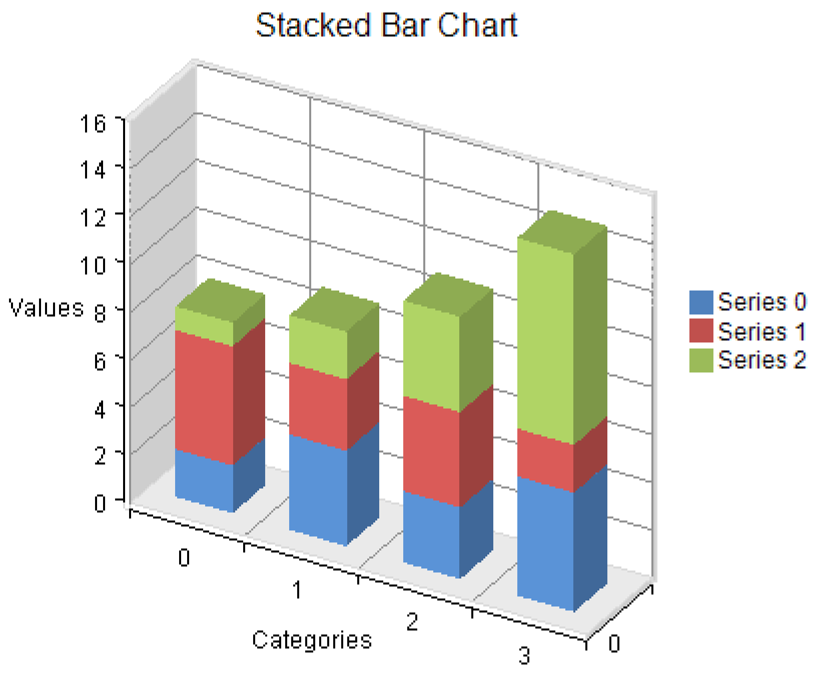
A cluster bar series is a composite series that groups together two or more bar series. A cluster bar series can be assigned a border, fill effect, width, depth, and an origin for the bars as well as a width for the group. Assigning null for a border or fill effect indicates that the property is unset. 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.

**Stacked Bar Charts**

The stacked bar shows the bars vertically stacked:



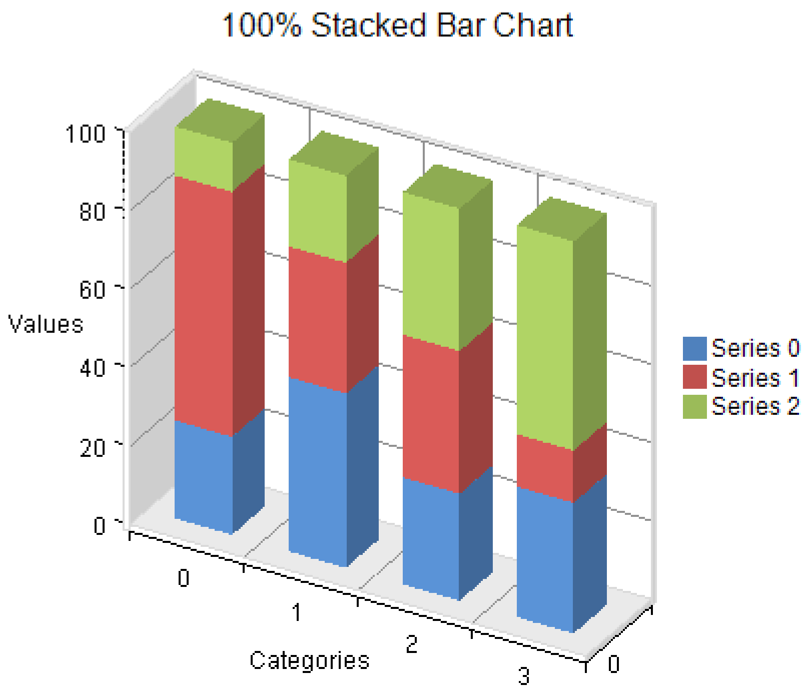
A stacked bar series is a composite series that groups together two or more bar series. A stacked bar series can be assigned a border, fill effect, width, and a depth for the bars. You can also specify whether the group should be displayed 100% stacked. Assigning null for a border or fill effect indicates 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.

**Stacked 100% Bar Charts**

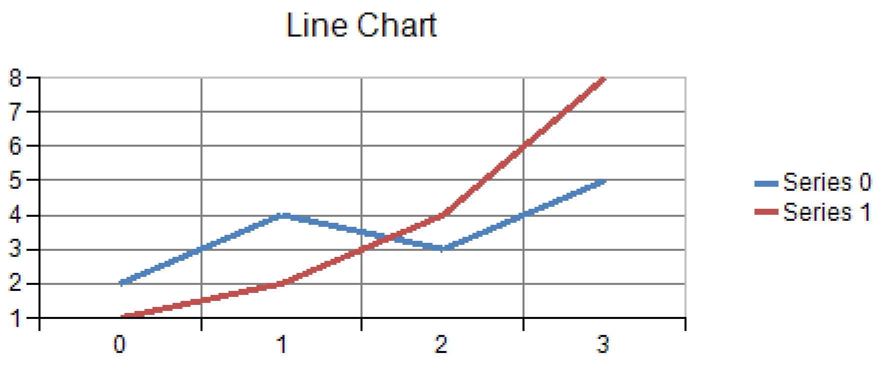
The stacked 100% bar chart shows the bars vertically stacked and spread all the way to the top (100%) but spread proportionately; otherwise, it is similar to a stacked bar chart:



For more information on the bar series object in the API, refer to the **BarSeries ('BarSeries Class' in the on-line documentation)** class.

## Line Charts

The line chart can be a basic one-dimensional Cartesian plot such as the one shown in this figure.



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

**Line Charts**

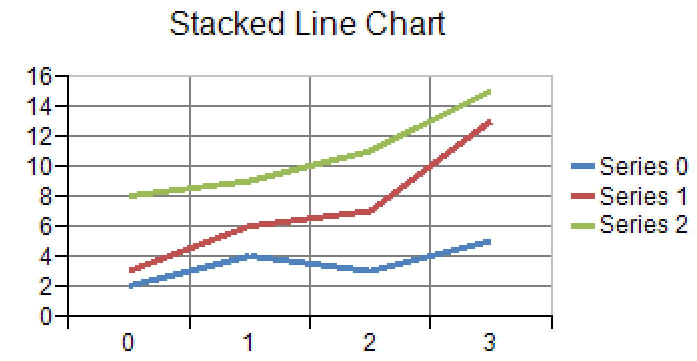
Each point in a line series contains a single data value. The data values are visualized as points on a line.

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.

**Stacked Line Charts**

The stacked line chart shows the points vertically stacked:



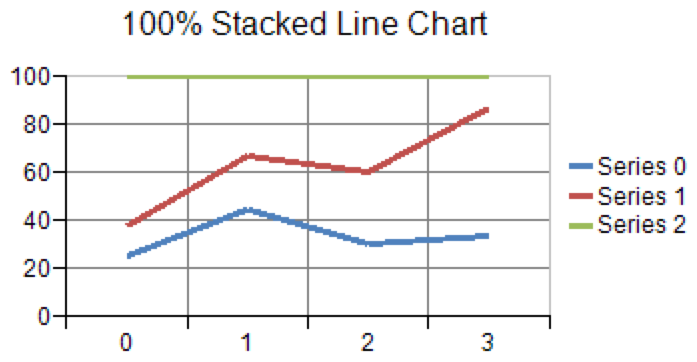
A stacked line series is a composite series that groups together two or more line series. A stacked line series can have a border, fill effect, or depth for the lines. You can also specify whether the line is jagged or smooth and whether drop lines are displayed. Assigning null for a border or fill effect 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.

**Stacked 100% Line Charts**

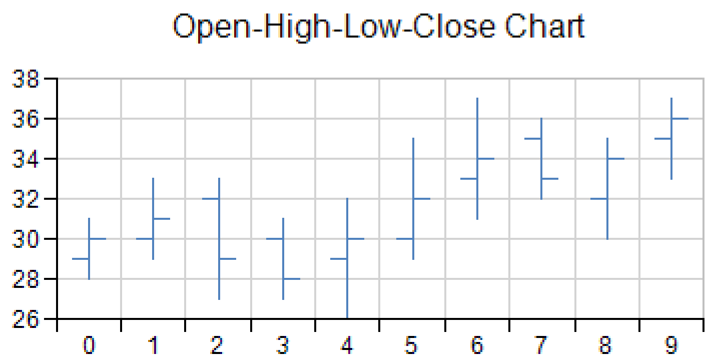
The stacked 100% line chart shows the points vertically stacked and spread all the way to the top (100%) but spread proportionately; otherwise, it is similar to a stacked line chart:



For more information on the line series object in the API, refer to the **LineSeries ('LineSeries Class' in the on-line documentation)** class.

## 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:

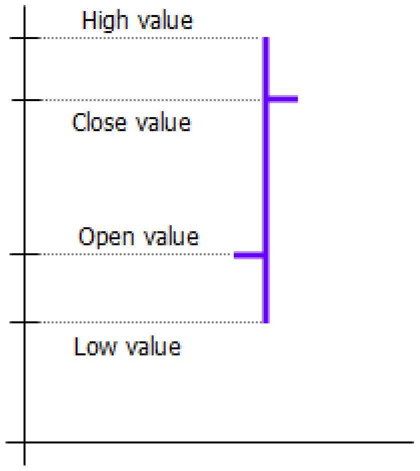


The normal high-low series are displayed as vertical lines with the high value being the vertically highest point on the line and the low value being the lowest point on the line. The opening market value is the small horizontal tick on the left of the line and the closing value is the small horizontal 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.

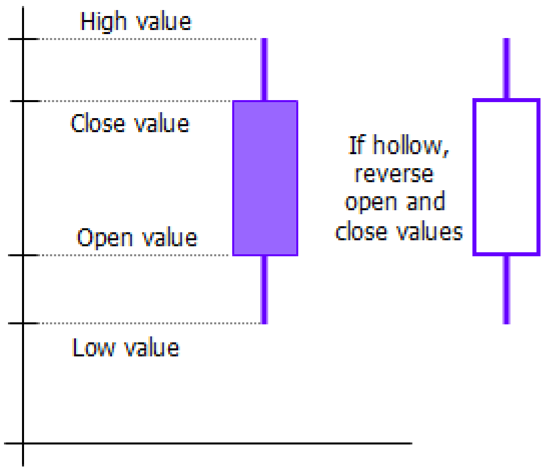
An open-high-low-close series can be assigned a line for up or down points, and a width. The width is measured relative to the floor grid cell (0 = no depth, 1 = width to edge of grid cell).

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.



**Candlestick Charts**

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.

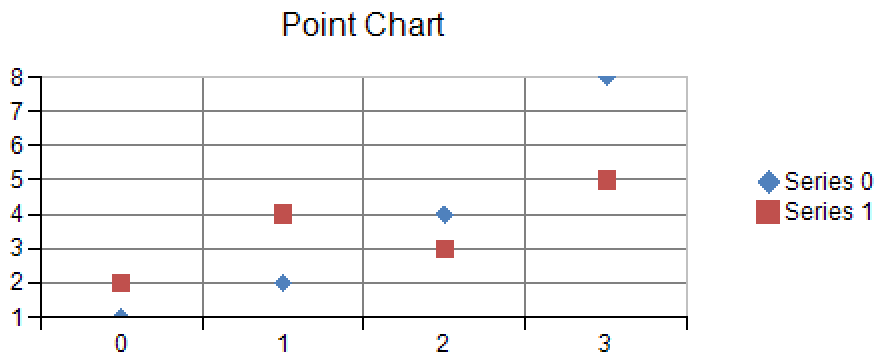
Each point can be assigned a border for up and down points. You can also set a fill effect for up and down points.

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:



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

**Point Charts**

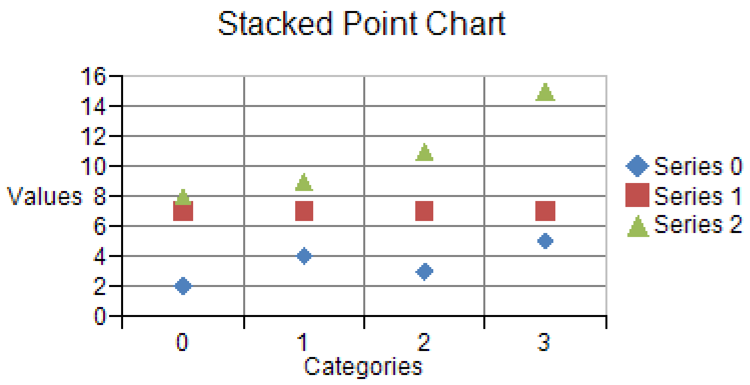
A point marker is used to visualize each data value. Each point in a point series contains a single data value.

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.

**Stacked Point Charts**

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.

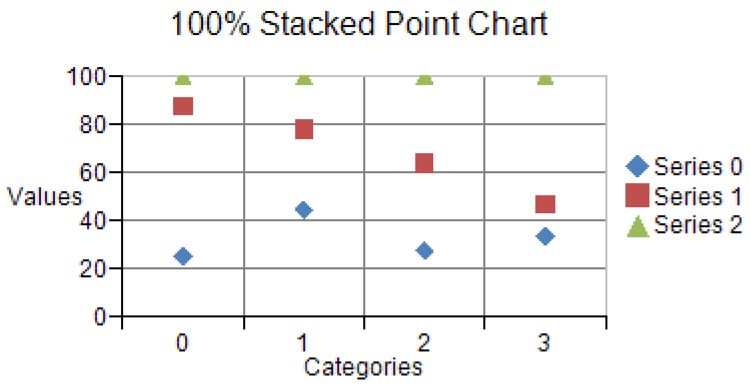
A stacked point series can have a border, fill effect, size, or depth for the point markers. You can also specify whether the group should be displayed as 100% stacked. Assigning null for a border or fill effect indicates that the property is unset. Size is measured in model units. Depth is measured 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.

**Stacked 100% Point Charts**

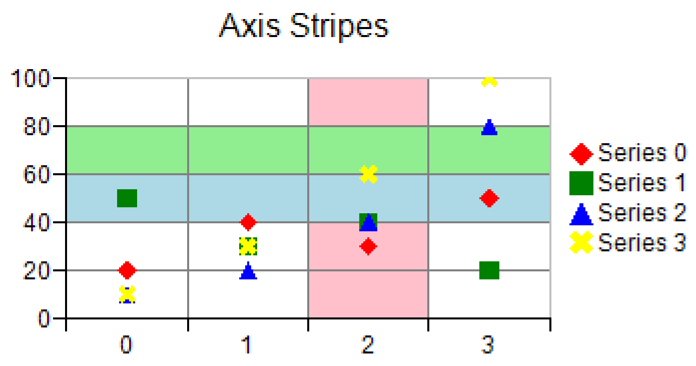
The stacked 100% point chart shows the points vertically stacked and spread all the way to the top (100%) but spread proportionately; otherwise, it is similar to a stacked point chart:



For more information on the point series object in the API, refer to the **PointSeries ('PointSeries Class' in the on-line documentation)** class.

## 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.

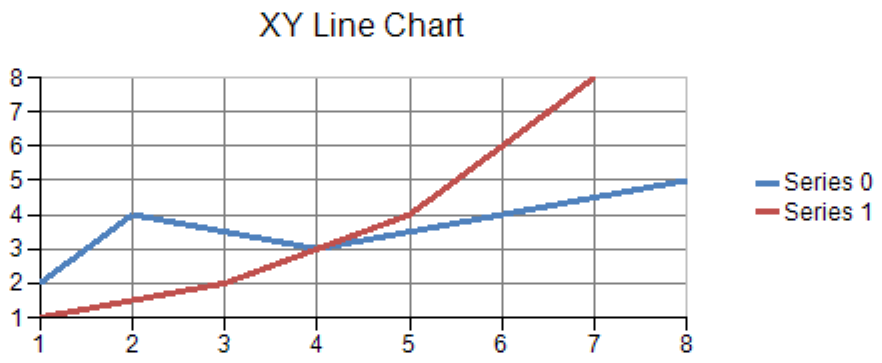
For more information on the stripe object in the API, refer to the **Stripe ('Stripe Class' in the on-line documentation)** class.

For more information on the value axis series object in the API, refer to the ValueAxis class.

## 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.

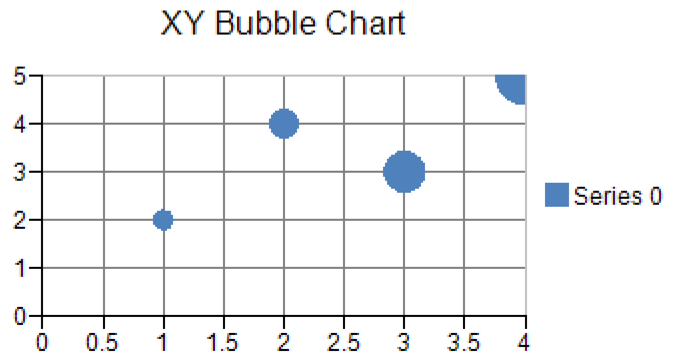


You can have any of these types of XY plots.

- **XY Bubble Charts**
- **XY Line Charts**
- **XY Point Charts**
- **XY Stripe Charts**

## XY Bubble Charts

The bubble chart can be an XY plot such as the one shown in this figure.



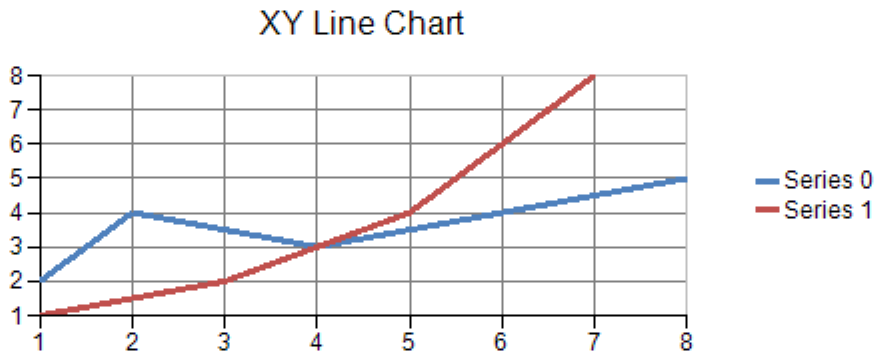
Each point contains two values: value and size.

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 Line Charts

The line chart can be an XY plot such as the one shown in this figure.

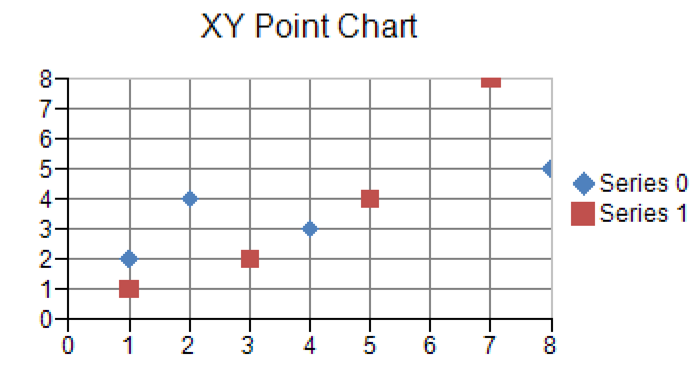


An XZ line series can have a border, fill effect, and depth for the line. You can also specify whether the line is jagged or smooth and whether drop lines are displayed. Assigning null for a border or fill effect indicates that the property is unset. Depth is measured relative to the floor grid cell (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 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.

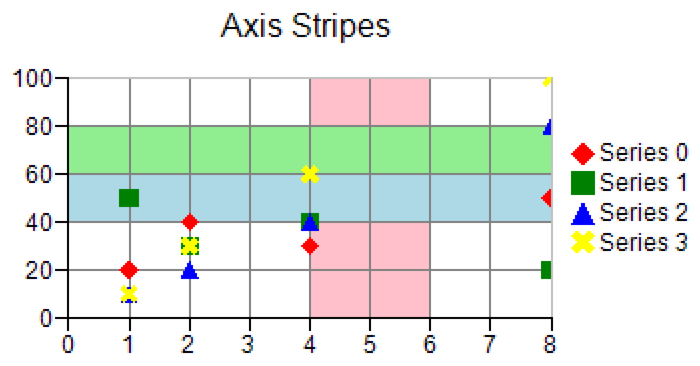
An XY point series can have a border, fill effect, shape, size, and depth for the point markers. Assigning null for a border or fill effect indicates that the property is unset. Size is measured in model units. Depth is measured relative to the floor grid cell (0 = no width, 1 = width of floor grid cell).

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 **PointSeries ('PointSeries 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.



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.

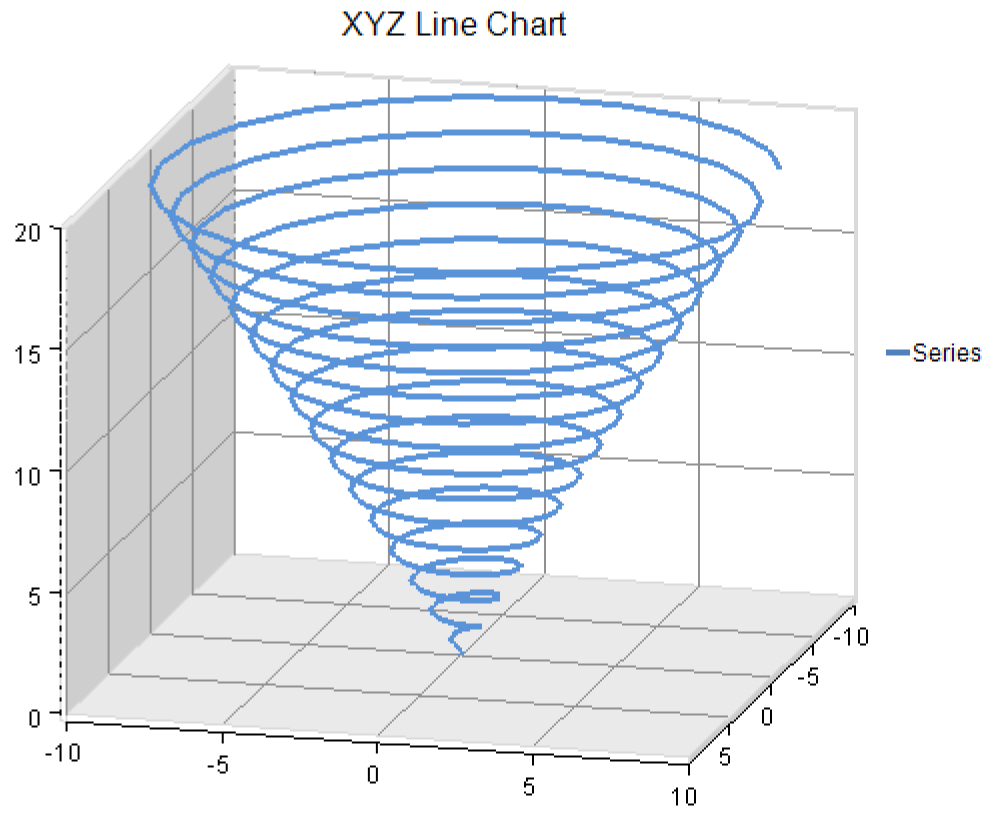
For more information on the stripe object in the API, refer to the **Stripe ('Stripe Class' in the on-line documentation)** class.

## 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.



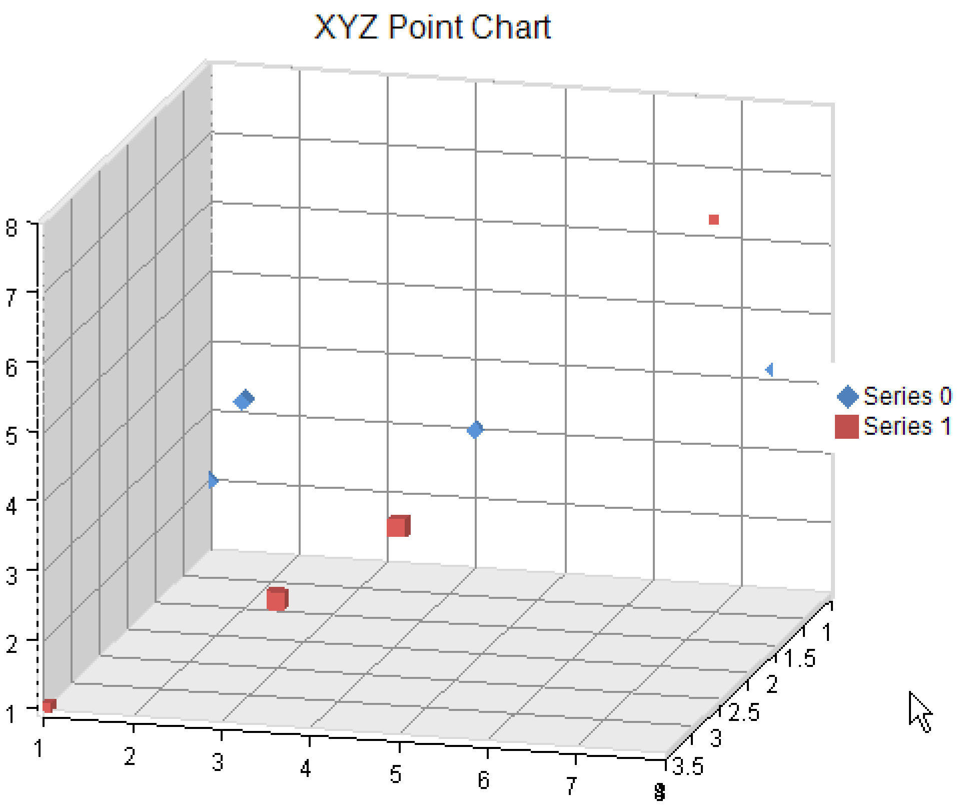
You can have any of these types of XYZ plots.

- **XYZ Point Charts**
- **XYZ Line Charts**
- **XYZ Surface Charts**
- **XYZ Stripe Charts**

For details on the API, see the **XYZPlotArea ('XYZPlotArea Class' in the on-line documentation)** class.

## XYZ Point Charts

The point chart can be an XYZ plot such as the one shown in this figure.



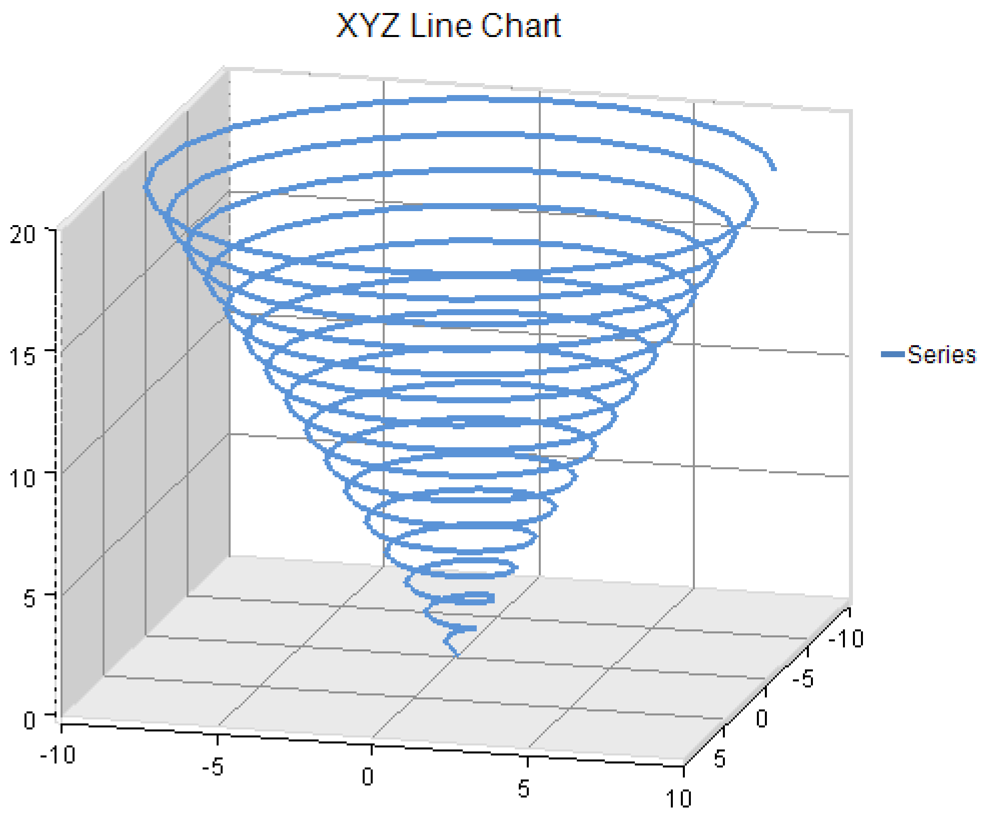
Each point in an XYZ point series has three data values: x, y, and z. Each point is visualized as point marker.

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.

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 line chart can be an XYZ plot such as the one shown in this figure.



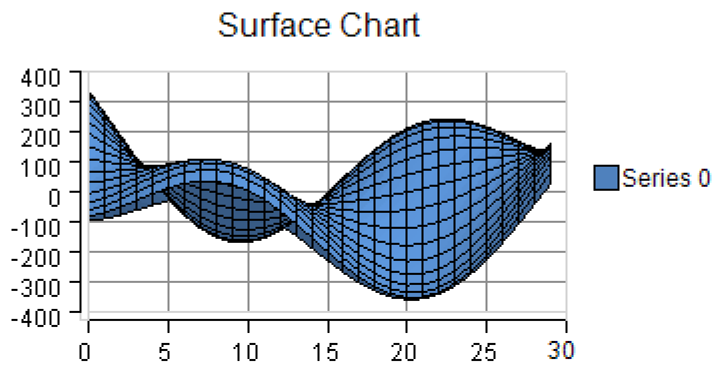
Each point in an XYZ line series has three data values: x, y, and z. Each point is visualized as a point on a line.

An XYZ line series or each point in the series can be assigned a border or a fill effect 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.

For more information on the line 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.



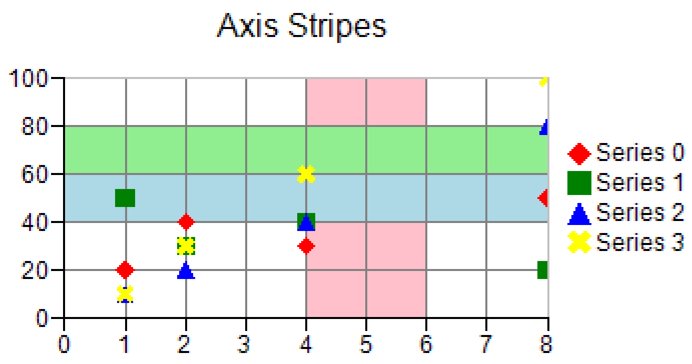
Each point in a XYZ series has three data values: x, y, and z. Each point is visualized as a point on a surface.

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.

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.

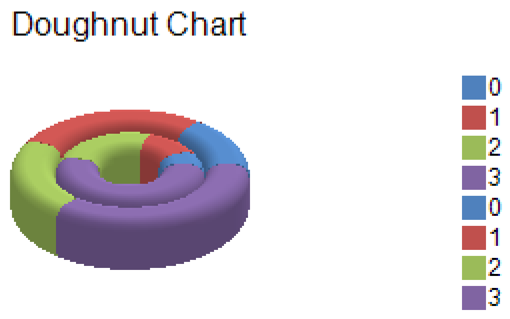


You can specify a start and end value for the stripe on the chart axis. You can also specify a fill type and color for the stripe. A stripe can be added to a chart with the axis properties in the appropriate plot area class.

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:



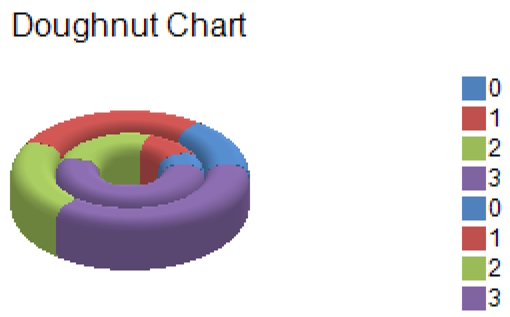
You can have any of these types of Pie plots.

- **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.



Each point in a pie series has a single data value. Each point is visualized as a pie slice.

The **HoleSize** property is used to create the doughnut chart. If this property is not set when using the **PieSeries ('PieSeries Class' in the on-line documentation)** class, the chart would be a pie chart.

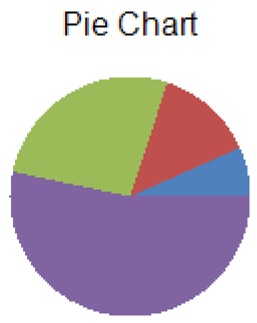
A pie series can be assigned a border and fill effect for the pie slices. Assigning null for a border or fill effect indicates that the property is null.

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 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.



Each point in a pie series has a single data value. Each point is visualized as a pie slice.

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.

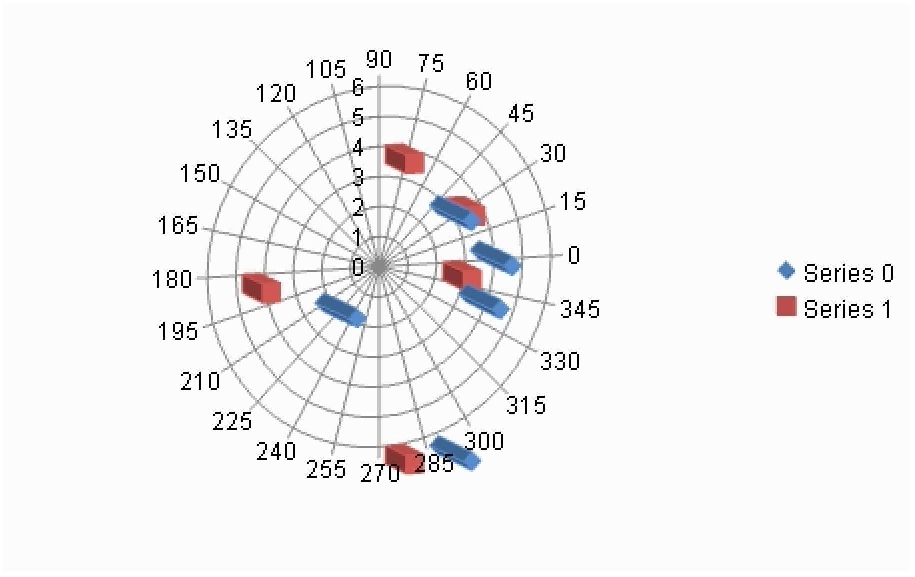
For more information on the pie series object in the API, refer to the **PieSeries ('PieSeries Class' in the on-line documentation)** class.

## 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. The Depth property in the plot area class was used to add depth to the point markers.



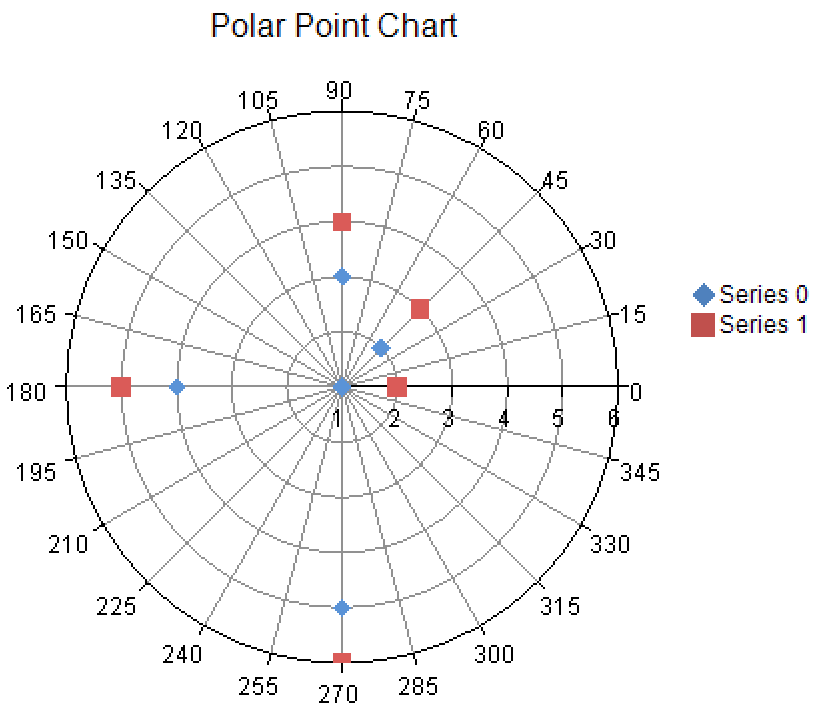
You can have any of these types of Polar plots.

- **Polar Point Charts**
- **Polar Line Charts**
- **Polar Area Charts**
- **Polar Stripe Charts**

For details on the API, see the **PolarPlotArea ('PolarPlotArea Class' in the on-line documentation)** class.

## Polar Point Charts

The point chart can be a polar plot such as the one shown in the figure:



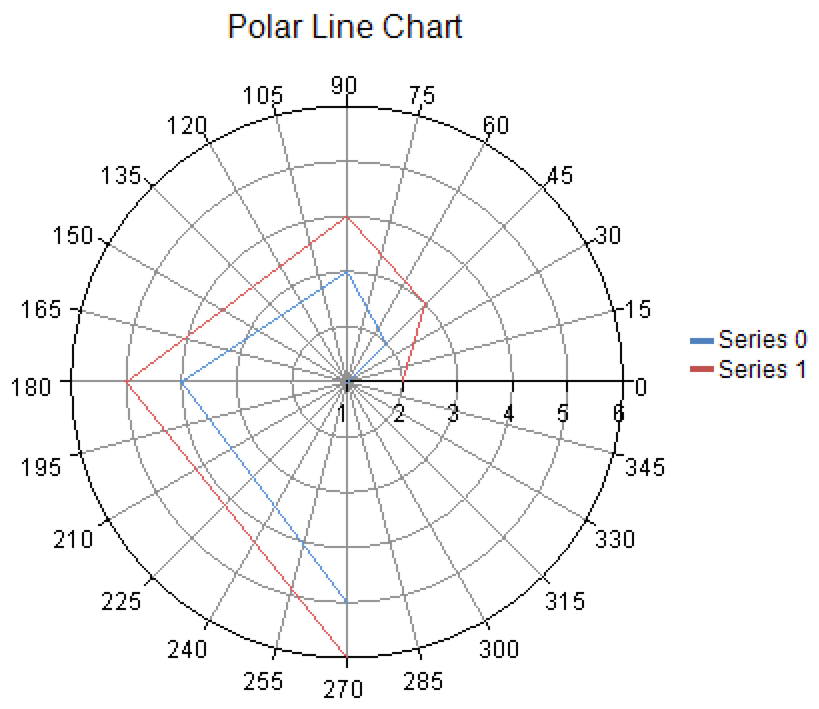
Each point has two data values: x (angle) and y (radius). Each point is visualized as a point marker.

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).

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

The line chart can be a polar plot such as the one shown in the following figure:



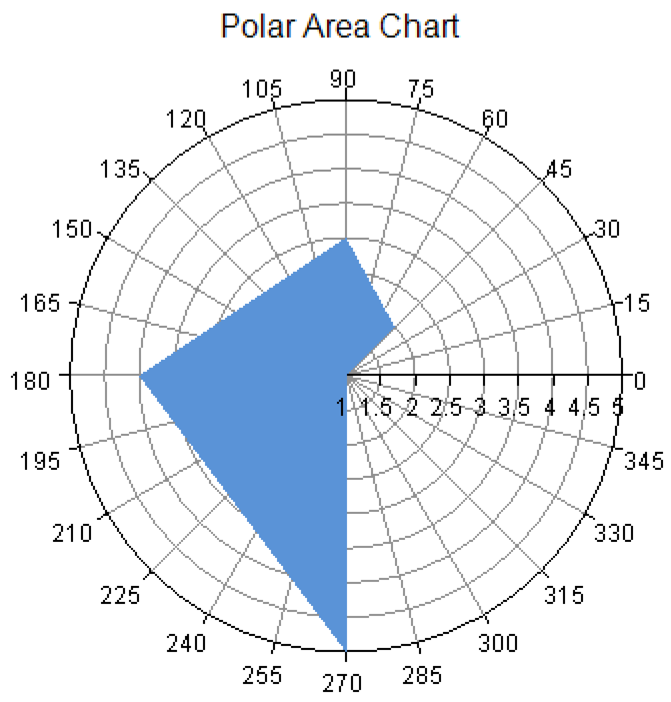
Each point has two data values: x (angle) and y (radius). Each point is visualized as a point on a line.

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).

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

The area chart can be a polar plot such as the one shown in the following figure:



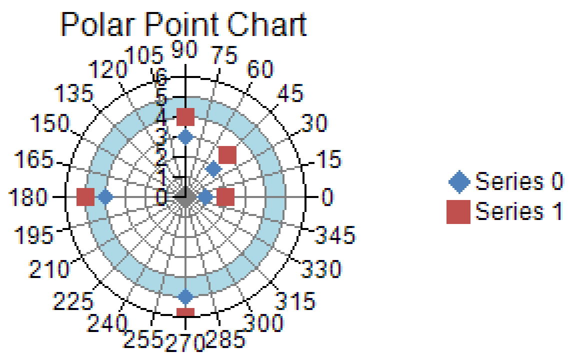
Each point has two data values: x (angle) and y (radius). Each point is visualized as a point on an area.

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).

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.



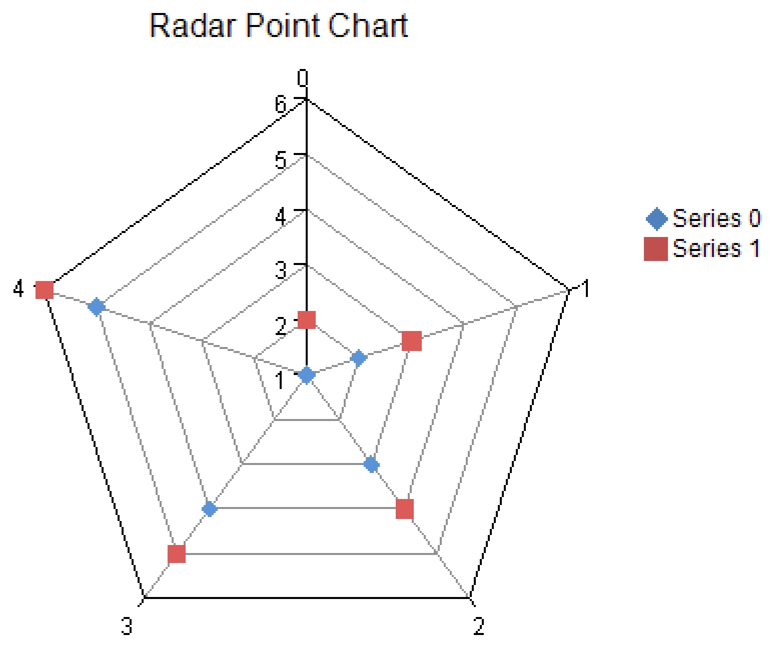
You can specify a start and end value for the stripe on the chart axis. You can also specify a fill type and color for the stripe. A stripe can be added to a chart with the axis properties in the appropriate plot area class.

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.



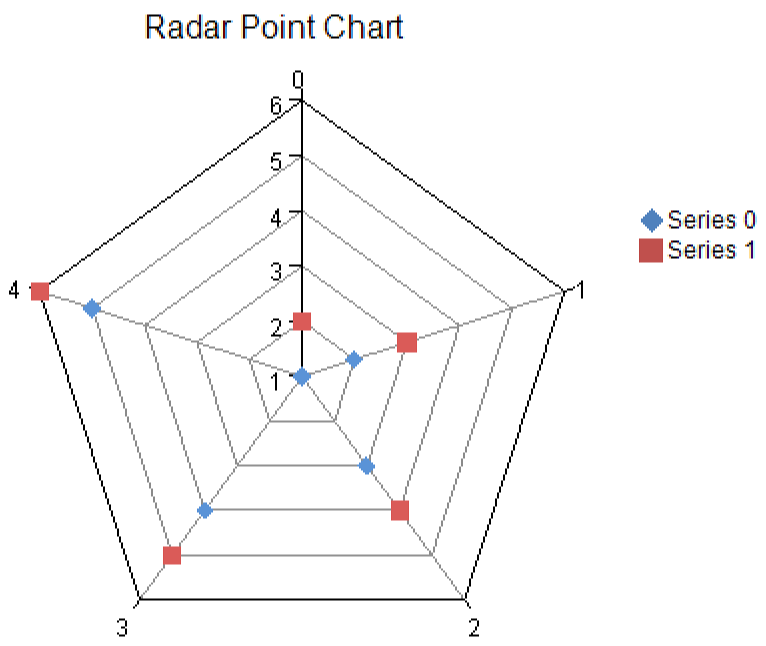
You can have any of these types of Radar plots.

- **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 chart can be a radar plot such as the one shown in this figure:



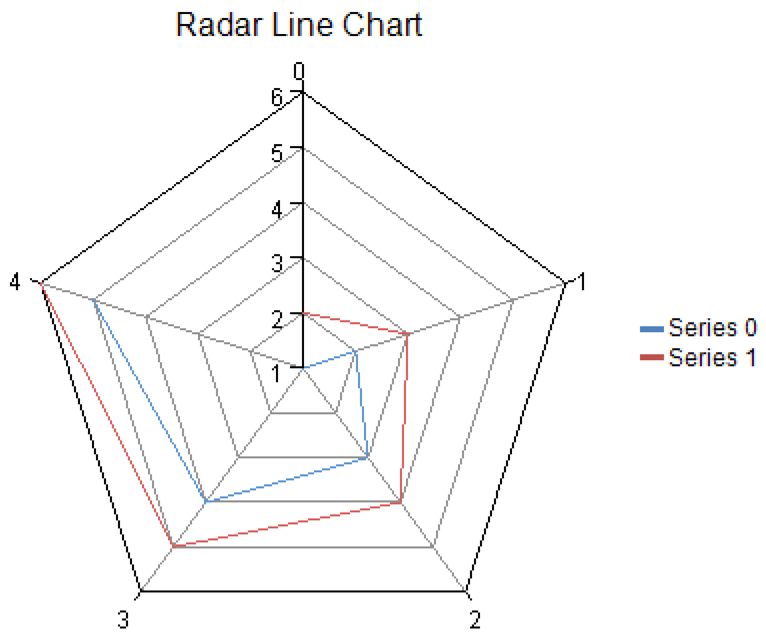
Each point has a single data value: y. Each point is visualized as a point marker.

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).

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

The line chart can be a radar plot such as the one shown in this figure:



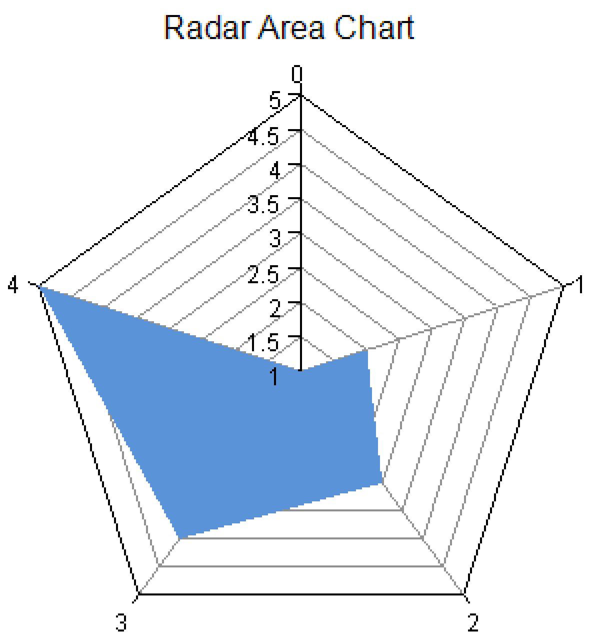
Each point has a single data value: y. Each point is visualized as point on a line.

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).

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

The area chart can be a radar plot such as the one shown in this figure:



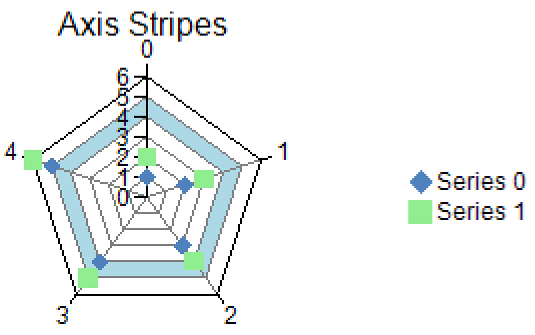
Each point has a single data value: y. Each point is visualized as a point on an area.

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).

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.

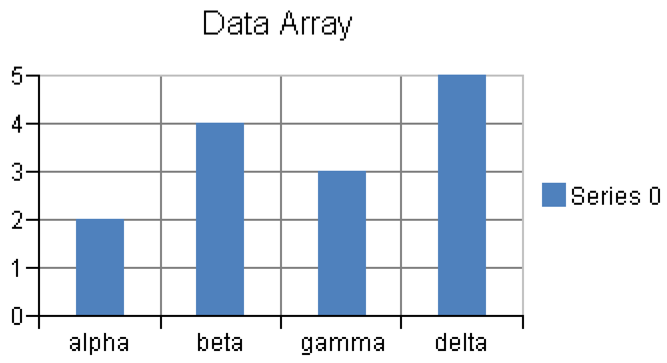
For more information on the stripe object in the API, refer to the **Stripe ('Stripe Class' in the on-line documentation)** class.

## Data Plot Types

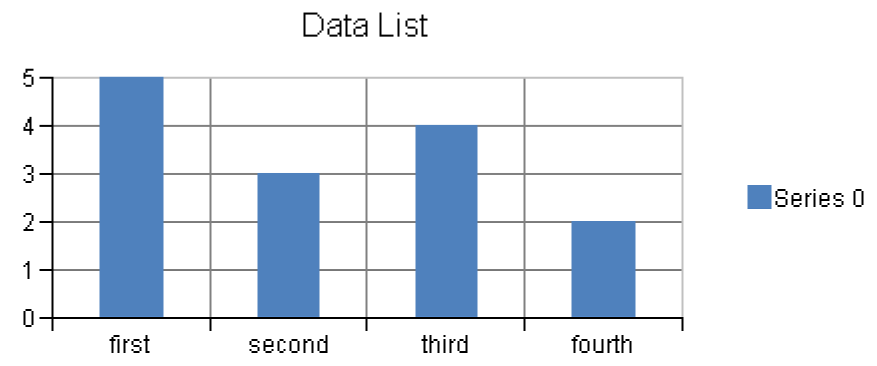
Data charts are charts that can be bound. Any series in any of the plot types can be bound to a data source using the data source property in the series class. You can use any of these types of data sources.

- Array
- List
- Table

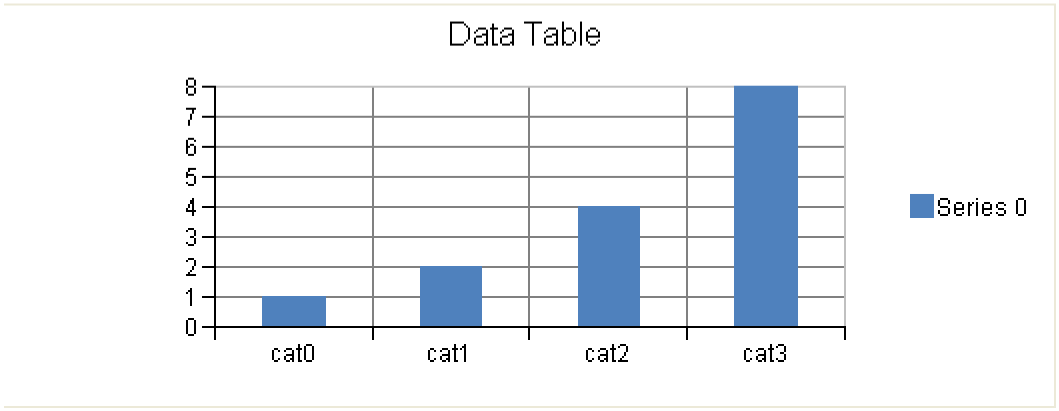
The array data chart can be a one-dimensional plot such as the one shown in this figure. This array chart shows the bars alongside each other vertically.



The following is an example of a bar chart bound to a list data source:



The following is an example of a bar chart bound to a table data source:



## Plots and Series

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 subtypes of plot areas: Y, XY, XYZ, Pie, Polar, Radar, Data, Sunburst, and Treemap.

A series is a collection of data points that are displayed in the plot area. 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.

See the following classes for more information:

- **AreaSeries ('AreaSeries Class' in the on-line documentation)**
- **BarSeries ('BarSeries Class' in the on-line documentation)**

- **BoxWhiskerSeries ('BoxWhiskerSeries Class' in the on-line documentation)**
- **FunnelSeries ('FunnelSeries 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)**
- **HistogramSeries ('HistogramSeries Class' in the on-line documentation)**
- **LineSeries ('LineSeries Class' in the on-line documentation)**
- **ParetoSeries ('ParetoSeries Class' in the on-line documentation)**
- **PieSeries ('PieSeries Class' in the on-line documentation)**
- **PolarSeries ('PolarSeries Class' in the on-line documentation)**
- **PointSeries ('PointSeries Class' in the on-line documentation)**
- **RadarSeries ('RadarSeries Class' in the on-line documentation)**
- **SunburstSeries ('SunburstSeries Class' in the on-line documentation)**
- **TreemapSeries ('TreemapSeries Class' in the on-line documentation)**
- **WaterfallSeries ('WaterfallSeries Class' in the on-line documentation)**
- XYSeries
- **XYZSeries ('XYZSeries Class' in the on-line documentation)**
- **YSeries ('YSeries Class' in the on-line documentation)**

**Using Code**

The following example adds a series to a plot area.

**Example**

The following example adds a series to a plot area.

**C#**

```
BarSeries series = new 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);
YPlotArea plotArea = new YPlotArea();
plotArea.Location = new PointF(0.2f, 0.2f);
plotArea.Size = new SizeF(0.6f, 0.6f);
plotArea.Series.Add(series);
```

**VB**

```
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)
```

**Using the Chart Designer**

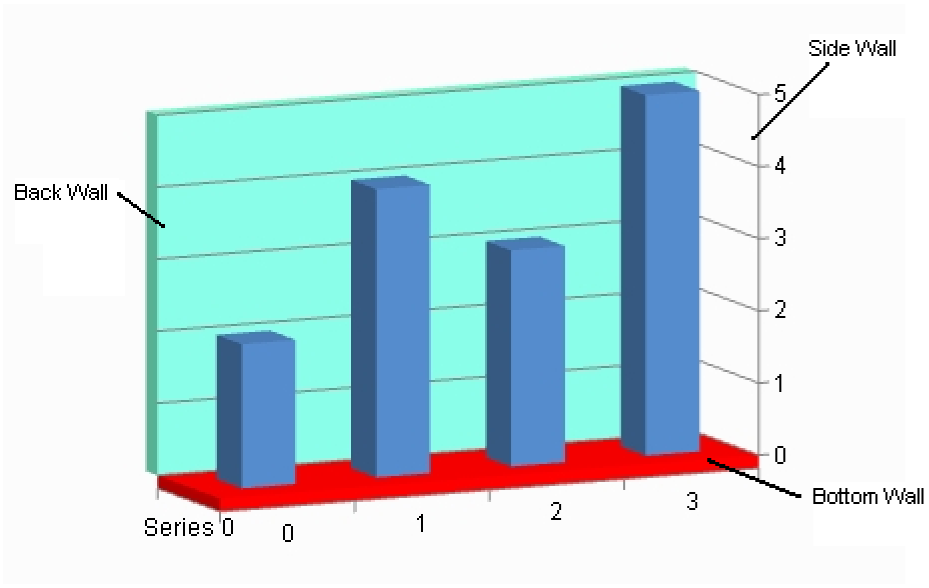
1. Select the **PlotArea Collection** editor.
2. Select the **Series Collection** editor.
3. Set properties as needed.

## 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.

The following image displays a chart with back, bottom, and side walls.



See the following for more information on how to set properties for walls:

- **Wall ('Wall Class' in the on-line documentation)**
- XYPlotArea

**Using Code**

Use properties in the plot area classes to set options for the walls.

**Example**

The following example sets properties for the wall.

**C#**

```
YPlotArea plotArea = new YPlotArea();
plotArea.BackWall.Visible = true;
```

**VB**

```
Dim plotArea As New FarPoint.Win.Chart.YPlotArea()
```

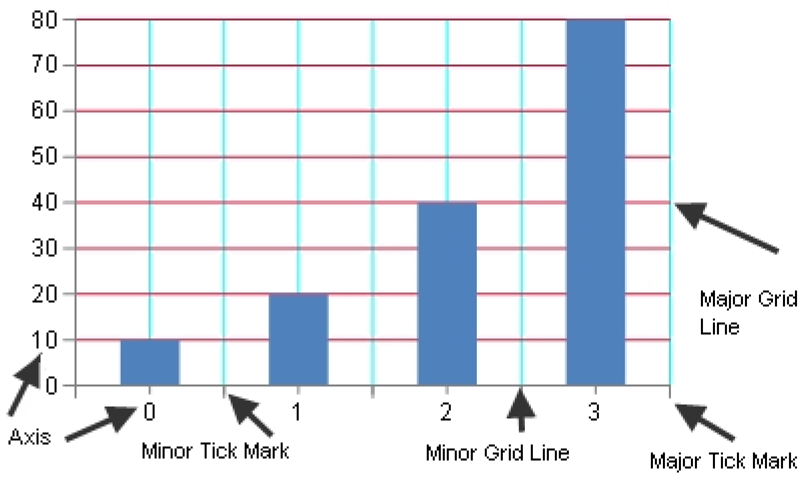
```
plotArea.BackWall.Visible = True
```

**Using the Chart Designer**

1. Select the **PlotArea Collection** editor.
2. Click the drop-down button on the right side of the **Add** button (lower, left side of dialog) to change the plot type.
3. Set the wall properties as needed.

## Axis and Other Lines

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.

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.

Units can be assigned for major and minor tick marks and tick mark labels. The units can also be automatically generated for the chart view. Length (measured in model units), can also be set for major and minor tick marks. The units can be automatically generated.

An axis can be assigned a collection of stripes. A stripe represents a range of values on the axis and is used to highlight a range of values on a given axis. A stripe is associated with an axis, but is painted on a wall.

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. 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). For more information, see the following classes:

- **IndexAxis ('IndexAxis Class' in the on-line documentation)**
- ValueAxis

Markers represent a data point and can have many shapes. For more information, see the **MarkerShape ('MarkerShape Enumeration' in the on-line documentation)** enumeration.

Stripes are used to highlight a range of values on a given axis. A stripe is associated with an axis, but is painted on a wall.

**Using Code**

Use properties in the plot area classes to set axis options.

**Example**

The following example sets a title for the axis.

**C#**

```
YPlotArea plotArea = new YPlotArea();
plotArea.Location = new PointF(0.2f, 0.2f);
plotArea.Size = new SizeF(0.6f, 0.6f);
plotArea.XAxis.Title = "Categories";
plotArea.YAxis[0].Title = "Values";
```

**VB**

```
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.YAxis(0).Title = "Values"
```

**Using the Chart Designer**

1. Select the **PlotArea** from the **Format** menu.
2. Select the **XAxis** and **YAxis Collections**.
3. Set the Title and other properties as needed.

## 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 elements in the chart that can have fill effects are label, legend, wall, stripe, and the chart itself.

The following fill effects are available in the **Fill ('Fill Class' in the on-line documentation)** class:

- NoFill
- SolidFill
- ImageFill
- GradientFill

You can fill elements using the **Fill** property in the following classes:

- **ChartModel ('ChartModel Class' in the on-line documentation)**
- **LabelArea ('LabelArea Class' in the on-line documentation)**
- **LegendArea ('LegendArea Class' in the on-line documentation)**

- **Stripe ('Stripe Class' in the on-line documentation)**
- **Wall ('Wall Class' in the on-line documentation)**

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, you can set the fill of the wall set by the **BackWall** property in the **YPlotArea ('YPlotArea Class' in the on-line documentation)** class.

**Using Code**

Use properties to set fill effects.

**Example**

The following example sets a fill effect for a bar.

**C#**

```
BarSeries series = new BarSeries();
series.BarFill = new SolidFill(Color.Red);
series.Values.Add(2.0);
series.Values.Add(4.0);
series.Values.Add(3.0);
series.Values.Add(5.0);
YPlotArea plotArea = new YPlotArea();
plotArea.Location = new PointF(0.2f, 0.2f);
plotArea.Size = new SizeF(0.6f, 0.6f);
plotArea.Series.Add(series);
ChartModel model = new ChartModel();
model.PlotAreas.Add(plotArea);
fpChart1.Model = model;
```

**VB**

```
Dim series As New FarPoint.Win.Chart.BarSeries()
series.BarFill = New 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 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 ChartModel()
model.PlotAreas.Add(plotArea)
fpChart1.Model = model
```

**Using Code**

Use properties to set fill effects.

**Example**

You can set the fill effect before or after adding the data points if you set the fill effect for the entire series.

**C#**

```
BarSeries series = new BarSeries();
series.BarFill = new SolidFill(Color.Red);
series.Values.Add(2.0);
\\ OR
BarSeries series = new BarSeries();
series.Values.Add(2.0);
series.BarFill = new SolidFill(Color.Red);
```

**VB**

```
Dim series As New FarPoint.Win.Chart.BarSeries()
series.BarFill = New SolidFill(Color.Red)
series.Values.Add(2.0)
' OR
Dim series As New FarPoint.Win.Chart.BarSeries()
series.Values.Add(2.0)
series.BarFill = New SolidFill(Color.Red)
```

**Using Code**

Use properties to set fill effects.

**Example**

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#**

```
BarSeries series = new BarSeries();
series.Values.Add(2.0);
series.Values.Add(4.0);
series.BarFills.Add(new SolidFill(Color.Green));
```

**VB**

```
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))
```

**Using Code**

Use properties to set fill effects.

**Example**

You can assign fill effects for lines and markers as well. For example:

**C#**

```
PointSeries series = new PointSeries();
series.PointFill = new SolidFill(Color.Lime);
series.PointBorder = new SolidLine(Color.Red);
series.PointMarker = new BuiltinMarker(MarkerShape.Triangle, 10.0f);
series.Values.Add(2.0);
series.Values.Add(4.0);
series.Values.Add(3.0);
series.Values.Add(5.0);
```

**VB**

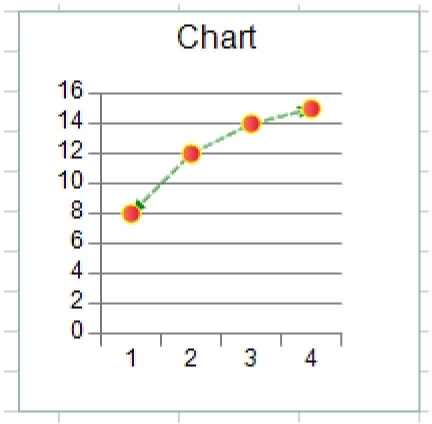
```
Dim series As New PointSeries()
series.PointFill = New SolidFill(Color.Lime)
series.PointBorder = New SolidLine(Color.Red)
series.PointMarker = New BuiltinMarker(MarkerShape.Triangle, 10F)
series.Values.Add(2.0)
series.Values.Add(4.0)
series.Values.Add(3.0)
series.Values.Add(5.0)
```

**Using the Chart Designer**

1. Select the **Fill** option.
2. Set properties as needed.

## 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.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);
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)
```

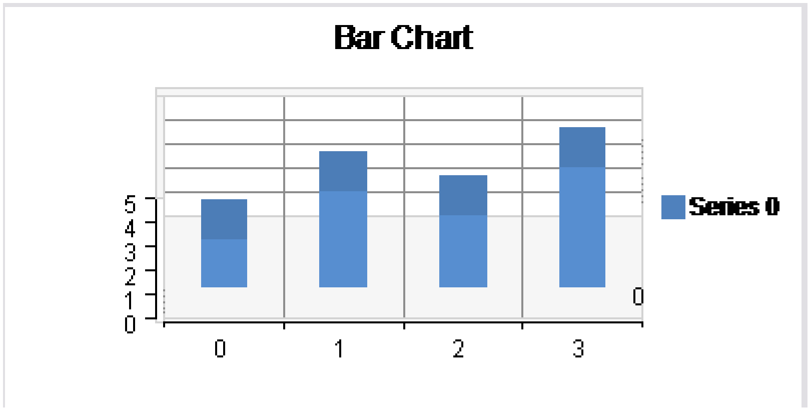
**Using the Chart Designer**

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

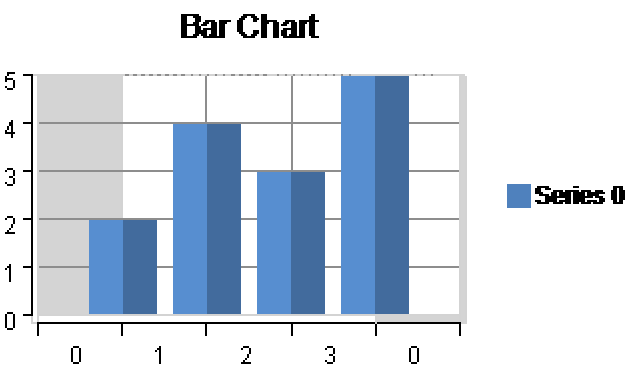
You can specify the elevation or rotation for a chart.

The elevation rotates the graph counterclockwise around the horizontal axis. The following image displays a graph with a changed elevation.



For API information, see the **Elevation ('Elevation Property' in the on-line documentation)** property.

The rotation rotates the graph counterclockwise around the vertical axis. The following image displays a graph with a changed rotation.



For API information, see the **Rotation ('Rotation Property' in the on-line documentation)** property.

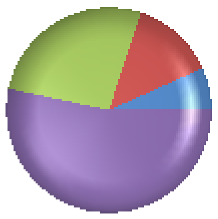
## Lighting, Shapes, and Borders

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.DirectionalLight light1 = new FarPoint.Win.Chart.DirectionalLight();
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.DirectionalLight()
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
```

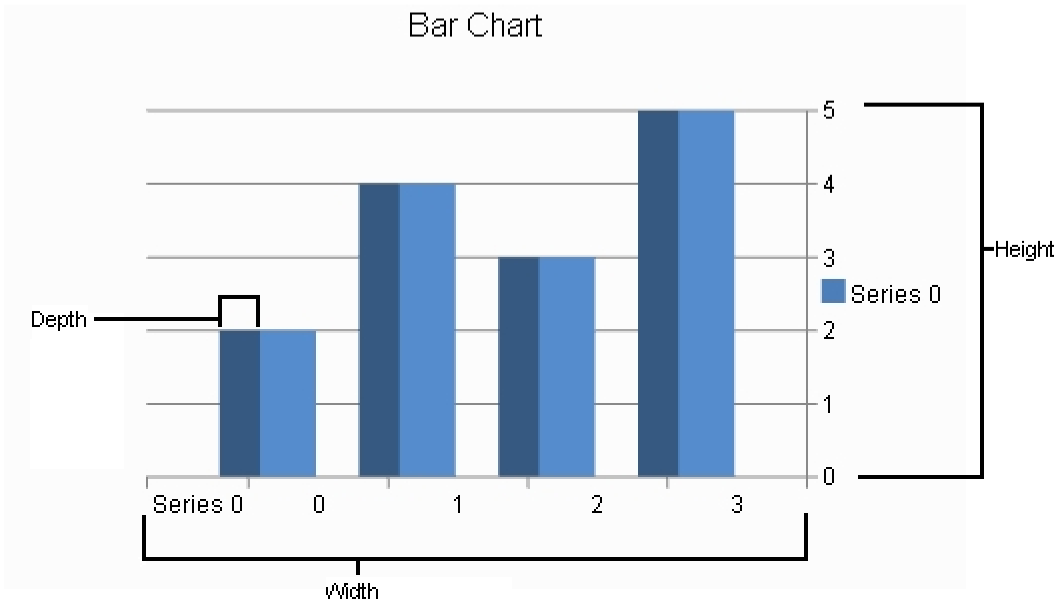
**Using the Chart Designer**

1. Select the **Plot Areas Collection**.
2. Select the **Light Collection** editor.
3. Set properties as needed.

## Size - Height, Width, and Depth

You can set the height, width, and depth for the plot area of the chart. The height of the chart is the distance from the top to the bottom of the plot area. The width of the chart is the distance from the right to the left of the plot area. The depth of the plot area is the distance from the back to the front of the chart.

In two dimensions, the height and width would be the rectangle that makes up the plot area. In three dimensions, the height, width, and depth would be the cube that makes up the plot area. The depth is the size of the cube along the z-axis. The following image shows a 3D chart.



See the following for more information:

- **Size ('Size Property' in the on-line documentation)** (width and height)
- **Depth ('Depth Property' in the on-line documentation)**

**Using Code**

Use properties to set the size and depth.

**Example**

The following example sets the size for a plot area.

**C#**

```
PiePlotArea plotArea = new PiePlotArea();
plotArea.Location = new PointF(0.2f, 0.2f);
plotArea.Size = new SizeF(0.6f, 0.6f);
plotArea.Series.Add(series);
```

**VB**

```
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)
```

**Using the Chart Designer**

1. Select the **PlotArea Collection** editor.
2. Set properties as needed.

## Labels

The labels contain the plot title and the axis labels. You can set the main title for the chart using the **Text** property in the **LabelArea ('LabelArea Class' in the on-line documentation)** class.

You can set the text, alignment, and other formatting properties for the axis labels. The label text can be bound to a datasource with the **TitleDataSource** and **TitleDataField** properties. See the following for more information:

- **YPlotArea ('YPlotArea Class' in the on-line documentation)**
- **IndexAxis ('IndexAxis Class' in the on-line documentation)**
- ValueAxis

**Using Code**

1. Create a plot area.
2. To set the title in the plot area class for the x-axis, set the IndexAxis.**Title** property.
3. Set **TitleTextFill** and **TitleTextFont** for additional formatting. You can also set the **TitleOffset**.
4. To set the title in the plot area for the y-axis, set the ValueAxis.**Title** property.
5. Set **TitleTextFill** and **TitleTextFont** for additional formatting. You can also set the **TitleOffset**.

**Example**

The following example sets a title for the axis.

**C#**

```
YPlotArea plotArea = new YPlotArea();
plotArea.Location = new PointF(0.2f, 0.2f);
plotArea.Size = new SizeF(0.6f, 0.6f);
plotArea.XAxis.Title = "Categories";
plotArea.XAxis.TitleTextFont = new System.Drawing.Font("Arial", 12);
plotArea.XAxis.TitleTextFill = new FarPoint.Win.Chart.SolidFill(Drawing.Color.Crimson);
plotArea.YAxes[0].Title = "Values";
plotArea.YAxes[0].TitleTextFont = new System.Drawing.Font("Comic Sans MS", 12);
plotArea.YAxes[0].TitleTextFill = new
FarPoint.Win.Chart.SolidFill(Drawing.Color.Chartreuse);
```

**VB**

```
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.XAxis.TitleTextFont = New System.Drawing.Font("Arial", 12)
plotArea.XAxis.TitleTextFill = New FarPoint.Win.Chart.SolidFill(Drawing.Color.Crimson)
plotArea.YAxes(0).Title = "Values"
plotArea.YAxes(0).TitleTextFont = New System.Drawing.Font("Comic Sans MS", 12)
plotArea.YAxes(0).TitleTextFill = New
FarPoint.Win.Chart.SolidFill(Drawing.Color.Chartreuse)
```

**Using the Chart Designer**

1. Select the **PlotArea** from the **Format** menu.
2. Select the **XAxis** and **YAxis Collections**.
3. Set the Title and other properties as needed

## 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:

- **LegendArea ('LegendArea Class' in the on-line documentation)**
- **LegendAreaCollection ('LegendAreaCollection Class' in the on-line documentation)**

**Using Code**

Use location and alignment properties in the legend area classes to set the legend.

**Example**

The following example sets properties for the legend.

**C#**

```
LegendArea legend = new LegendArea();
legend.Location = new PointF(0.98f, 0.5f);
legend.AlignmentX = 1.0f;
legend.AlignmentY = 0.5f;
```

**VB**

```
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. Select the **Legend Area Collection** editor.
2. Set properties as needed.

## 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:

- **Creating Plot Types**
- **Connecting to Data**
- **Saving or Loading a Chart**
- **Using the Chart Designer**
- **Using the Chart Control**

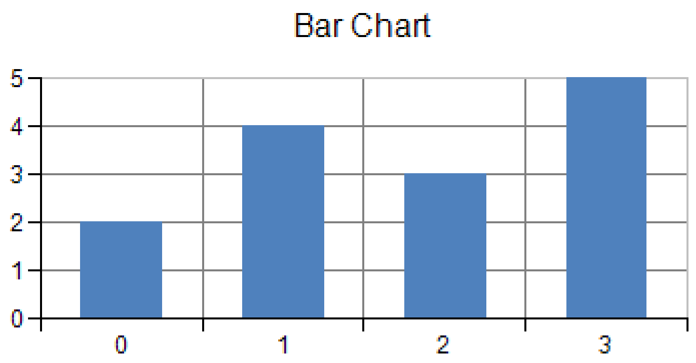
## Creating Plot Types

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**
- **Combining Plot Types**
- **Creating a Sunburst Chart (on-line documentation)**
- **Creating a Treemap Chart (on-line documentation)**

## 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:

- **AreaSeries ('AreaSeries Class' in the on-line documentation)**
- **BarSeries ('BarSeries Class' in the on-line documentation)**
- **BoxWhiskerSeries ('BoxWhiskerSeries Class' in the on-line documentation)**
- **FunnelSeries ('FunnelSeries 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)**
- **HistogramSeries ('HistogramSeries Class' in the on-line documentation)**
- **LineSeries ('LineSeries Class' in the on-line documentation)**
- **ParetoSeries ('ParetoSeries Class' in the on-line documentation)**
- **PointSeries ('PointSeries Class' in the on-line documentation)**
- **WaterfallSeries ('WaterfallSeries Class' in the on-line documentation)**
- **YSeries ('YSeries Class' in the on-line documentation)**

**Using Code**

1. Use the **BarSeries ('BarSeries Class' in the on-line documentation)** class to add data to a Chart control.
2. Use the **YPlotArea ('YPlotArea Class' in the on-line documentation)** class to create a plot area.
3. Set the location and size of the plot area.
4. Add the series to the plot area.
5. Create a label and legend for the chart.

6. Create a chart model and add the plot area, label, and legend to the model.
7. Create a chart and add the chart model to it.

**Example**

The following example demonstrates creating a Y Plot chart and adding unbound data to the control.

**C#**

```
BarSeries series = new 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);
YPlotArea plotArea = new YPlotArea();
plotArea.Location = new PointF(0.2f, 0.2f);
plotArea.Size = new SizeF(0.6f, 0.6f);
plotArea.Series.Add(series);
LabelArea label = new LabelArea();
label.Text = "Bar Chart";
label.Location = new PointF(0.5f, 0.02f);
label.AlignmentX = 0.5f;
label.AlignmentY = 0.0f;
LegendArea legend = new LegendArea();
legend.Location = new PointF(0.98f, 0.5f);
legend.AlignmentX = 1.0f;
legend.AlignmentY = 0.5f;
ChartModel model = new ChartModel();
model.LabelAreas.Add(label);
model.LegendAreas.Add(legend);
model.PlotAreas.Add(plotArea);
chart2DControl1.Model = model;
```

**VB**

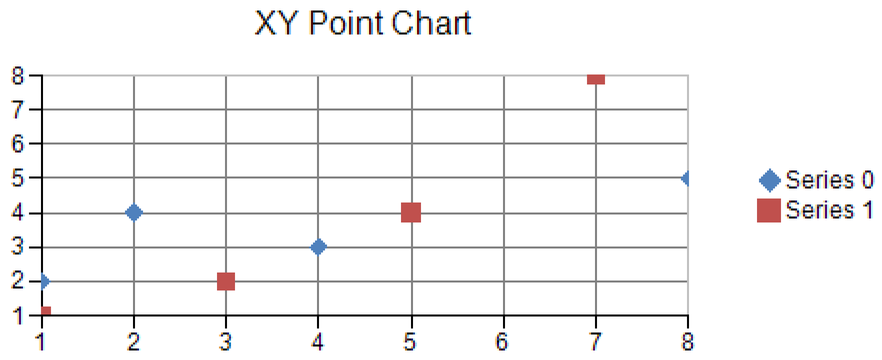
```
Dim series As New FarPoint.Win.Chart.BarSeries()
series.SeriesName = "Series 0"
series.Values.Add(2.0)
series.Values.Add(4.0)
series.Values.Add(3.0)
series.Values.Add(5.0)
Dim plotArea As New FarPoint.Win.Chart.YPlotArea()
plotArea.Location = New PointF(0.2F, 0.2F)
plotArea.Size = New SizeF(0.6F, 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 the Chart Designer**

1. Select the **PlotArea Collection** editor.
2. Set properties as needed.

## 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 class.

The following classes are also available when creating XY plot type charts:

- **XYBubbleSeries ('XYBubbleSeries Class' in the on-line documentation)**
- **XYPointSeries ('XYPointSeries Class' in the on-line documentation)**
- **XYLineSeries ('XYLineSeries Class' in the on-line documentation)**

**Using Code**

1. Use the **XYPointSeries ('XYPointSeries Class' in the on-line documentation)** class to add data to a Chart control.
2. Use the **XYPlotArea ('XYPlotArea Class' in the on-line documentation)** class to create the plot area.
3. Set the location and size of the plot area.
4. Add the series to the plot area.
5. Create a label and legend for the chart.
6. Create a chart model and add the plot area, label, and legend to the model.
7. Create a chart and add the chart model to it.

**Example**

The following example demonstrates using unbound data to create an XY point chart.

**C#**

```
XYPointSeries series0 = new 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);
XYPointSeries series1 = new 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);
XYPlotArea plotArea = new 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);
LabelArea label = new LabelArea();
label.Text = "XY Point Chart";
label.Location = new PointF(0.5f, 0.02f);
label.AlignmentX = 0.5f;
label.AlignmentY = 0.0f;
LegendArea legend = new LegendArea();
legend.Location = new PointF(0.98f, 0.5f);
legend.AlignmentX = 1.0f;
legend.AlignmentY = 0.5f;
ChartModel model = new ChartModel();
model.LabelAreas.Add(label);
model.LegendAreas.Add(legend);
model.PlotAreas.Add(plotArea);
chart2DControl1.Model = model;
```

**VB**

```
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)
```

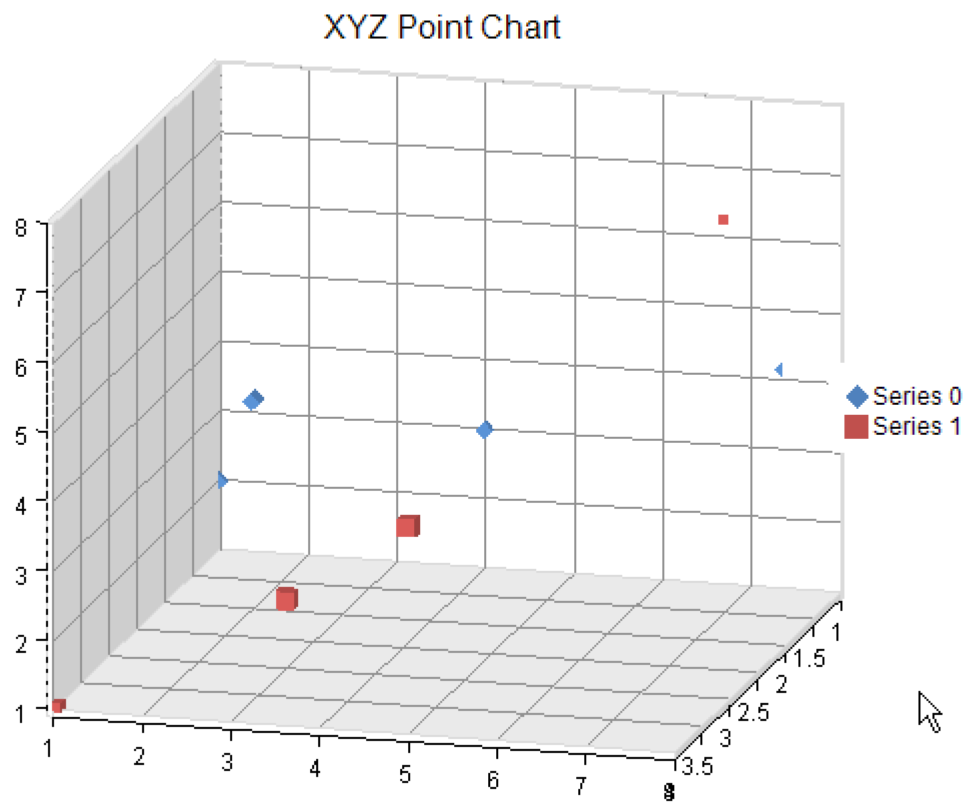
```
series1.YValues.Add(8.0)
Dim plotArea As New FarPoint.Win.Chart.XYPlotArea()
plotArea.Location = New PointF(0.2F, 0.2F)
plotArea.Size = New SizeF(0.6F, 0.6F)
plotArea.Series.Add(series0)
plotArea.Series.Add(series1)
Dim label As New FarPoint.Win.Chart.LabelArea()
label.Text = "XY Point Chart"
label.Location = 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. Select the **PlotArea Collection** editor.
2. Click the drop-down button on the right side of the **Add** button (lower, left side of dialog).
3. Select the **XYPlotArea** option and set properties as needed.

## 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:

- **XYZSeries ('XYZSeries Class' in the on-line documentation)**
- **XYZPointSeries ('XYZPointSeries Class' in the on-line documentation)**
- **XYZSurfaceSeries ('XYZSurfaceSeries Class' in the on-line documentation)**
- **XYZLineSeries ('XYZLineSeries Class' in the on-line documentation)**

**Using Code**

1. Use the **XYZPointSeries ('XYZPointSeries Class' in the on-line documentation)** class to add data to a Chart control.
2. Use the **XYZPlotArea ('XYZPlotArea Class' in the on-line documentation)** class to create the plot area.
3. Set the location and size of the plot area.
4. Add the series to the plot area.
5. Create a label and legend for the chart.
6. Create a chart model and add the plot area, label, and legend to the model.
7. Create a chart and add the chart model to it.

**Example**

The following example demonstrates using unbound data to create an XYZ point chart.

**C#**

```
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.Rotation = -21;
plotArea.Elevation = 15;
plotArea.Series.Add(series0);
plotArea.Series.Add(series1);
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);
fpChart1.Model = model;
```

**VB**

```
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)
```

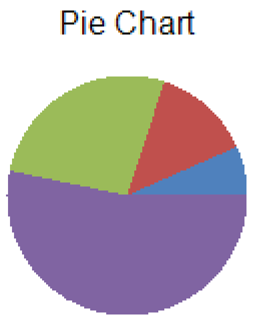
```
series0.YValues.Add(4.0)
series0.YValues.Add(3.0)
series0.YValues.Add(5.0)
series0.ZValues.Add(1.0)
series0.ZValues.Add(2.0)
series0.ZValues.Add(1.0)
series0.ZValues.Add(2.0)
Dim series1 As New FarPoint.Win.Chart.XYZPointSeries()
series1.SeriesName = "Series 1"
series1.XValues.Add(1.0)
series1.XValues.Add(3.0)
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.Elevation = 15
plotArea.Rotation = -21
plotArea.Series.Add(series0)
plotArea.Series.Add(series1)
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 = 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)
fpChart1.Model = model
```

**Using the Chart Designer**

1. Select the **PlotArea Collection** editor.
2. Click the drop-down button on the right side of the **Add** button (lower, left side of dialog).
3. Select the **XYZPlotArea** option and set properties as needed.

## 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 details on the API, see the **PiePlotArea ('PiePlotArea Class' in the on-line documentation)** class.

The following class is also available when creating Pie plot type charts:

- **PieSeries ('PieSeries Class' in the on-line documentation)**

**Using Code**

1. Use the **PieSeries ('PieSeries Class' in the on-line documentation)** class to add data to a Chart control.
2. Use the **PiePlotArea ('PiePlotArea Class' in the on-line documentation)** class to create the plot area.
3. Set the location and size of the plot area.
4. Add the series to the plot area.
5. Create a label and legend for the chart.
6. Create a chart model and add the plot area, label, and legend to the model.
7. Create a chart and add the chart model to it.

**Example**

The following example demonstrates using unbound data to create a Pie chart.

**C#**

```
PieSeries series = new PieSeries();
series.SeriesName = "Series 0";
series.Values.Add(1.0);
series.Values.Add(2.0);
series.Values.Add(4.0);
series.Values.Add(8.0);
PiePlotArea plotArea = new PiePlotArea();
plotArea.Location = new PointF(0.2f, 0.2f);
plotArea.Size = new SizeF(0.6f, 0.6f);
plotArea.Series.Add(series);
LabelArea label = new LabelArea();
label.Text = "Pie Chart";
label.Location = new PointF(0.5f, 0.02f);
label.AlignmentX = 0.5f;
label.AlignmentY = 0.0f;
LegendArea legend = new LegendArea();
legend.Location = new PointF(0.98f, 0.5f);
legend.AlignmentX = 1.0f;
legend.AlignmentY = 0.5f;
ChartModel model = new ChartModel();
model.LabelAreas.Add(label);
model.LegendAreas.Add(legend);
model.PlotAreas.Add(plotArea);
```

```
chart2DControl1.Model = model;
```

**VB**

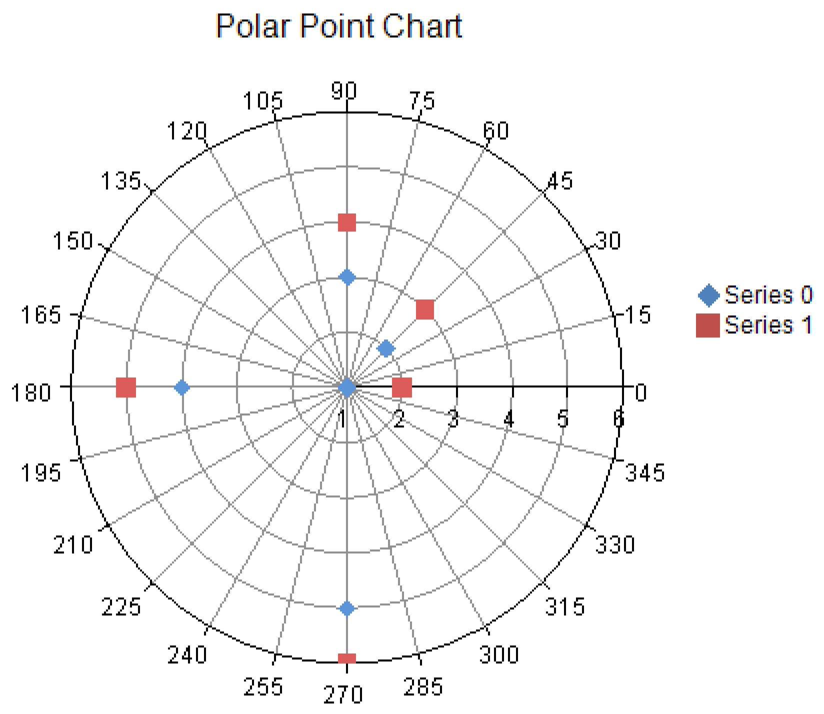
```
Dim series As New FarPoint.Win.Chart.PieSeries()
series.SeriesName = "Series 0"
series.Values.Add(1.0)
series.Values.Add(2.0)
series.Values.Add(4.0)
series.Values.Add(8.0)
Dim plotArea As New FarPoint.Win.Chart.PiePlotArea()
plotArea.Location = New PointF(0.2F, 0.2F)
plotArea.Size = New SizeF(0.6F, 0.6F)
plotArea.Series.Add(series)
Dim label As New FarPoint.Win.Chart.LabelArea()
label.Text = "Pie 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. Select the **PlotArea Collection** editor.
2. Click the drop-down button on the right side of the **Add** button (lower, left side of dialog).
3. Select the **PiePlotArea** option and set properties as needed.

## 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:

- **PolarSeries ('PolarSeries Class' in the on-line documentation)**
- **PolarAreaSeries ('PolarAreaSeries Class' in the on-line documentation)**
- **PolarPointSeries ('PolarPointSeries Class' in the on-line documentation)**
- **PolarLineSeries ('PolarLineSeries Class' in the on-line documentation)**
- **PolarAngleAxis ('PolarAngleAxis Class' in the on-line documentation)**
- **PolarRadiusAxis ('PolarRadiusAxis Class' in the on-line documentation)**

**Using Code**

1. Use the **PolarPointSeries ('PolarPointSeries Class' in the on-line documentation)** class to add data to a Chart control.
2. Use the **PolarPlotArea ('PolarPlotArea Class' in the on-line documentation)** class to create the plot area.
3. Set the location and size of the plot area.
4. Add the series to the plot area.
5. Create a label and legend for the chart.
6. Create a chart model and add the plot area, label, and legend to the model.
7. Create a chart and add the chart model to it.

**Example**

The following example demonstrates using unbound data to create 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 = FarPoint.Win.Chart.new 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;
```

### VB

```
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)
```

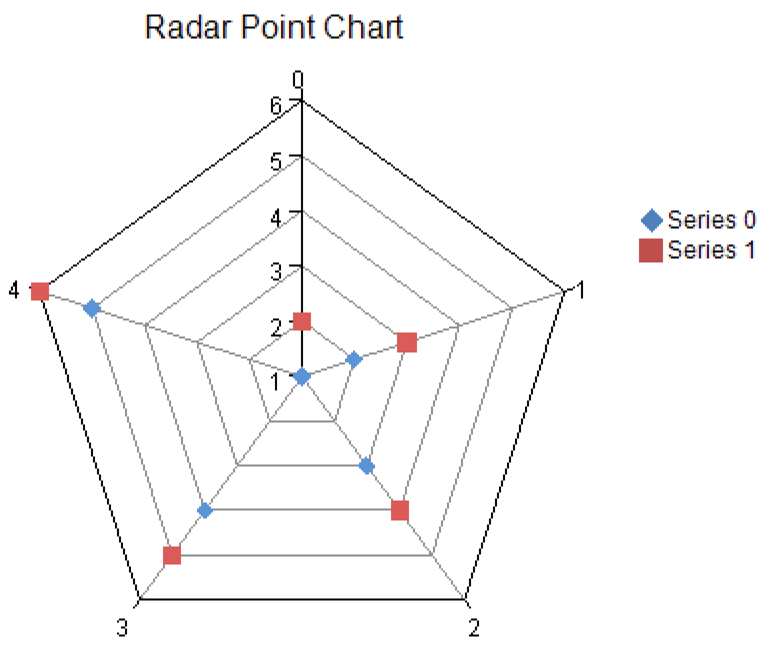
```
series0.YValues.Add(5.0)
Dim series1 As New FarPoint.Win.Chart.PolarPointSeries()
series1.SeriesName = "Series 1"
series1.XValues.Add(0.0)
series1.XValues.Add(45.0)
series1.XValues.Add(90.0)
series1.XValues.Add(180.0)
series1.XValues.Add(270.0)
series1.YValues.Add(2.0)
series1.YValues.Add(3.0)
series1.YValues.Add(4.0)
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 = 1F
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. Select the **PlotArea Collection** editor.
2. Click the drop-down button on the right side of the **Add** button (lower, left side of dialog).
3. Select the **PolarPlotArea** option and set properties as needed.

## 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:

- **RadarSeries ('RadarSeries Class' in the on-line documentation)**
- **PolarLineSeries ('PolarLineSeries Class' in the on-line documentation)**
- **RadarAreaSeries ('RadarAreaSeries Class' in the on-line documentation)**
- **RadarPointSeries ('RadarPointSeries Class' in the on-line documentation)**
- **RadarIndexAxis ('RadarIndexAxis Class' in the on-line documentation)**
- **RadarValueAxis ('RadarValueAxis Class' in the on-line documentation)**

**Using Code**

1. Use the **RadarPointSeries ('RadarPointSeries Class' in the on-line documentation)** class to add data to a Chart control.
2. Use the **RadarPlotArea ('RadarPlotArea Class' in the on-line documentation)** class to create the plot area.
3. Set the location and size of the plot area.
4. Add the series to the plot area.
5. Create a label and legend for the chart.
6. Create a chart model and add the plot area, label, and legend to the model.
7. Create a chart and add the chart model to it.

**Example**

The following example demonstrates using unbound data to create 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;
```

## VB

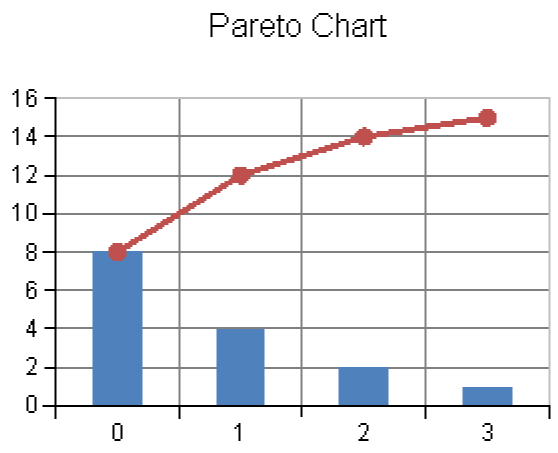
```
Dim series0 As New FarPoint.Win.Chart.RadarPointSeries()
series0.SeriesName = "Series 0"
series0.Values.Add(1.0)
series0.Values.Add(2.0)
series0.Values.Add(3.0)
series0.Values.Add(4.0)
series0.Values.Add(5.0)
Dim series1 As New FarPoint.Win.Chart.RadarPointSeries()
series1.SeriesName = "Series 1"
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 = 0F
Dim legend As New FarPoint.Win.Chart.LegendArea()
legend.Location = New PointF(0.98F, 0.5F)
legend.AlignmentX = 1F
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. Select the **PlotArea Collection** editor.
2. Click the drop-down button on the right side of the **Add** button (lower, left side of dialog).
3. Select the **RadarPlotArea** option and set properties as needed.

## Combining Plot Types

Multiple series from the same major category are compatible with each other and can be combined in a single plot area. For example, a bar series and a line series can be combined together in a YPlotArea.



For details on the API, see the **YPlotArea ('YPlotArea Class' in the on-line documentation)** class.

The following classes are used to create the bar and line series example:

- **BarSeries ('BarSeries Class' in the on-line documentation)**
- **LineSeries ('LineSeries Class' in the on-line documentation)**

**Using Code**

1. Use the **BarSeries ('BarSeries Class' in the on-line documentation)** and **LineSeries ('LineSeries Class' in the on-line documentation)** classes to add data to the Chart control.
2. Use the **YPlotArea ('YPlotArea Class' in the on-line documentation)** class to create the plot area.
3. Set the location and size of the plot area.
4. Add both series to the plot area.
5. Create a label for the chart.

6. Create a chart model and add the plot area and label to the model.
7. Create a chart and add the chart model to it.

**Example**

The following example demonstrates using unbound data to create a chart that uses a bar series and a line series.

**C#**

```
BarSeries series0 = new BarSeries();
series0.Values.Add(8.0);
series0.Values.Add(4.0);
series0.Values.Add(2.0);
series0.Values.Add(1.0);
LineSeries series1 = new LineSeries();
series1.PointMarker = new BuiltinMarker(MarkerShape.Circle, 7.0f);
series1.Values.Add(8.0);
series1.Values.Add(12.0);
series1.Values.Add(14.0);
series1.Values.Add(15.0);
YPlotArea plotArea = new 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);
LabelArea label = new LabelArea();
label.Location = new PointF(0.5f, 0.02f);
label.AlignmentX = 0.5f;
label.AlignmentY = 0.0f;
label.Text = "Pareto Chart";
ChartModel model = new ChartModel();
model.LabelAreas.Add(label);
model.PlotAreas.Add(plotArea);
chart2DControl1.Model = model;
```

**VB**

```
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).

For more information on adding data to a Chart control, see the following topics:

- **Using a Bound Data Source**
- **Using an UnBound Data Source**
- **Using Raw and Represented Data**

## 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 more information about the DataSource property, refer to the specific chart type in the Assembly Reference (for example: SeriesNameDataSource in the **RadarLineSeries ('RadarLineSeries Class' in the on-line documentation)** class).

**Using Code**

Create a data source and then bind the control.

**Example**

The following example demonstrates how to bind the control to a data source.

**C#**

```
// Create an array and bind the control
object[] values = new object[] { 2, 4.0, 3.0m, "5.0" };
BarSeries series = new BarSeries();
series.Values.DataSource = values;
```

**VB**

```
' Create an array and bind the control
Dim values() As Object = {2, 4.0, 3.0D, "5.0"}
Dim series As New BarSeries()
series.Values.DataSource = values
```

**Using Code**

Create a data source and then bind the control.

**Example**

The following example demonstrates how to bind the control to a data 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;
```

**VB**

```
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 add double values to the chart control without using a datasource.

**Using Code**

Add data to the series.

**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);
```

**VB**

```
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 the Chart Designer**

1. Select the **PlotArea Collection** editor.
2. Select the **Series Collection** editor.
3. Select the **Values Collection** editor.
4. Set values as needed.

## Using Raw and Represented Data

You can set the scale of the data before displaying the data.

For example, if the data values are in the millions, you may wish to display them using a much smaller scale such as hundreds (100,000,000 vs 100). Use the **DisplayUnits** property in the ValueAxis class to set the scale.

**Using Code**

Use the **DisplayUnits** property to create a smaller scale on the axis.

**Example**

The following example uses the **DisplayUnits** property.

**C#**

```
FarPoint.Win.Chart.BarSeries series = new FarPoint.Win.Chart.BarSeries();
series.Values.Add(10000.0);
series.Values.Add(20000.0);
series.Values.Add(40000.0);
series.Values.Add(80000.0);
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.XAxis.Title = "Entry";
plotArea.XAxis.TitleVisible = true;
plotArea.YAxes[0].DisplayUnits = 1000.0;
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(200, 200);
chart.Location = new Point(100, 100);
chart.Model = model;
fpSpread1.Sheets[0].Charts.Add(chart);
```

**VB**

```
Dim series As New FarPoint.Win.Chart.BarSeries()
series.Values.Add(10000.0)
series.Values.Add(20000.0)
series.Values.Add(40000.0)
series.Values.Add(80000.0)
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.XAxis.Title = "Entry" 'IndexAxis
plotArea.XAxis.TitleVisible = True 'IndexAxis
plotArea.YAxes(0).DisplayUnits = 1000.0 'ValueAxis
plotArea.Series.Add(series)
Dim model As New FarPoint.Win.Chart.ChartModel()
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)
```

## Saving or Loading a Chart

You can read or write to a file or stream using the IXmlSerializable interface.

You can also save and load xml files at design time. Select the arrow icon at the top right edge of the control after drawing the control on the form to see the menu options.

#### Using Code

Use general methods to save to a file.

#### Example

The following code writes to a file.

### C#

```
ChartModel model = fpChart1.Model;
XmlTextWriter writer = new XmlTextWriter("c:\\home\\temp\\test.xml", null);
model.WriteXml(writer);
writer.Close();
```

### VB

```
Dim model As ChartModel = fpChart1.Model
Dim writer As New XmlTextWriter("c:\home\temp\test.xml", Nothing)
model.WriteXml(writer)
writer.Close()
```

#### Using Code

Use general methods to read from a file.

#### Example

The following code reads from a file.

### C#

```
ChartModel model = fpChart1.Model;
XmlTextReader reader = new XmlTextReader("c:\\home\\temp\\test.xml");
model.ReadXml(reader);
reader.Close();
```

### VB

```
Dim model As ChartModel = fpChart1.Model
Dim reader As New XmlTextReader("c:\home\temp\test.xml")
model.ReadXml(reader)
reader.Close()
```

#### Using the Designer

1. Select the arrow icon at the top right edge of the control after drawing the control on the form to see the menu options.
2. Choose **Save to XML** or **Load from XML**.

## Using the Chart Designer

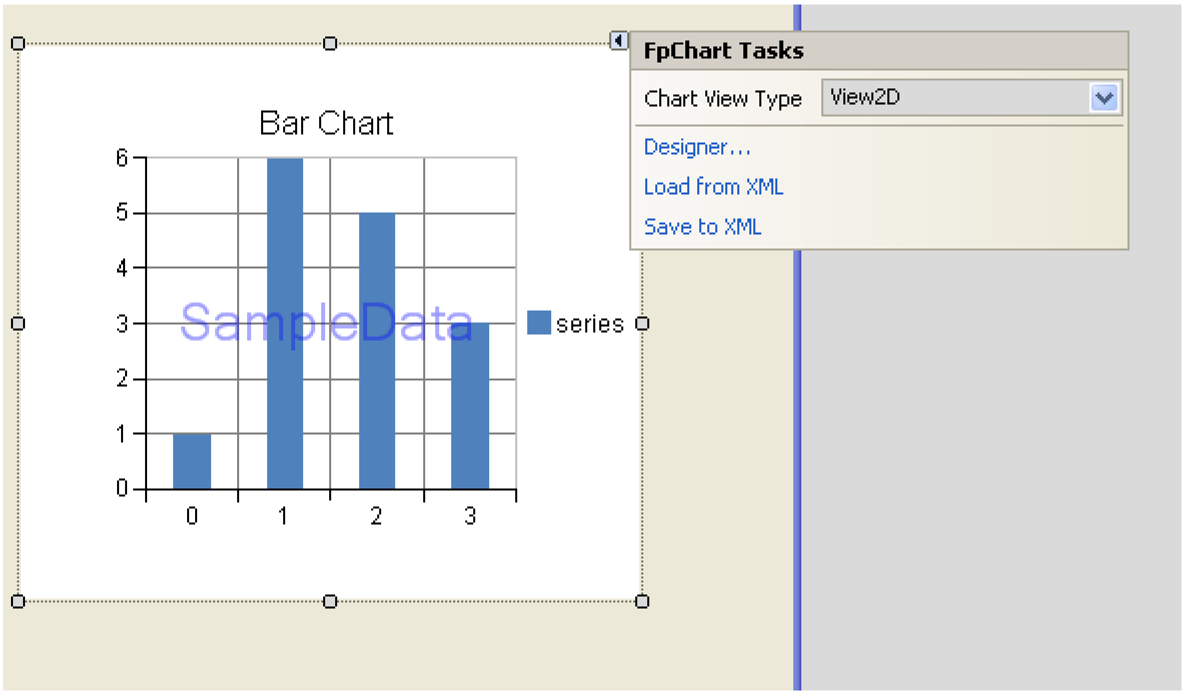
The chart designer graphical interface saves time and effort and provides a visual representation of the chart control as you change settings in the designer. You can apply the changes to the control or save the changes to an XML formatted file.

Here are topics to help you use the Chart Designer:

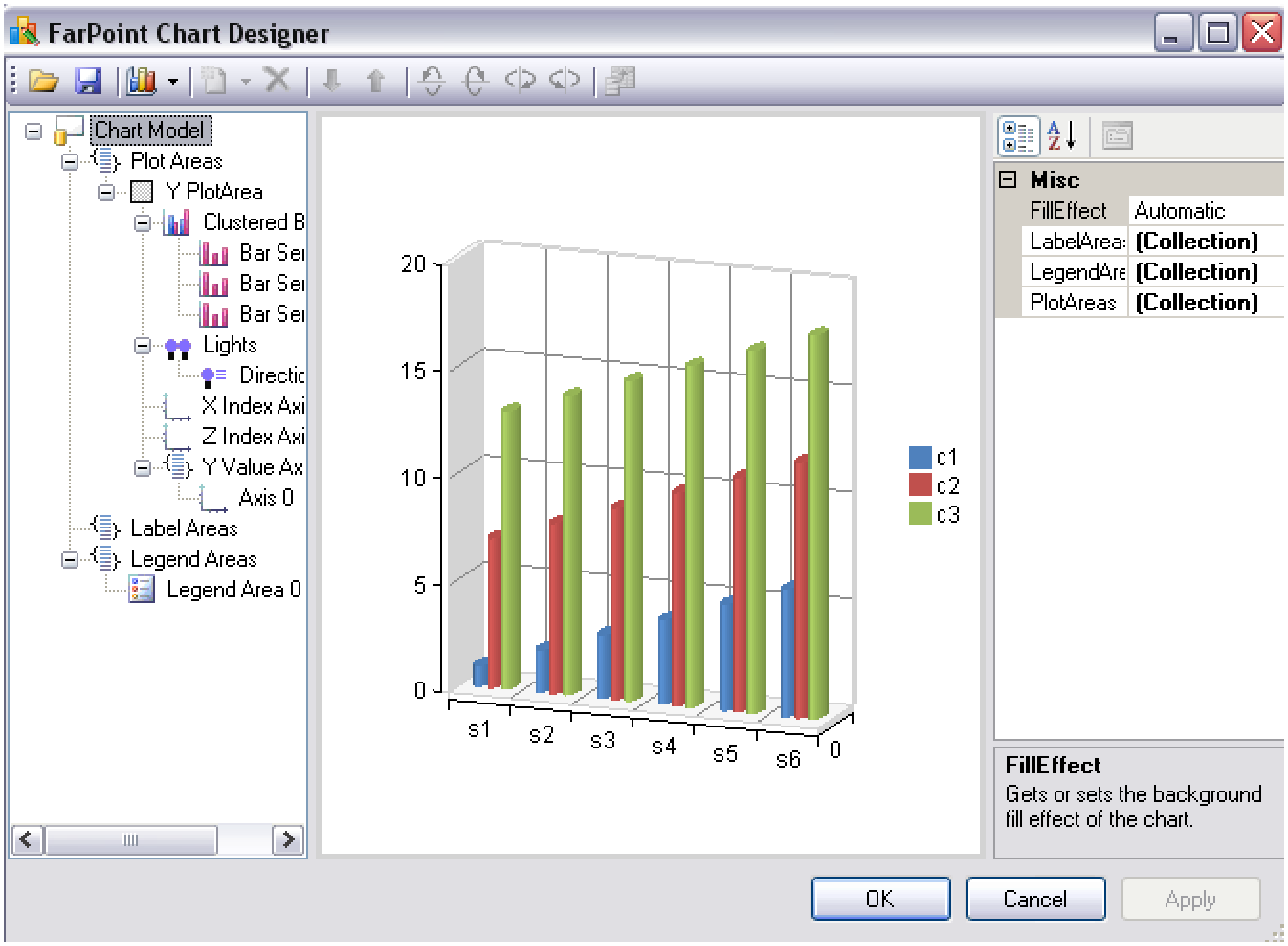
- **Opening the Chart Designer**
- **Creating a Chart Control**
- **Chart Collection Editors**
- **Chart Designer Toolbar**

## Opening the Chart Designer

You can open the designer by clicking on the arrow at the top right edge of the control when it is on a form in Visual Studio. Then click on the **Designer** menu option to load the chart designer.



This opens the designer with the various editors that can be used to customize the chart control.



| | |
|---|---|
| FillEffect | Automatic |
| LabelArea: | (Collection) |
| LegendAre | (Collection) |
| PlotAreas | (Collection) |

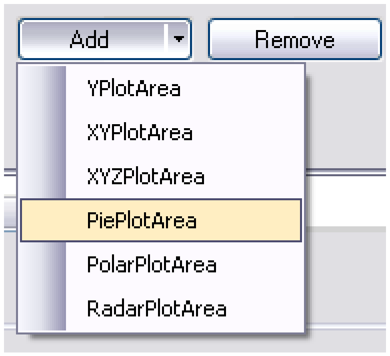
**FillEffect**
Gets or sets the background fill effect of the chart.

## Creating a Chart Control

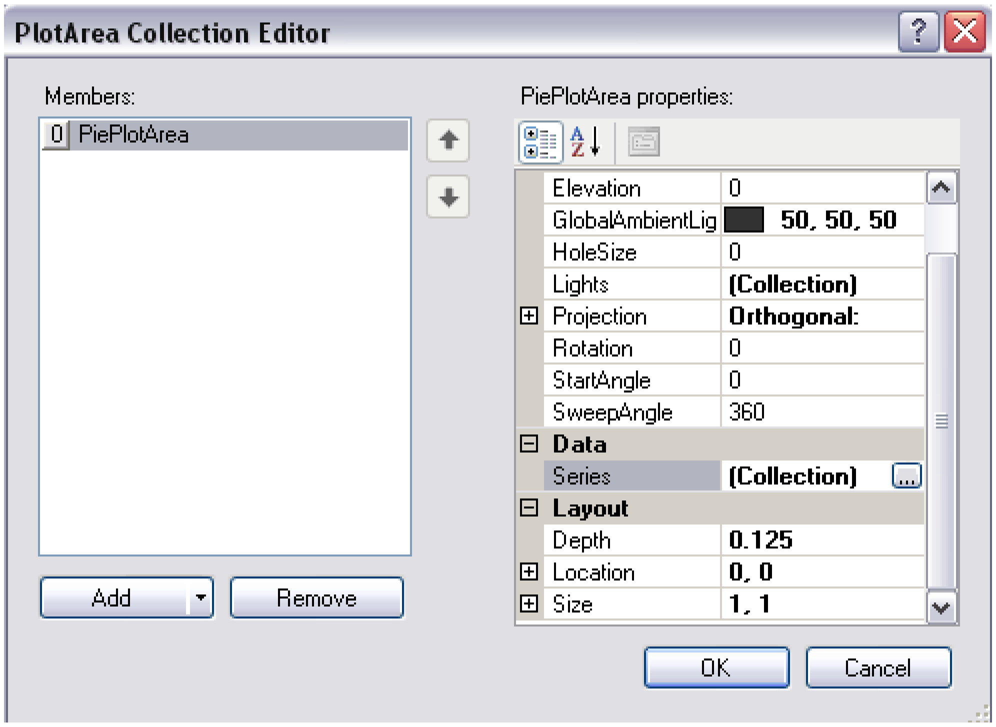
The following steps show how to create a Pie chart using the designer.

1. Open the designer (click on the arrow at the top right edge of the control when it is on a form in Visual Studio or click on the **Designer** menu option to load the chart designer).

2. Click on the **PlotArea Collection** drop-down button. The YPlotArea is already selected. Click **Remove**. Then click **Add** and select the **PiePlotArea**.

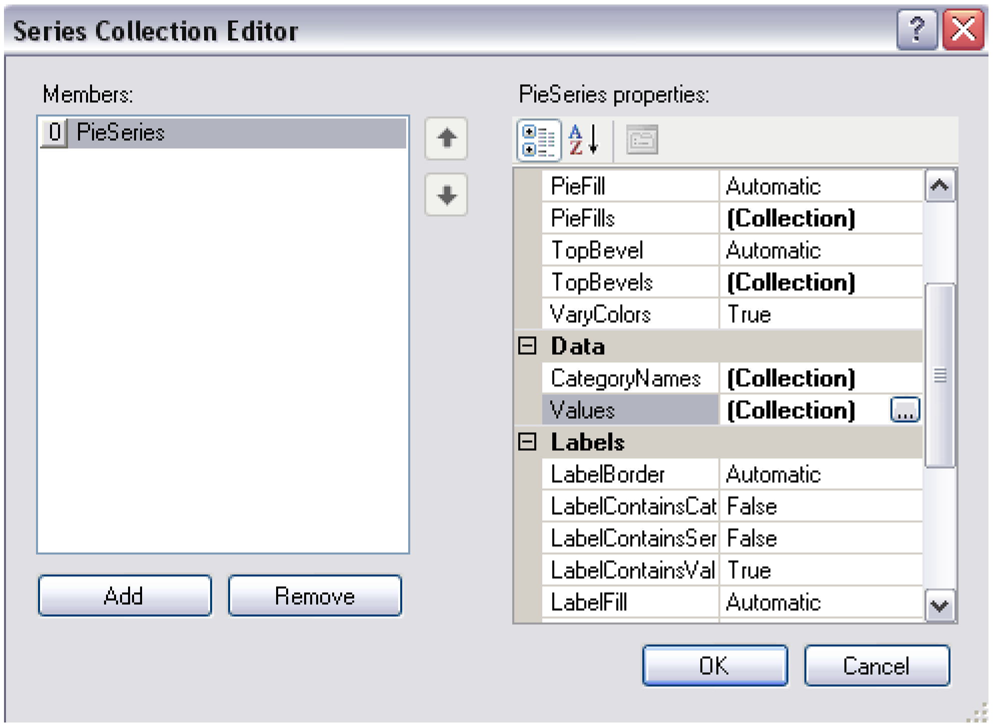


3. Select the **Series Collection** drop-down button.



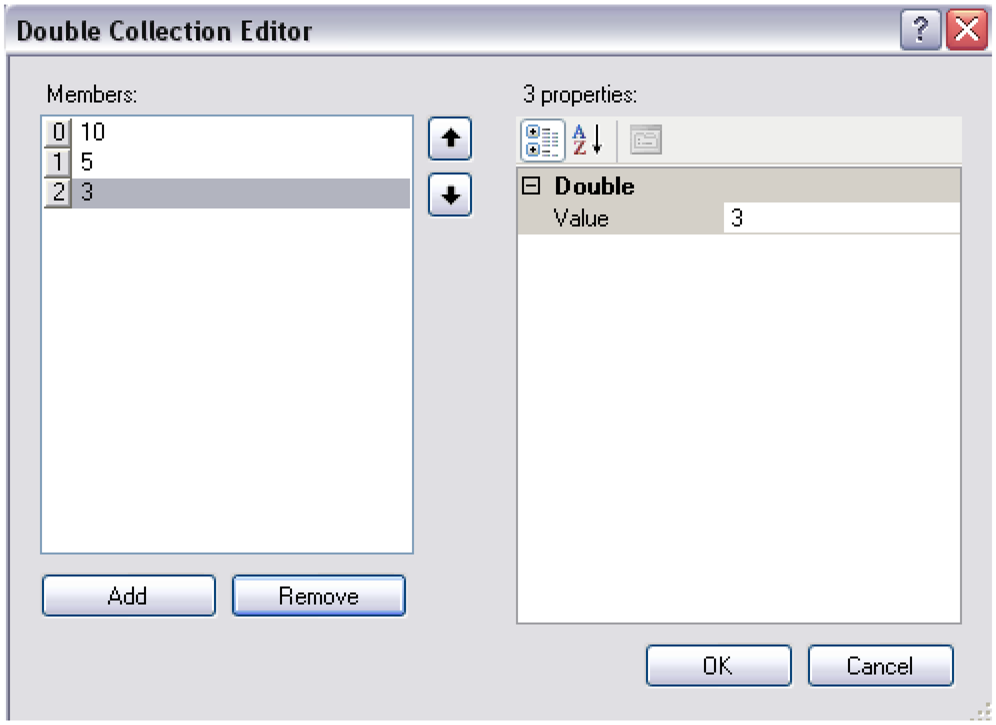
| | |
|---|---|
| Elevation | 0 |
| GlobalAmbientLig | 50, 50, 50 |
| HoleSize | 0 |
| Lights | (Collection) |
| ⊞ Projection | Orthogonal: |
| Rotation | 0 |
| StartAngle | 0 |
| SweepAngle | 360 |
| ⊟ **Data** | |
| Series | (Collection) |
| ⊟ **Layout** | |
| Depth | 0.125 |
| ⊞ Location | 0, 0 |
| ⊞ Size | 1, 1 |

4. Click the **Add** button in the **Series Collection Editor**. Then select the **Values Collection** drop-down button.

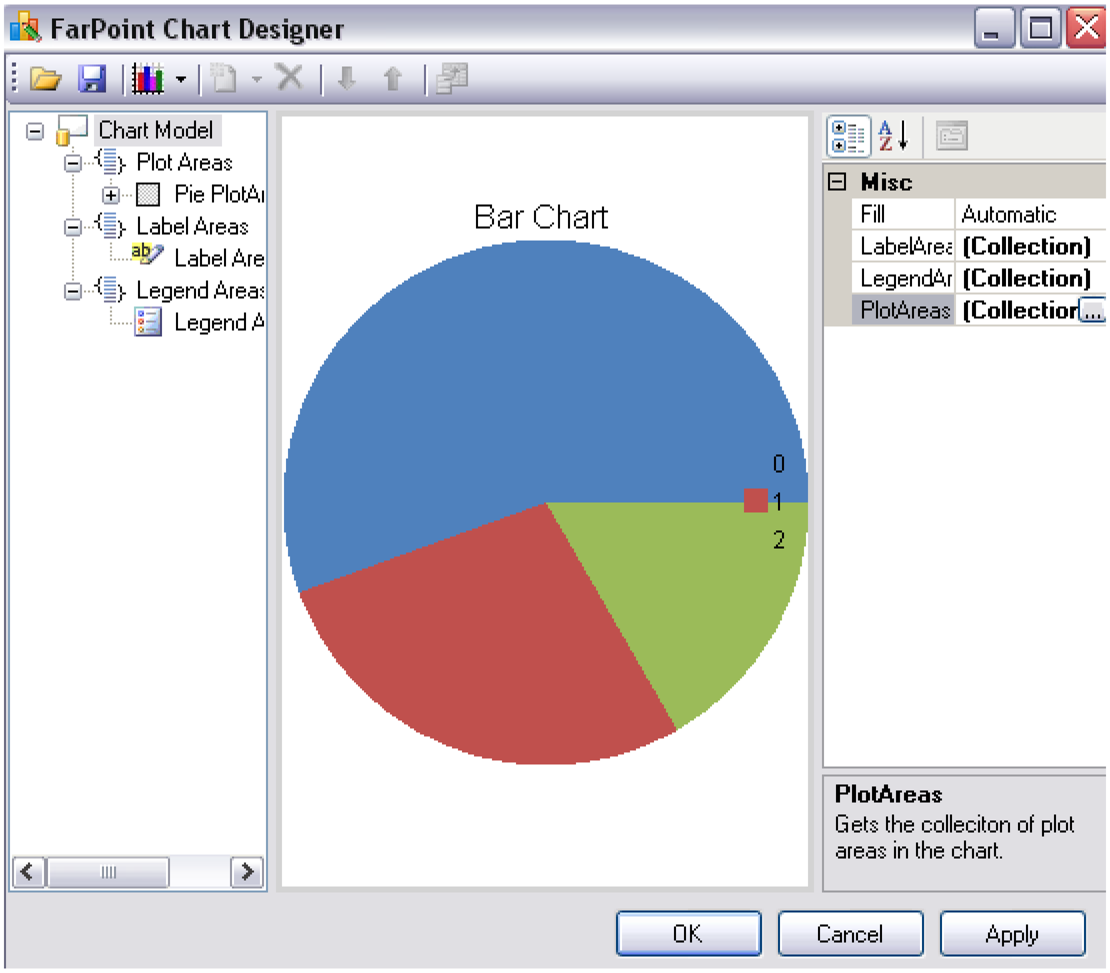


| | |
|---|---|
| PieFill | Automatic |
| PieFills | (Collection) |
| TopBevel | Automatic |
| TopBevels | (Collection) |
| VaryColors | True |
| ⊟ **Data** | |
| CategoryNames | (Collection) |
| Values | (Collection) |
| ⊟ **Labels** | |
| LabelBorder | Automatic |
| LabelContainsCat | False |
| LabelContainsSer | False |
| LabelContainsVal | True |
| LabelFill | Automatic |

5. Click the **Add** button and add multiple data values. Type a double value in the text area on the right side of the editor.



6. Select **OK** on the three dialogs. Click **Apply and OK** to apply the designer changes to the control and close the designer. The **LabelArea Collection** editor can be used to change the text of the legend (from Bar to Pie, for example).



Misc

| | |
|---|---|
| Fill | Automatic |
| LabelArea | (Collection) |
| LegendAr | (Collection) |
| PlotAreas | (Collection) |

**PlotAreas**
Gets the colleciton of plot areas in the chart.

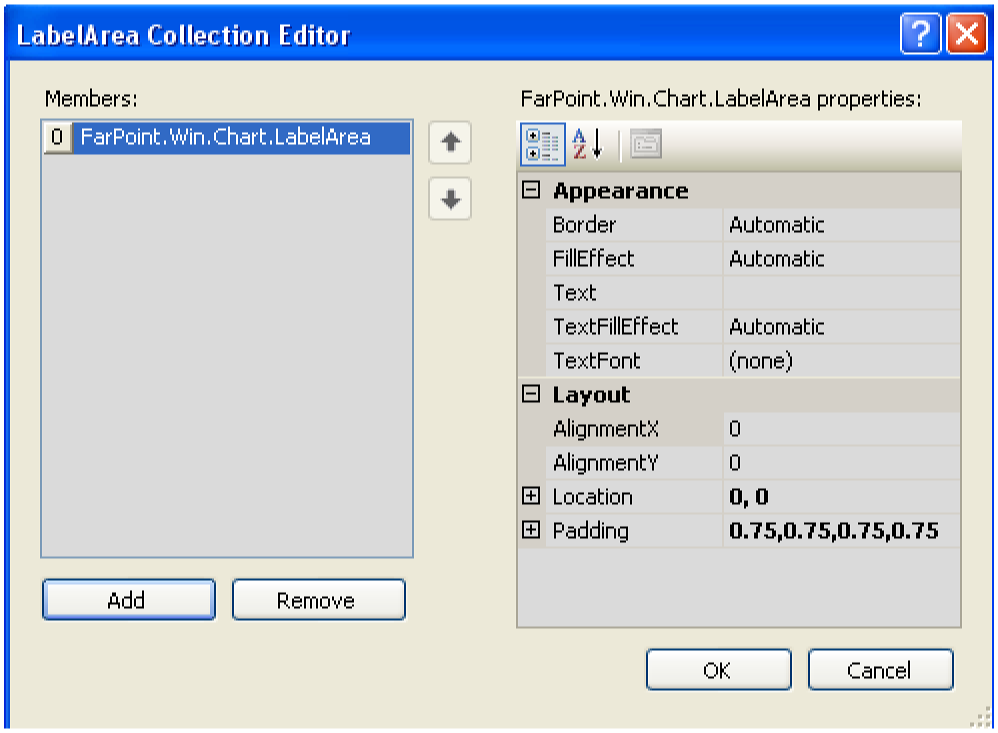
## Chart Collection Editors

There are several editors that can be used to edit areas of the chart control. Open the chart designer and select the appropriate collection drop-down under the **Misc** section.

- **Label Collection Editor**
- **Legend Collection Editor**
- **Plot Collection Editor**
- **Light Collection Editor**
- **Series Collection Editor**

## Label Collection Editor

The **LabelArea Collection Editor** can be used to create labels for the chart and appears as follows:



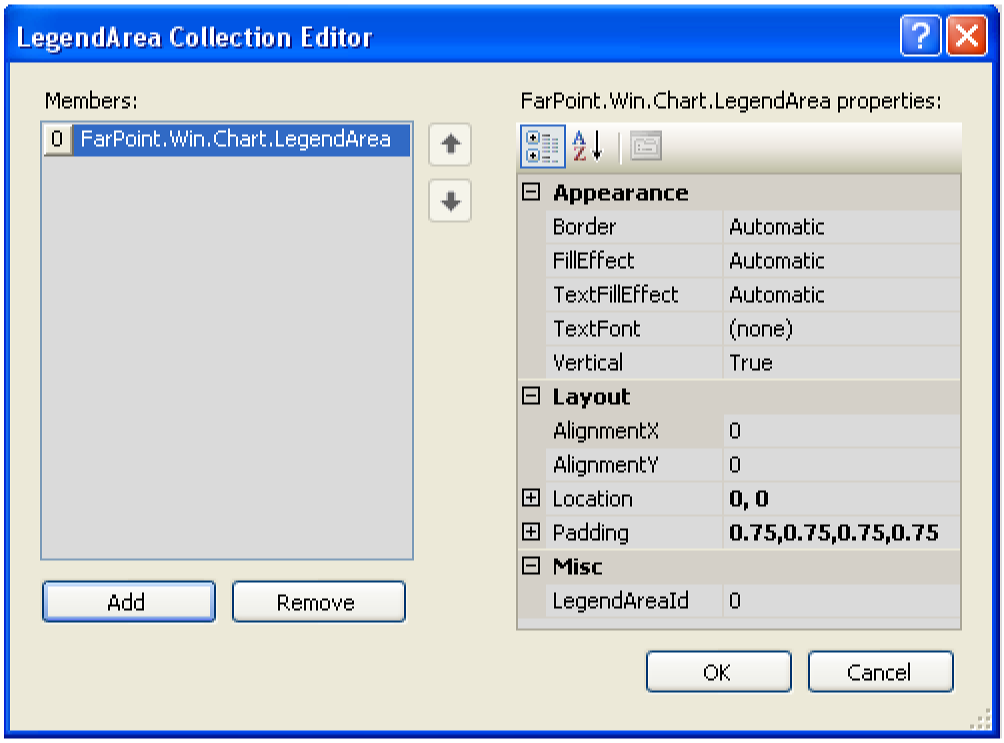
0 FarPoint.Win.Chart.LabelArea

FarPoint.Win.Chart.LabelArea properties:

| | |
|---|---|
| **Appearance** | |
| Border | Automatic |
| FillEffect | Automatic |
| Text | |
| TextFillEffect | Automatic |
| TextFont | (none) |
| **Layout** | |
| AlignmentX | 0 |
| AlignmentY | 0 |
| Location | **0, 0** |
| Padding | **0.75,0.75,0.75,0.75** |

## Legend Collection Editor

The **LegendArea Collection Editor** can be used to create legends for the chart and appears as follows:



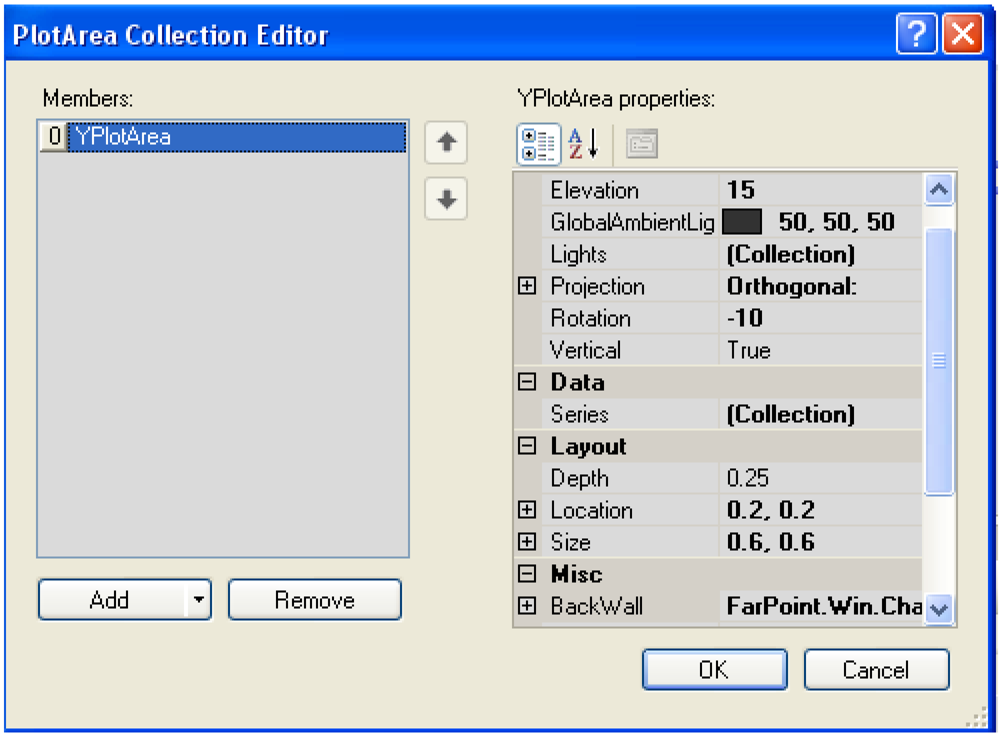
0 FarPoint.Win.Chart.LegendArea

FarPoint.Win.Chart.LegendArea properties:

| | |
|---|---|
| **Appearance** | |
| Border | Automatic |
| FillEffect | Automatic |
| TextFillEffect | Automatic |
| TextFont | (none) |
| Vertical | True |
| **Layout** | |
| AlignmentX | 0 |
| AlignmentY | 0 |
| Location | **0, 0** |
| Padding | **0.75,0.75,0.75,0.75** |
| **Misc** | |
| LegendAreaId | 0 |

## Plot Collection Editor

The **PlotArea Collection Editor** can be used to create plots for the chart and appears as follows:



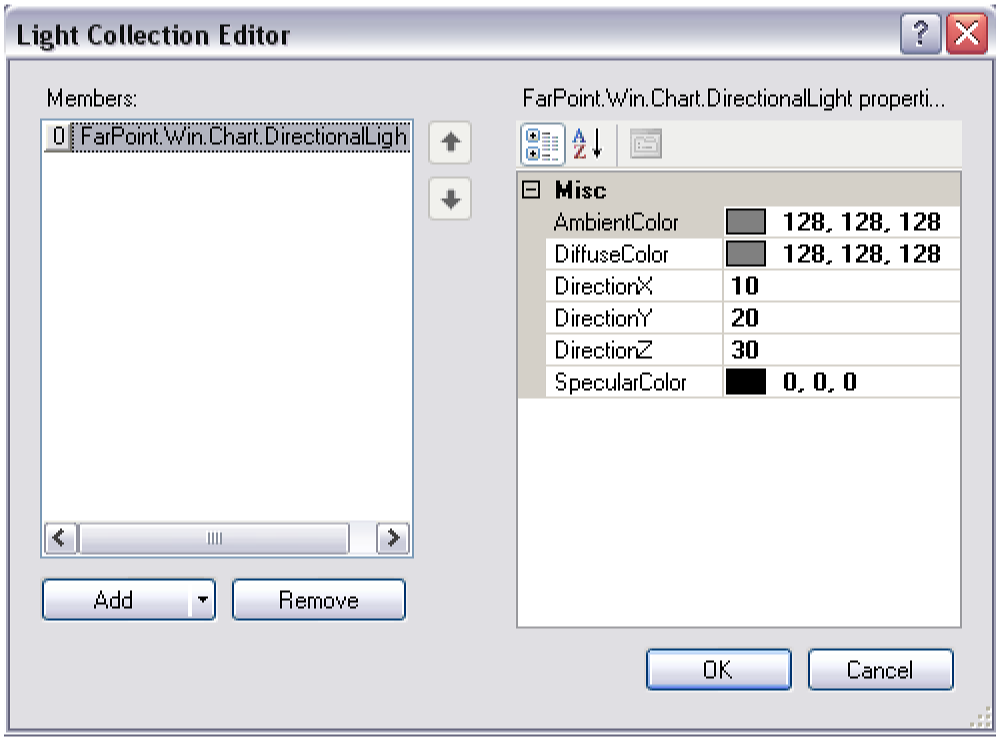
0 YPlotArea

YPlotArea properties:

| | |
|---|---|
| Elevation | 15 |
| GlobalAmbientLig | 50, 50, 50 |
| Lights | (Collection) |
| ⊞ Projection | Orthogonal: |
| Rotation | -10 |
| Vertical | True |
| ⊟ **Data** | |
| Series | (Collection) |
| ⊟ **Layout** | |
| Depth | 0.25 |
| ⊞ Location | 0.2, 0.2 |
| ⊞ Size | 0.6, 0.6 |
| ⊟ **Misc** | |
| ⊞ BackWall | FarPoint.Win.Cha |

## Light Collection Editor

The **Light Collection Editor** can be used to create lighting effects for the chart and appears as follows:



0 FarPoint.Win.Chart.DirectionalLigh

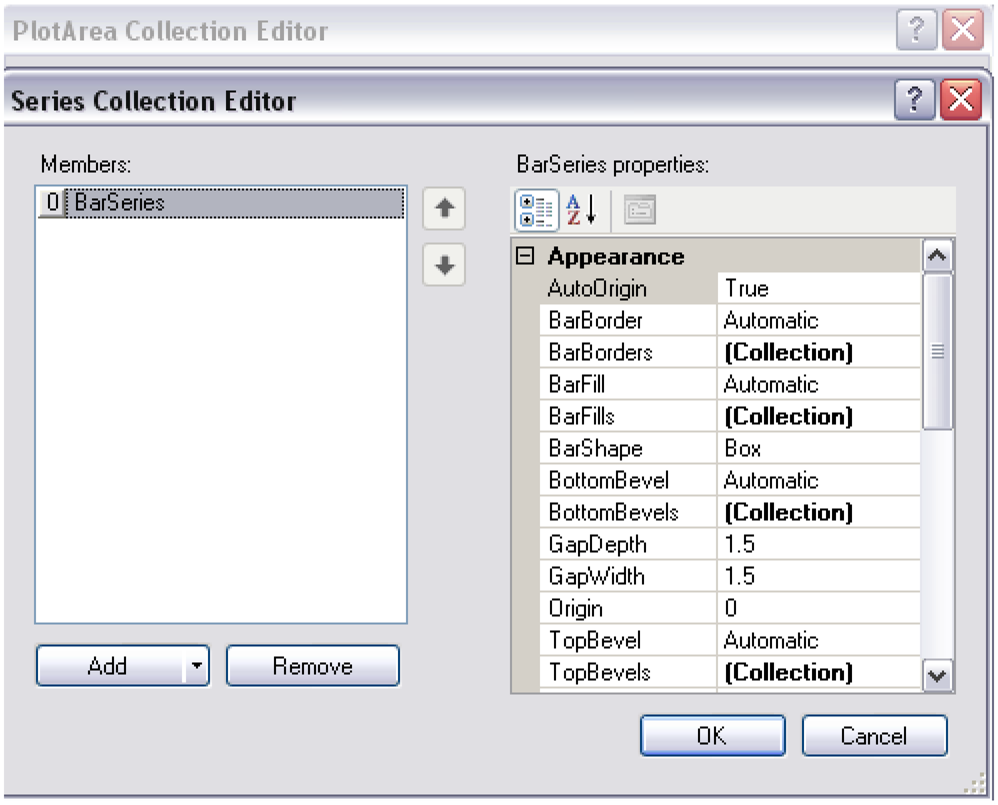
FarPoint.Win.Chart.DirectionalLight properti...

| | |
|---|---|
| ⊟ **Misc** | |
| AmbientColor | 128, 128, 128 |
| DiffuseColor | 128, 128, 128 |
| DirectionX | 10 |
| DirectionY | 20 |
| DirectionZ | 30 |
| SpecularColor | 0, 0, 0 |

The **Light Collection Editor** is under the **Appearance** section after you select the **Plot Areas Collection**. You can also select a plot area from the diagram on the left side of the designer and then select the **Light Collection** under the **Appearance** section.

## Series Collection Editor

The **Series Collection Editor** can be used to set borders, bar shapes, and fill options, add chart data, specify labels and names, and other options. The editor appears as follows:

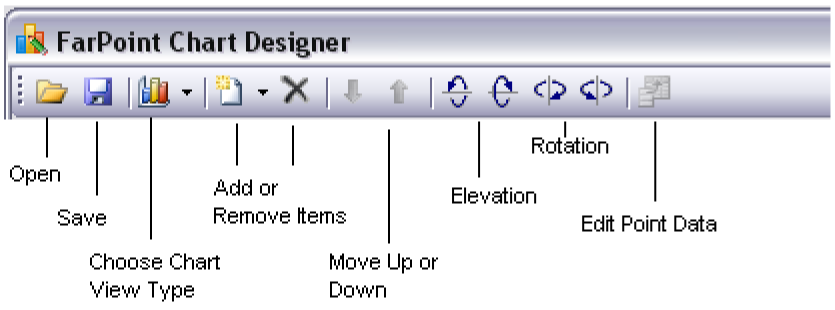


| | |
|---|---|
| **Appearance** | |
| AutoOrigin | True |
| BarBorder | Automatic |
| BarBorders | **(Collection)** |
| BarFill | Automatic |
| BarFills | **(Collection)** |
| BarShape | Box |
| BottomBevel | Automatic |
| BottomBevels | **(Collection)** |
| GapDepth | 1.5 |
| GapWidth | 1.5 |
| Origin | 0 |
| TopBevel | Automatic |
| TopBevels | **(Collection)** |

The **Series Collection Editor** is under the **Data** section after you select the **Plot Areas Collection**. You can also select a plot area from the diagram on the left side of the designer and then select the **Series Collection** under the **Data** section.

## Chart Designer Toolbar

The Chart Designer icons can be used to save the chart to a file, change the chart view type, add or remove items, elevate or rotate the chart, and edit data points.



| Icon | Description |
|---|---|
| Open | This allows you to open the chart control from an XML file. |
| Save | This allows you to open the chart control to an XML file. |
| Chart View Type | This allows you to change the view type to 2D or 3D. |
| Add Items | This allows you to add items to the chart (drop-down list under the Add option). |
| Remove Items | This allows you to move up or down the list if you have created multiple series. |
| Elevation | This allows you to rotate the graph counterclockwise around the horizontal axis. |
| Rotation | This allows you to rotate the graph counterclockwise around the vertical axis. |
| End Point Data | This allows you to bring up the data points in the chart (added with the Double Collection Editor under PlotAreas, Series, and Values). |

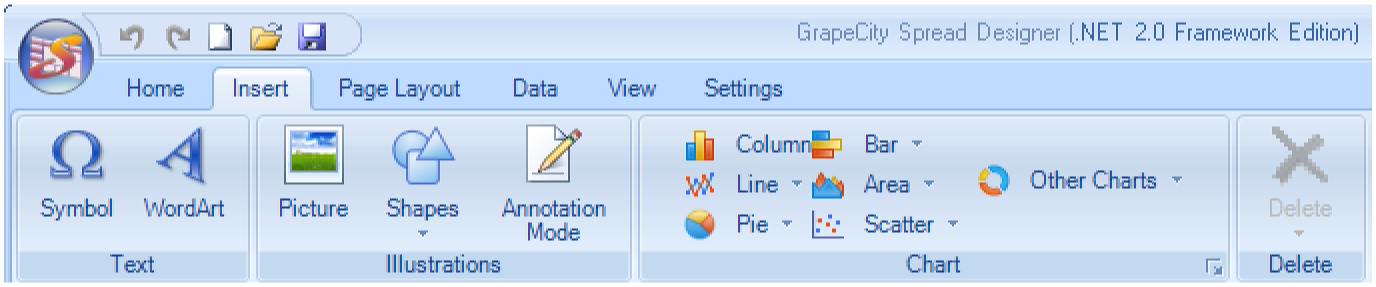
## Using the Chart Control

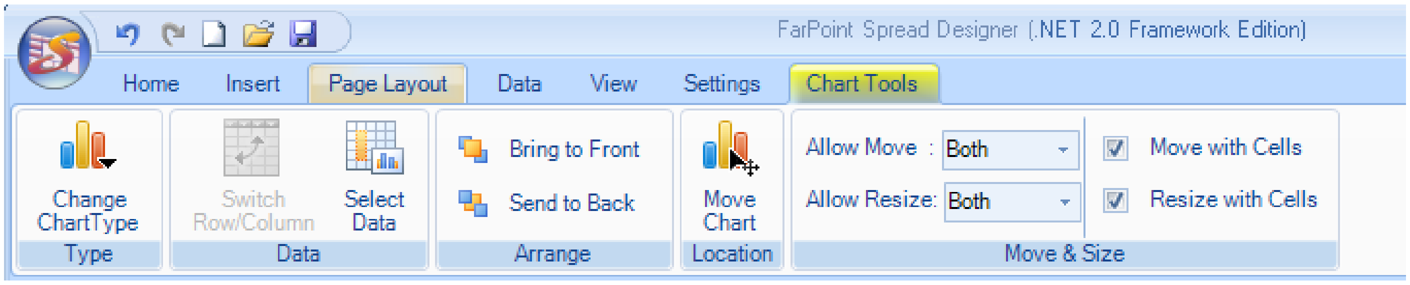
You can add charts using code, the Spread designer, or the Chart designer. You can also bind charts and let the end user make changes to the chart at run time. For more information, see the following topics:

- **Adding a Chart Control**
- **Changing Chart Options**
- **Using the Chart Designer**
- **Binding the Chart Control**
- **Allowing the User to Change the Chart**
- **Adding a Context Menu**

## Adding a Chart Control

You can add a chart control to the sheet using code or the Spread designer. You can also allow the user to resize the chart and the range of data used in the chart control. The following image shows the Chart section in the Spread Designer under the **Insert** menu. The second image shows the **Chart Tools** menu option which is displayed after a chart is added.



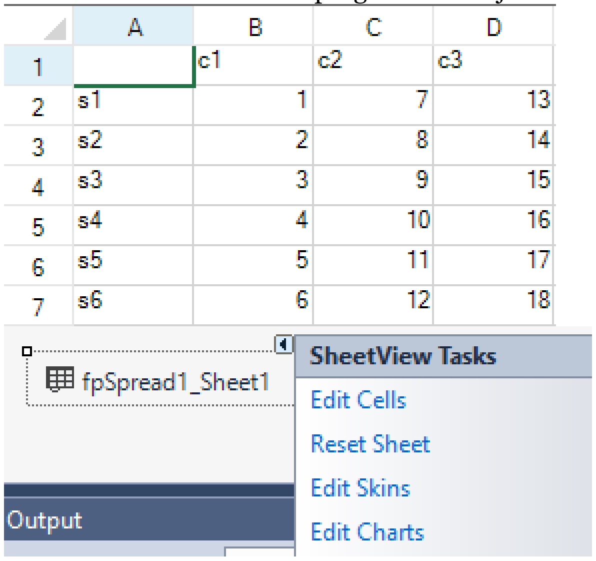


**Using the Spread Designer or Edit Chart Verb**

1. Open the Spread Designer and type chart data in the cells (similar to the data in the code example after this section).
2. Select a range of cells with data.
3. Click on the **Insert** menu option and then pick the chart type (see the above image).
4. The **Chart Tool** menu will appear with additional options.
5. Close the Spread Designer and save the changes.

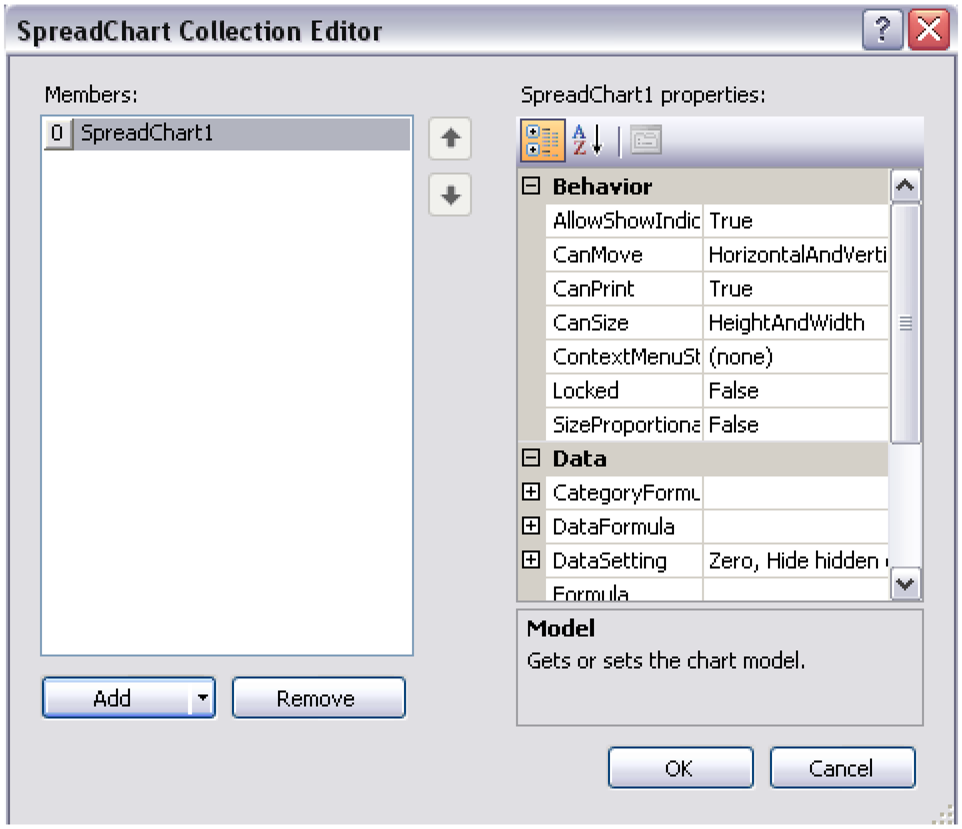
or

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**.



| | A | B | C | D |
|---|---|---|---|---|
| 1 | | c1 | c2 | c3 |
| 2 | s1 | 1 | 7 | 13 |
| 3 | s2 | 2 | 8 | 14 |
| 4 | s3 | 3 | 9 | 15 |
| 5 | s4 | 4 | 10 | 16 |
| 6 | s5 | 5 | 11 | 17 |
| 7 | s6 | 6 | 12 | 18 |

3. This brings up the **SpreadChart Collection Editor**. Click **Add** to add a chart and then set the chart properties.


SpreadChart Collection Editor

Members:

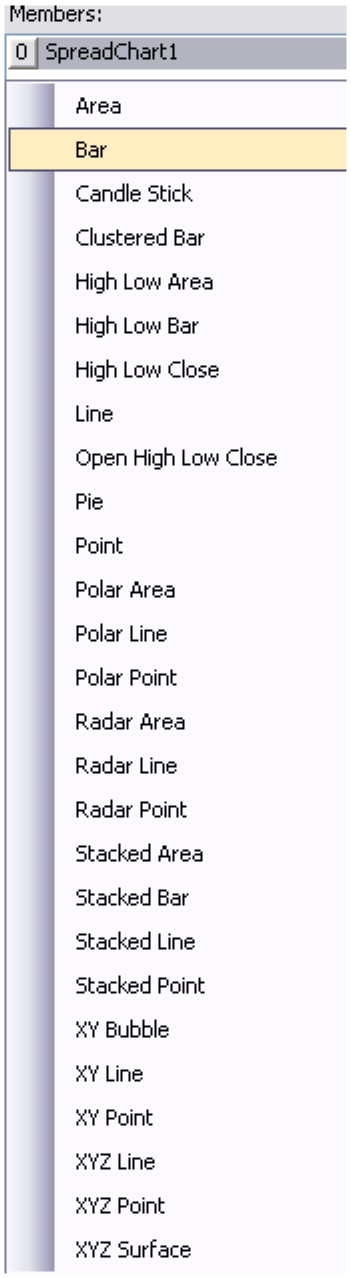
0 SpreadChart1

SpreadChart1 properties:

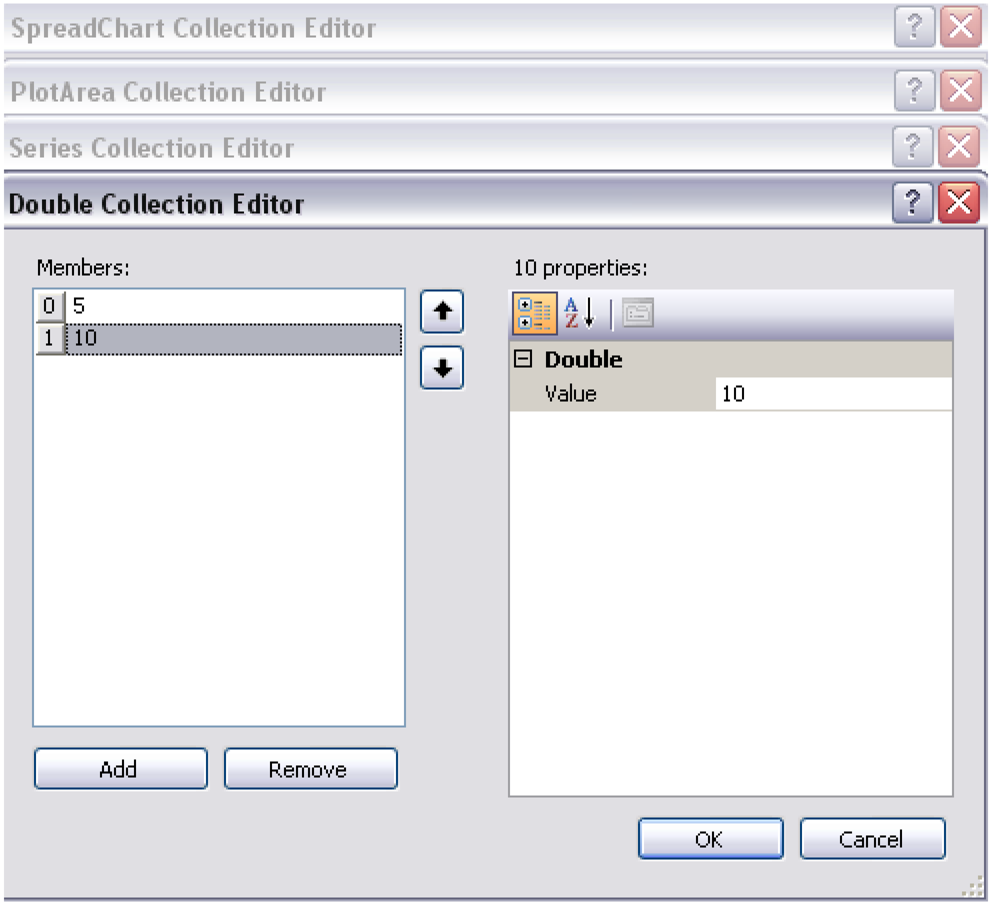
| | |
|---|---|
| **Behavior** | |
| AllowShowIndic | True |
| CanMove | HorizontalAndVerti |
| CanPrint | True |
| CanSize | HeightAndWidth |
| ContextMenuSt | (none) |
| Locked | False |
| SizeProportiona | False |
| **Data** | |
| CategoryFormu | |
| DataFormula | |
| DataSetting | Zero, Hide hidden |
| Formula | |

**Model**
Gets or sets the chart model.

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);
```

**VB**

```
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)
```

**Using Code**

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.

**Example**

This example has code for the **Add** method used to add a chart control to Spread and the **Model** property used for the chart control outside of the Spread control.

**C#**

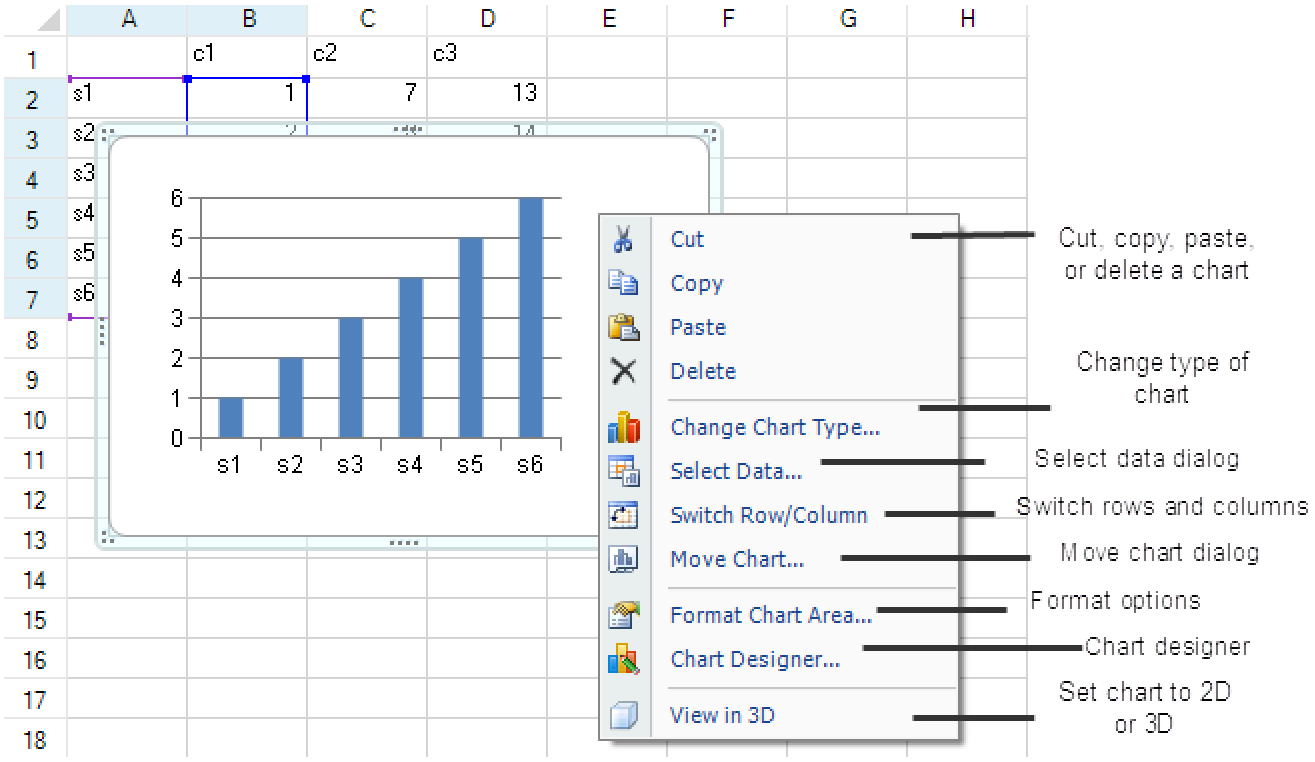
```
FarPoint.Win.Chart.BarSeries series = new FarPoint.Win.Chart.BarSeries();
series.Values.Add(2.0);
series.Values.Add(4.0);
series.Values.Add(3.0);
series.Values.Add(5.0);
FarPoint.Win.Chart.YPlotArea plotArea = new FarPoint.Win.Chart.YPlotArea();
plotArea.Location = new PointF(0.2f, 0.2f);
plotArea.Size = 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);
//How to add the Chart to Spread, requires the chart assembly
//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);
//Or
//How to use the chart outside of Spread, requires a chart control
//fpChart1.Model = model;
```

**VB**

```
Dim series As New FarPoint.Win.Chart.BarSeries()
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)
'How to add the Chart to Spread, requires the chart assembly
'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)
'Or
'How to use the chart outside of Spread, requires a chart control
'fpChart1.Model = model;
```

## Changing Chart Options

Many of the options in the **Chart Tools** tab in the Spread Designer are also available when you right-click on a chart control that has been added to the Spread control. You can access these options at design time or run time.



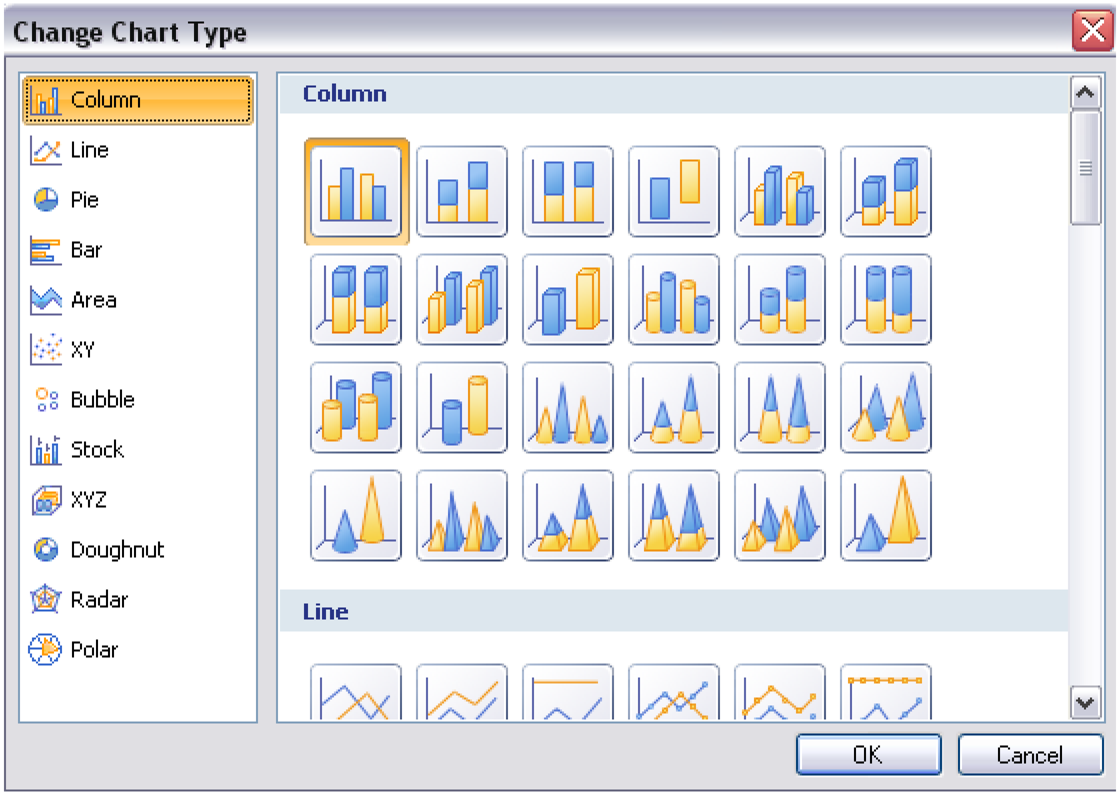
The following options are available:

- The **Cut**, **Copy**, and **Paste** options allow you to cut, copy, and paste the chart control. **Delete** allows you to delete the chart control.
- The **Change Chart Type** option allows you to change the chart type (bar to pie, for example).
- The **Select Data** dialog allows you to change the range of data in the chart as well as edit the series or category names.
- The **Switch Row/Column** option swaps the category and series names and swaps the rows and columns of data.
- The **Move Chart** option brings up a Move Chart dialog that allows you to move the chart to another sheet

view or a new sheet.

- The **Format Chart Area** option allows you to set backcolor, series shapes, etc. The area that you can format depends on what was selected before you right-clicked on the chart.
- The **Chart Designer** option brings up the chart designer.
- The **View** option allows you to set the chart to a 3D or 2D view.

The **Change Chart Type** option appears as follows:



The **Select Data** dialog appears as follows:

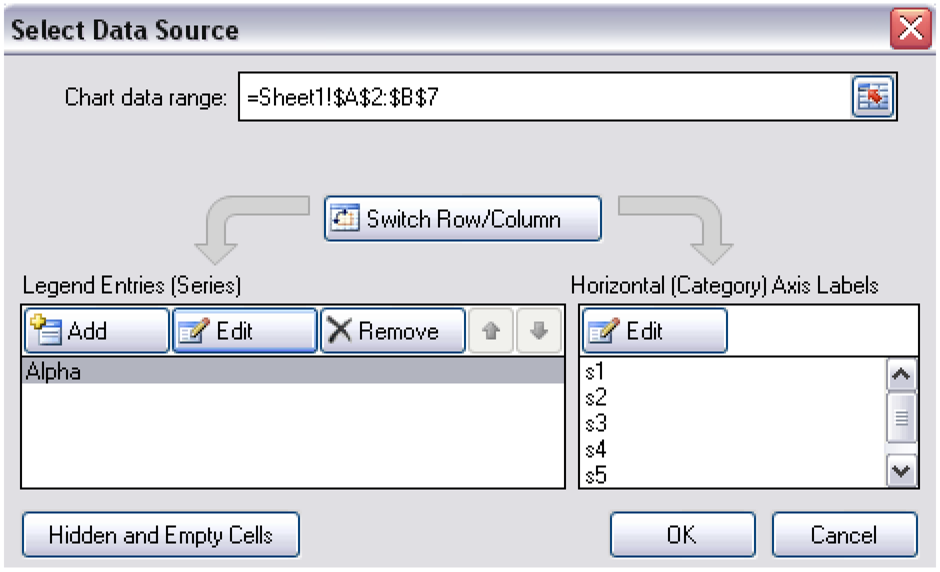


Chart data range: =Sheet1!$A$2:$B$7

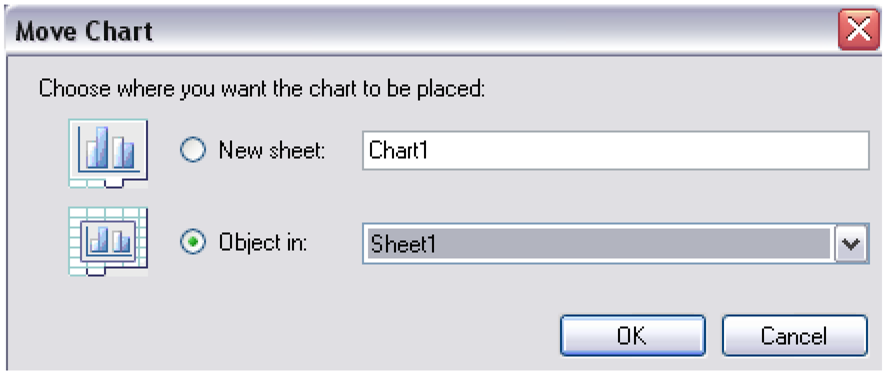
Legend Entries (Series)

Alpha

Horizontal (Category) Axis Labels

s1
s2
s3
s4
s5

The **Move Chart** dialog appears as follows:

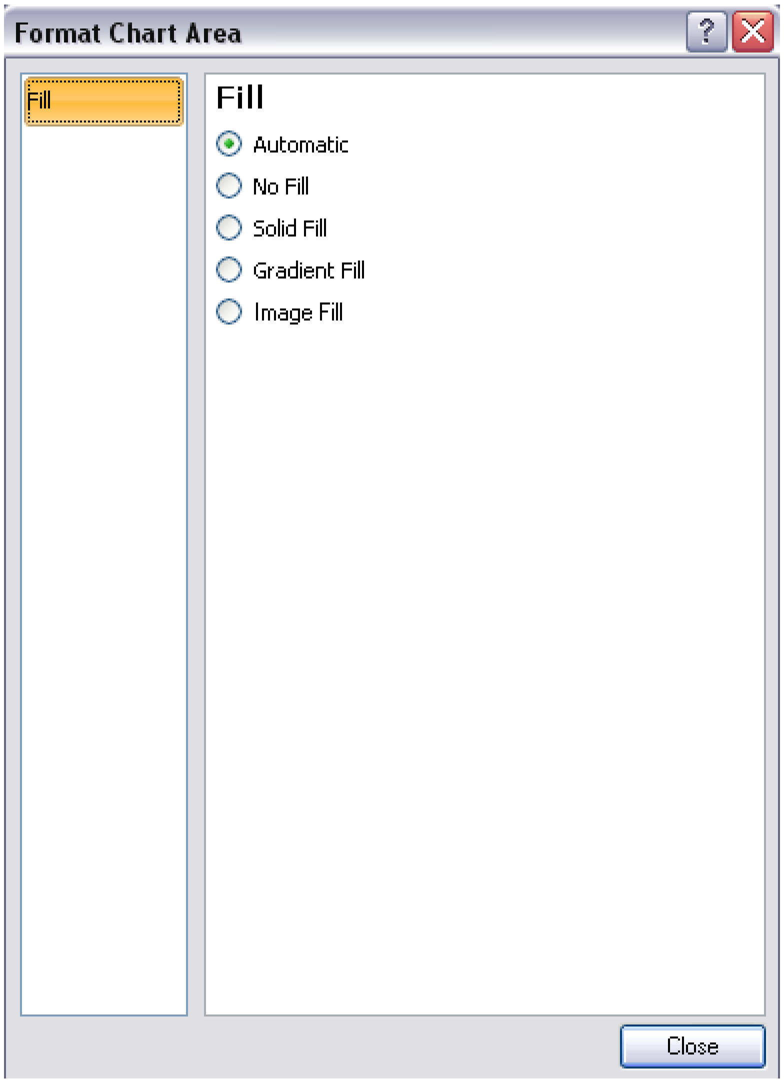

Move Chart

Choose where you want the chart to be placed:

[ ] New sheet: Chart1

[✓ Object in:] Sheet1

The **Format Chart Area** dialog appears as follows:


Format Chart Area

Fill

Fill

[✓ Automatic]

No Fill

Solid Fill

Gradient Fill

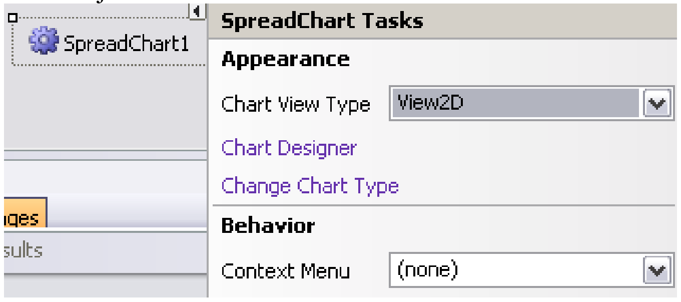
Image Fill

## Using the Chart Designer

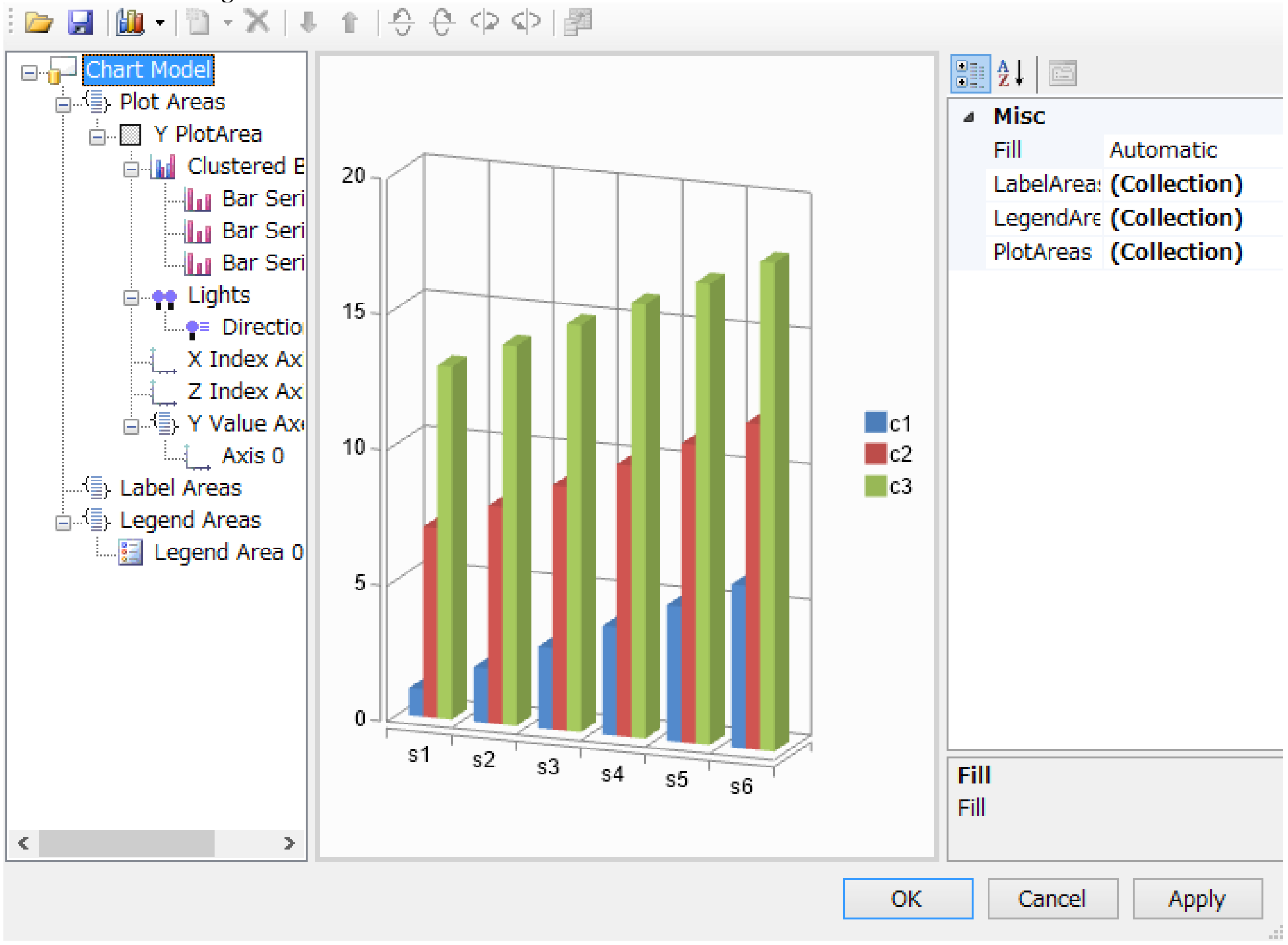
You can add a chart with the chart designer.

**Using the Chart Designer**

1. Use the **Edit Charts** verb to add a chart control (or use the **Insert** menu in the Spread Designer). This creates a Spread Chart object on the form.



2. Click the smart tag or verb of the Spread Chart object.
3. Select the Chart Designer.



4. The Chart Designer allows you to create a chart and set additional options. Use the **PlotAreas Collection** to create plots, series, and add data. Use the **LegendAreas Collection** to create a legend for the chart. Use the **LabelAreas Collection** to create labels for the chart.
5. Click **Apply** to save the changes.

## Binding the Chart Control

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.

**Using Code**

You can add values to Spread with an array or dataset and then use a cell range to put those values in a chart control. This example uses an array to put data in the control.

**Example**

**C#**

```
private void Form1_Load(object sender, System.EventArgs e)
{
object[,] values = { { "lg1", "lg2", "lg3" }, { "tt1", 2.0, 5.0 }, {"tt2",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.BarSeries), 400, 400,
0, 0);
}
private void button1_Click(object sender, EventArgs e)
{
FarPoint.Win.Chart.BarSeries series =
(FarPoint.Win.Chart.BarSeries)fpSpread1.Sheets[0].Charts[0].Model.PlotAreas[0].Series[0];
FarPoint.Win.Spread.Chart.SeriesDataField data =
(FarPoint.Win.Spread.Chart.SeriesDataField)series.Values.DataSource;
data.Formula = "Sheet1!$B$2:$E$1";
}
```

**VB**

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
Dim values As Object(,) = {{"lg1", "lg2", "lg3"}, {"tt1", 2.0R, 5.0R}, {"tt2", 4.0R,
5.0R}}
FpSpread1.Sheets(0).SetArray(0, 0, values)
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.BarSeries), 400,
400, 0, 0)
End Sub
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
Dim series As FarPoint.Win.Chart.BarSeries =
DirectCast(FpSpread1.Sheets(0).Charts(0).Model.PlotAreas(0).Series(0),
FarPoint.Win.Chart.BarSeries)
Dim data As FarPoint.Win.Spread.Chart.SeriesDataField =
DirectCast(series.Values.DataSource, FarPoint.Win.Spread.Chart.SeriesDataField)
data.Formula = "Sheet1!$B$2:$E$1"
End Sub
```

**Using Code**

You can add values to Spread with an array or dataset and then use a cell range to put those values in a chart control. This example uses a data table to put data in the control.

**Example**

**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;
dr = dt.NewRow();
dt.Rows.Add(dr);
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(0,0,2,2);
fpSpread1.Sheets[0].AddChart(cellRange, typeof(FarPoint.Win.Chart.BarSeries),
400,400,0,0);
```

**VB**

```
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, 2, 2)
FpSpread1.Sheets(0).AddChart(cellRange, GetType(FarPoint.Win.Chart.BarSeries), 400,
400, 0, 0)
```

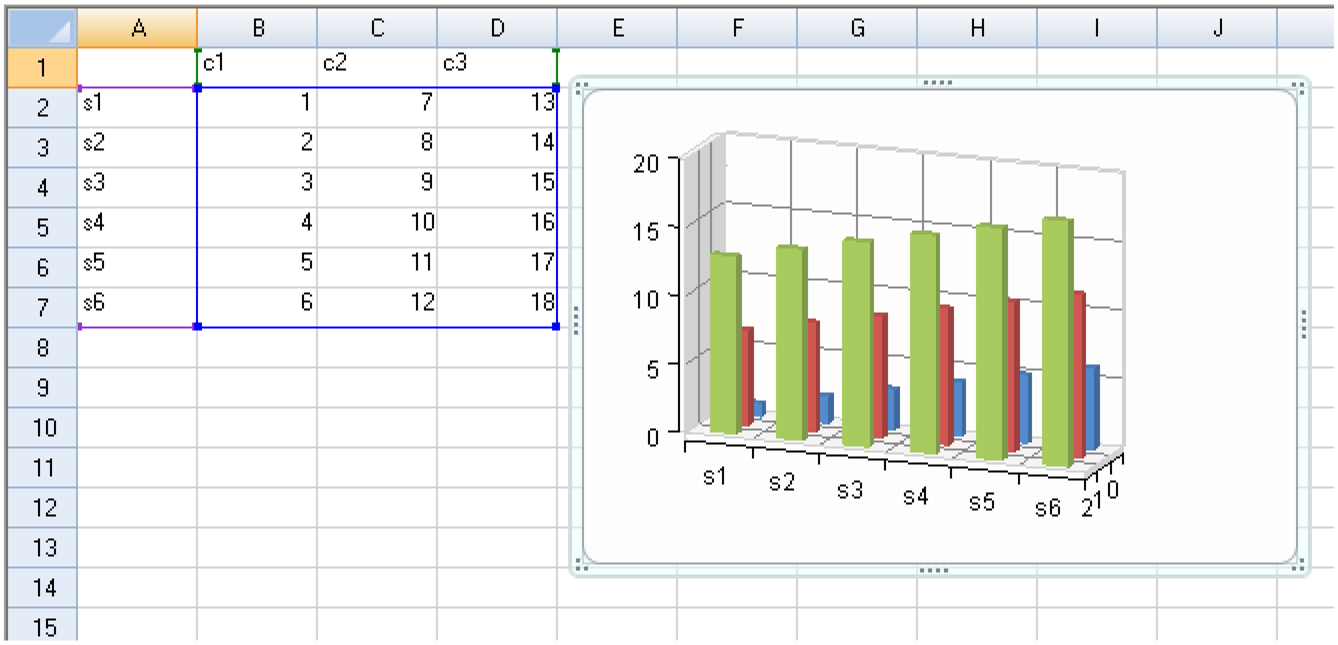
## Allowing the User to Change the Chart

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. Then they 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.



| | A | B | C | D |
|---|---|---|---|---|
| 1 | | c1 | c2 | c3 |
| 2 | s1 | 1 | 7 | 13 |
| 3 | s2 | 2 | 8 | 14 |
| 4 | s3 | 3 | 9 | 15 |
| 5 | s4 | 4 | 10 | 16 |
| 6 | s5 | 5 | 11 | 17 |
| 7 | s6 | 6 | 12 | 18 |

You can prevent the user from moving or resizing 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**

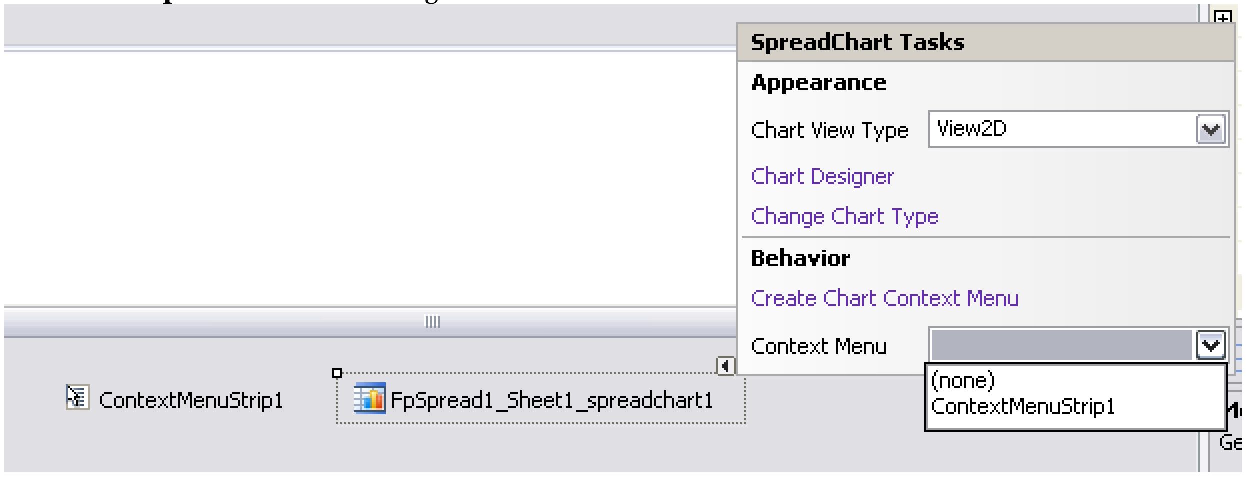
```
Dim chart As FarPoint.Win.Spread.Chart.SpreadChart
Dim range As New FarPoint.Win.Spread.Model.CellRange(0, 0, 7, 4)
chart = FpSpread1.Sheets(0).AddChart(range, GetType(FarPoint.Win.Chart.BarSeries), 400,
300, 300, 80, FarPoint.Win.Chart.ChartViewType.View3D, False)
chart.Locked = True
'chart.CanSize = FarPoint.Win.Spread.DrawingSpace.Sizing.None
'chart.CanMove = FarPoint.Win.Spread.DrawingSpace.Moving.Horizontal
```

## Adding a Context Menu

You can add a context menu to the form for the chart control so that when you right-click on the chart, you see the context menu. You can add the context menu at design time or with code. For more information on using code, see the **ContextMenuStrip ('ContextMenuStrip Property' in the on-line documentation)** property.

**Using the Designer**

1. Add a context menu control to the form.
2. Add a chart control to the form using the Spread Designer.
3. Click on the **Spread Chart** smart tag or verb.



4. Select the context menu you wish to associate with the chart control.

## Using 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 Support**
- **Using Touch Support**

## Understanding Touch Support

Touch support requires that the control support basic touch gestures. Touch messages are processed by the control when touch gestures are used.

The following topics provide additional information:

- **Understanding Touch Gestures**
- **Understanding Touch Messages**

## Understanding Touch Gestures

There are several types of touch gestures such as basic or common and pinch or stretch.

Basic touch gestures include the following:

| Gesture | Description |
|---|---|
| Tap | One finger touches the screen and lifts up. |
| Press and hold | One finger touches the screen and stays in place. |
| Slide | One or more fingers touch the screen and move in the same direction. |
| Swipe | One or more fingers touch the screen and move a short distance in the same direction. |
| Pinch | Two or more fingers touch the screen and move farther apart or closer together. |
| Rotate | Two or more fingers touch the screen and move in a clockwise or counter-clockwise arc. |
| Switch | Two or more fingers touch the screen and move farther apart. |

FpSpread uses standard pinch and stretch gestures when zooming. For more information, see http://msdn.microsoft.com/en-us/library/windows/apps/hh465415.aspx.

## 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 Touch Support

You can use touch support in many areas and in many types of interactions with the Spread control.

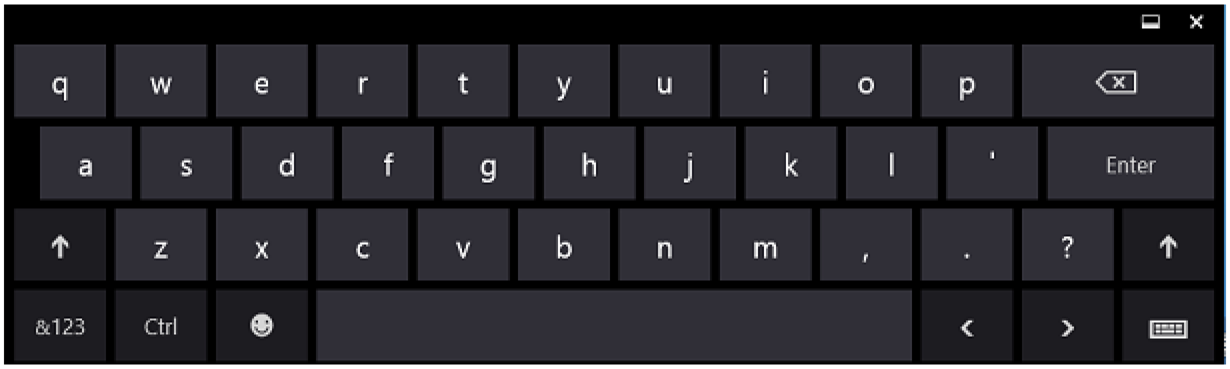
The following topics explain where touch support is available:

- **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 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. This property is supported in Microsoft Windows 8 and Windows Server 2012.

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.

**CS**

```
private void Form1_Load(object sender, EventArgs e)
 {
   fpSpread1.AutoScrollWhenKeyboardShowing = true;
   fpSpread1.InputScope = FarPoint.Win.InputScopeNameValue.Default;
 }

private void fpSpread1_EditModeOn(object sender, EventArgs e)
 {
   fpSpread1.ShowTouchKeyboard();
 }

private void fpSpread1_EditModeOff(object sender, EventArgs e)
 {
   fpSpread1.HideTouchKeyboard();
 }
```

**VB**

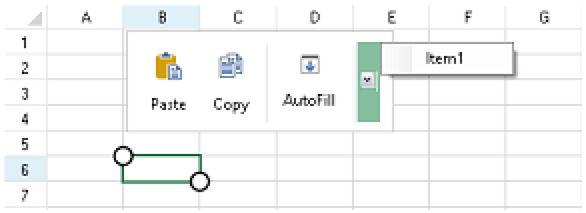
```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    FpSpread1.AutoScrollWhenKeyboardShowing = True
    FpSpread1.InputScope = FarPoint.Win.InputScopeNameValue.Default
End Sub

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. Cancel the default touch strip in the **TouchStripOpening ('TouchStripOpening Event' in the on-line documentation)** event.
2. Create a customized touch strip item.
3. Create and add a menu item
4. Add the new items to the touch strip.

**CS**

```
void autoFill_Click(object sender, EventArgs e)
        {
         FarPoint.Win.Spread.SpreadView activeView =
fpSpread1.GetRootWorkbook().GetActiveWorkbook();
            if (activeView != null)
            {
                activeView.ShowAutoFillIndicator();
            }
        }

        private void fpSpread1_TouchStripOpening(object sender,
FarPoint.Win.Spread.TouchStripOpeningEventArgs e)
        {
            e.Cancel = true;
            FarPoint.Win.Spread.CellTouchStrip touchStrip = new
FarPoint.Win.Spread.CellTouchStrip(this.fpSpread1);
            touchStrip.Items["Cut"].Visible = false;
            ToolStripSeparator separator1 = 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;

            ToolStripSeparator separator2 = new ToolStripSeparator();
            ToolStripDropDownButton dropDownMenu = new
ToolStripDropDownButton(System.Drawing.Image.FromFile("C:\\SpreadWin7\\dropdown.png"));
            dropDownMenu.ShowDropDownArrow = false;
            dropDownMenu.ImageScaling = ToolStripItemImageScaling.None;
            ContextMenuStrip menu = new System.Windows.Forms.ContextMenuStrip();
            menu.Items.Add("Item1");
            dropDownMenu.DropDown = menu;

            touchStrip.Items.AddRange(new ToolStripItem[] { separator1, autoFill,
separator2, dropDownMenu });
            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 fpSpread1_TouchStripOpening(sender As Object, e As
FarPoint.Win.Spread.TouchStripOpeningEventArgs)
e.Cancel = True
Dim touchStrip As New FarPoint.Win.Spread.CellTouchStrip(Me.fpSpread1)
touchStrip.Items("Cut").Visible = False
Dim separator1 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

Dim separator2 As New ToolStripSeparator()
Dim dropDownMenu As New
ToolStripDropDownButton(System.Drawing.Image.FromFile("C:\SpreadWin7\dropdown.png"))
dropDownMenu.ShowDropDownArrow = False
dropDownMenu.ImageScaling = ToolStripItemImageScaling.None
Dim menu As ContextMenuStrip = New System.Windows.Forms.ContextMenuStrip()
menu.Items.Add("Item1")
dropDownMenu.DropDown = menu

touchStrip.Items.AddRange(New ToolStripItem() {separator1, autoFill, separator2,
dropDownMenu})
touchStrip.Show(New Point(e.X - 20, e.Y - 35 - touchStrip.Height))
End Sub
```

## 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

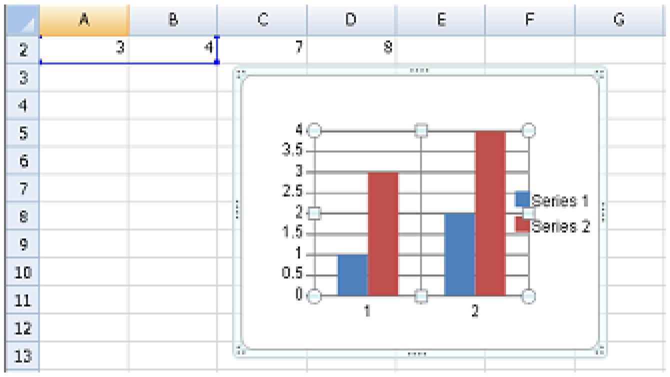
You can use touch gestures with editable cell notes.

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.



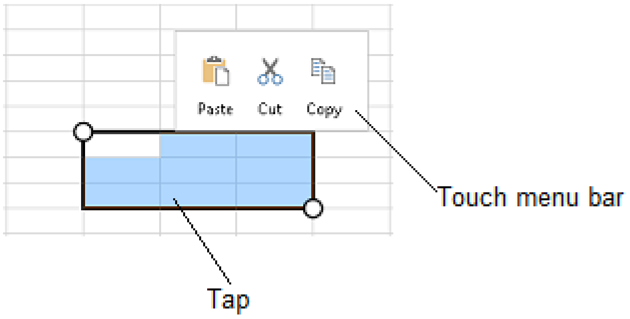
The Chart control uses the following touch gestures:

| 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. |
| Press edge then slide | Press left button on edge then move | Resizes a cell note, shape, or chart if CanSize is set to true. |
| Press chart then slide | Press left button on chart then move | Moves a cell note, shape, or chart if CanMove is set to true. |
| Press rotated handle and slide | Press left button on rotated handle and move | 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.

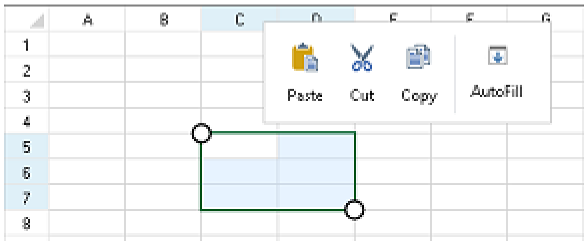


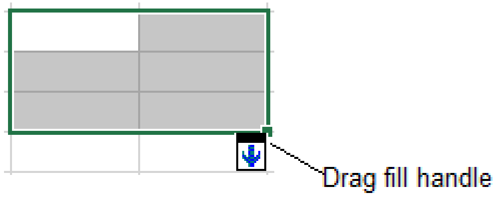
The **ClipboardOptions ('ClipboardOptions Property' in the on-line documentation)** property specifies what areas are part of the cut, copy, or paste.

## 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.





**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
```

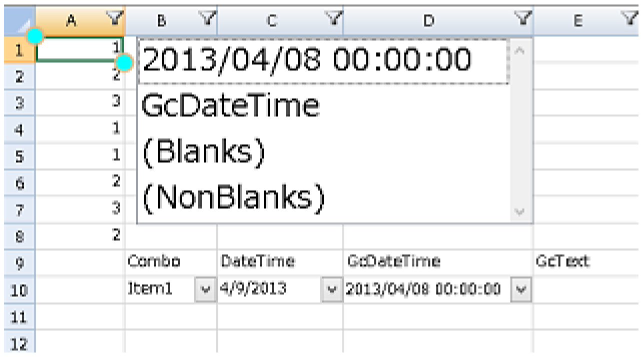
## Using Touch Support with Drop-Down Elements

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.

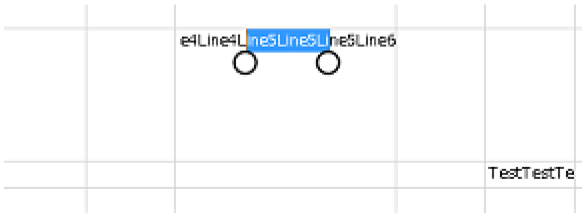
Button, check box, multiple option, hyperlink, slider, and filter bar cells use the following touch gestures:

| Touch Gesture | Mouse Action |
|---|---|
| Tap | Click |
| Double-tap | Double-click |
| Press and slide | 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");
    }
```

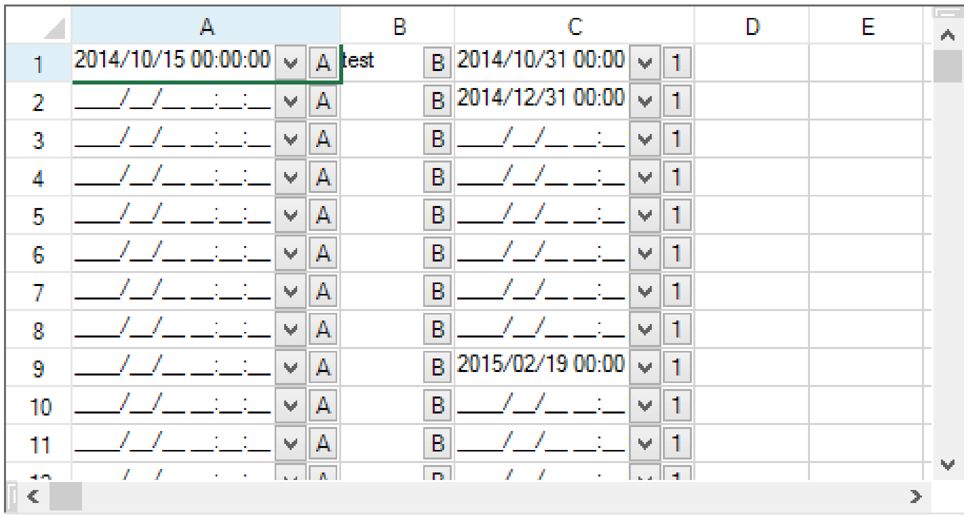
**VB**

```
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.



| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | 2014/10/15 00:00:00 ∨ A | test B | 2014/10/31 00:00 ∨ 1 | | |
| 2 | ____/__/__ __:__:__ ∨ A | B | 2014/12/31 00:00 ∨ 1 | | |
| 3 | ____/__/__ __:__:__ ∨ A | B | ____/__/__ __:__ ∨ 1 | | |
| 4 | ____/__/__ __:__:__ ∨ A | B | ____/__/__ __:__ ∨ 1 | | |
| 5 | ____/__/__ __:__:__ ∨ A | B | ____/__/__ __:__ ∨ 1 | | |
| 6 | ____/__/__ __:__:__ ∨ A | B | ____/__/__ __:__ ∨ 1 | | |
| 7 | ____/__/__ __:__:__ ∨ A | B | ____/__/__ __:__ ∨ 1 | | |
| 8 | ____/__/__ __:__:__ ∨ A | B | ____/__/__ __:__ ∨ 1 | | |
| 9 | ____/__/__ __:__:__ ∨ A | B | 2015/02/19 00:00 ∨ 1 | | |
| 10 | ____/__/__ __:__:__ ∨ A | B | ____/__/__ __:__ ∨ 1 | | |
| 11 | ____/__/__ __:__:__ ∨ A | B | ____/__/__ __:__ ∨ 1 | | |

**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 Grouping

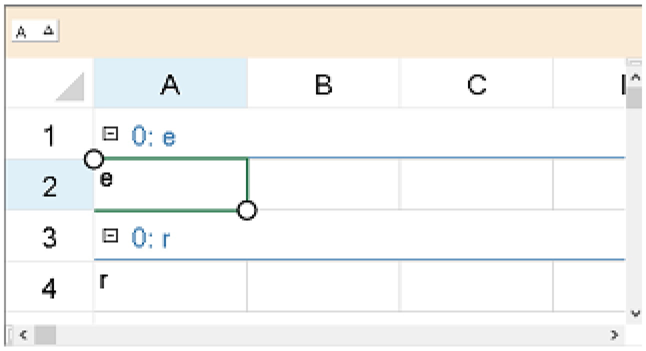
You can use touch gestures when grouping.

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**

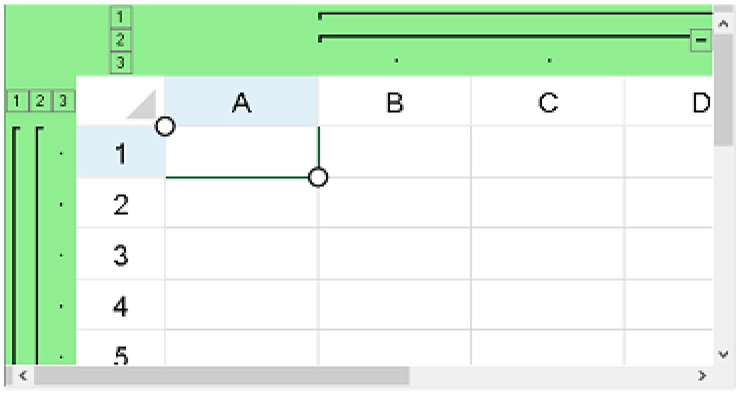
```
fpSpread1.AllowColumnMove = true;
fpSpread1.ActiveSheet.GroupBarInfo.Visible = true;
fpSpread1.ActiveSheet.AllowGroup = true;
fpSpread1.ActiveSheet.ZoomFactor = 2;
```

**VB**

```
FpSpread1.AllowColumnMove = True
FpSpread1.ActiveSheet.GroupBarInfo.Visible = True
FpSpread1.ActiveSheet.AllowGroup = True
FpSpread1.ActiveSheet.ZoomFactor = 2
```

## Using Touch Support 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**

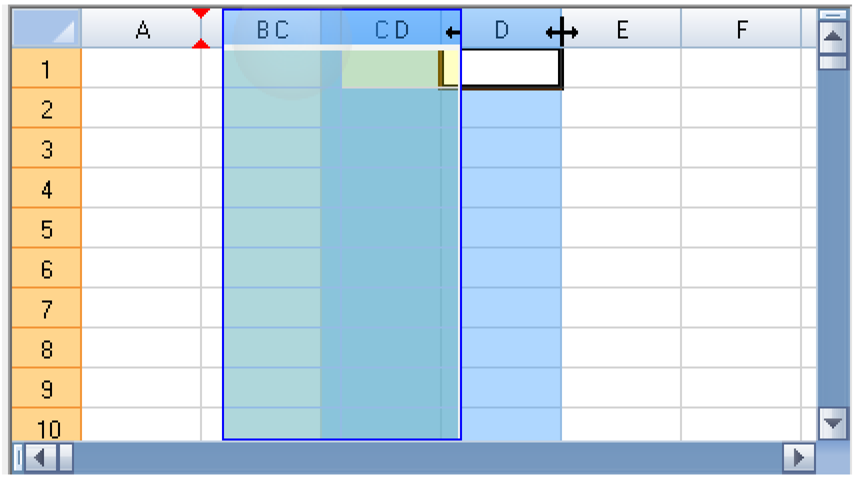
```
FpSpread1.ActiveSheet.Rows.Count = 11
FpSpread1.ActiveSheet.Columns.Count = 6
FpSpread1.InterfaceRenderer = Nothing
FpSpread1.ActiveSheet.RangeGroupBackgroundColor = Color.LightGreen
FpSpread1.ActiveSheet.RangeGroupButtonStyle =
FarPoint.Win.Spread.RangeGroupButtonStyle.Enhanced
FpSpread1.ActiveSheet.AddRangeGroup(0, 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 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
```

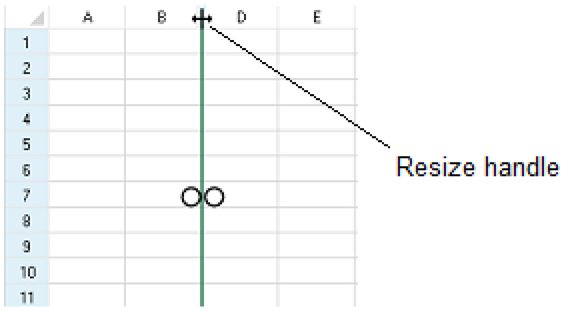
## Using Touch Support when Resizing Columns or Rows

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;
```

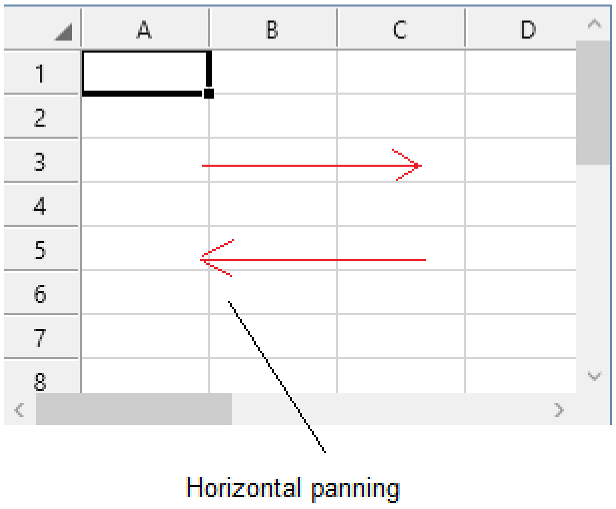
**VB**

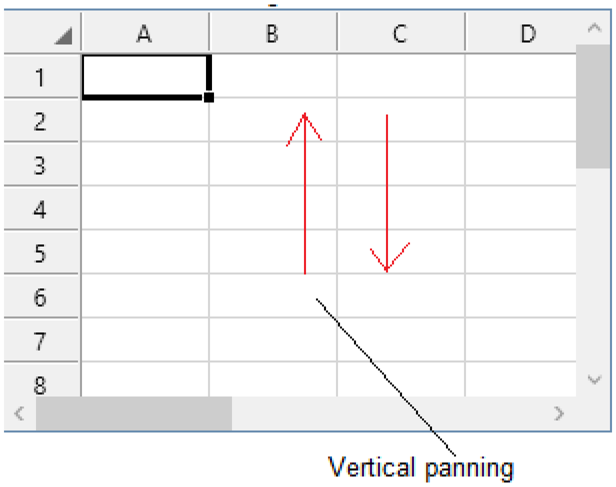
```
FpSpread1.ResizeZeroIndicator = FarPoint.Win.Spread.ResizeZeroIndicator.Enhanced
FpSpread1.ActiveSheet.Columns(0, 5).Resizable = True
FpSpread1.ActiveSheet.Rows(0, 10).Resizable = True
FpSpread1.ActiveSheet.Columns(2).Width = 0
```

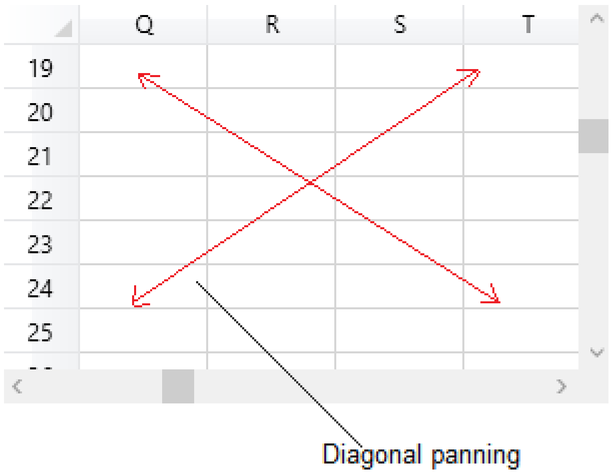
## Using Touch Support with Scrolling

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.







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 http://msdn.microsoft.com/en-us/library/windows/desktop/dd317331(v=vs.85).aspx. 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.

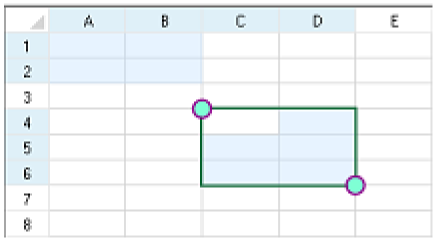
Tap a cell to select the cell and display the selection gripper. Press the cell selection gripper and slide. Release to select a cell range.

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.



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.

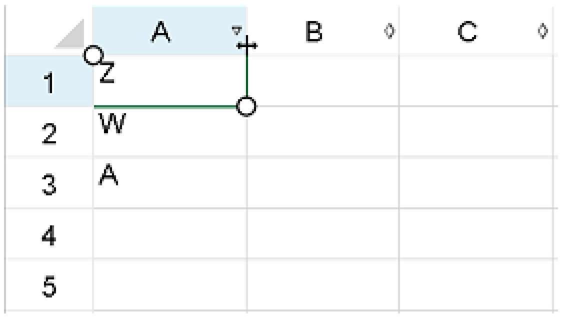
Use the following touch gestures with shapes:

| 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.



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

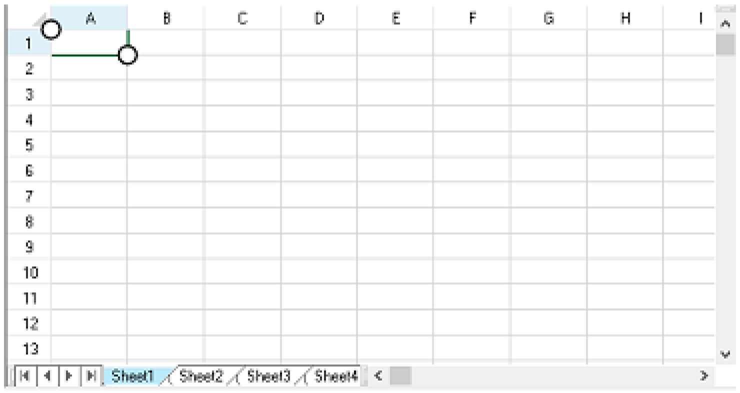
You can use touch gestures with viewports.

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.

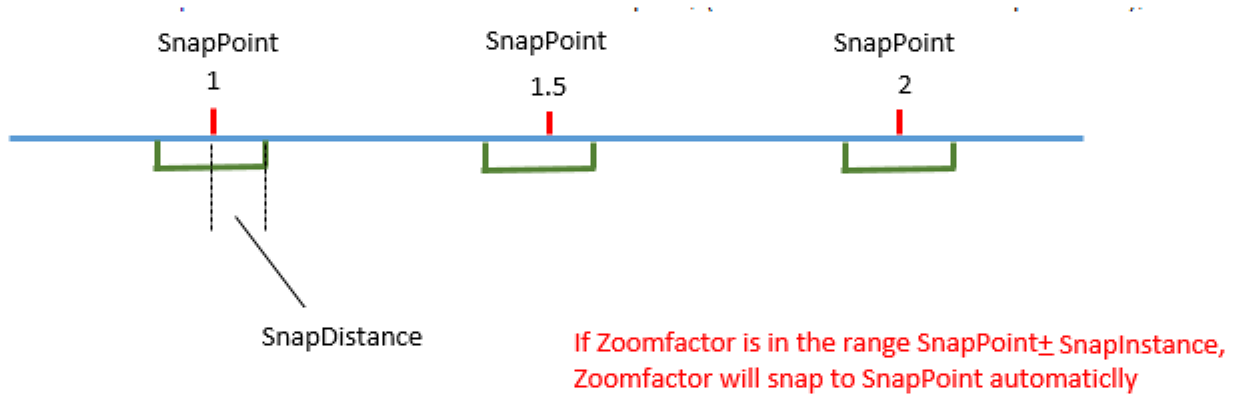
Set the **ZoomFactor ('ZoomFactor Property' in the on-line documentation)** property to change the size of the tab strip buttons. Some touch gestures are easier to use if the control is zoomed.

## 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%.


SnapPoint
1
SnapPoint
1.5
SnapPoint
2
SnapDistance
If Zoomfactor is in the range SnapPoint± SnapInstance,
Zoomfactor will snap to SnapPoint automaticlly

FpSpread uses standard pinch and stretch gestures when zooming. For more information, see [http://msdn.microsoft.com/en-us/library/windows/apps/hh465415.aspx](http://msdn.microsoft.com/en-us/library/windows/apps/hh465415.aspx).

**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)
```

# 2 Index