Developer's Guide

This guide provides introductory conceptual material and how-to explanations for routine tasks for developers using Spread for ASP.NET. It describes how an application developer would use the properties and methods in Spread to create spreadsheets and grids on Web Forms, bind to databases, and customize the component for your application.

- **Getting Started**
- **Understanding the Product**
- **Working with the Spread Designer**
- **Customizing the Appearance**
- **Customizing User Interaction**
- **Customizing with Cell Types**
- **Managing Data Binding**
- **Managing Data in the Component**
- **Managing Formulas**
- **Managing File Operations**
- **Using Sheet Models**
- **Maintaining State**
- **Working with the Chart Control**
- **Using Touch Support with the Component**

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

For a complete list of documentation, refer to the **Spread for ASP.NET Documentation (on-line documentation)**.

1 Table of Contents

Getting Started

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

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

Handling Installation

Here are the tasks for installing the product for development and for redistribution.

- **Installing the Product**
- **Licensing a Trial Project after Installation**
- **End-User License Agreement**
- **Creating a Build License**
- **Handling Redistribution**
- **Product Requirements**
- **Handling Variations in Windows Regional Settings**

Installing the Product

Installation instructions and a list of installed files for Spread for ASP.NET 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.NET 11** -> **ASP.NET** -> **SpreadASPReadMe**. 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 \Program Files\GrapeCity\Spread.NET 11\Docs\ASP.NET and 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 ASP.NET 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 Read Me.
2. Open the project in Microsoft Visual Studio.
3. Open the Visual Studio **Build** menu and select **Rebuild Solution**.
4. The web application is now licensed and no evaluation banners appear when you run it. You can distribute the Web application to unlicensed machines and no evaluation banners appear.

For licensing Web Site applications, open the Visual Studio **Build** menu and select **Build Runtime Licenses** to create the App_Licenses.dll file.

End-User License Agreement

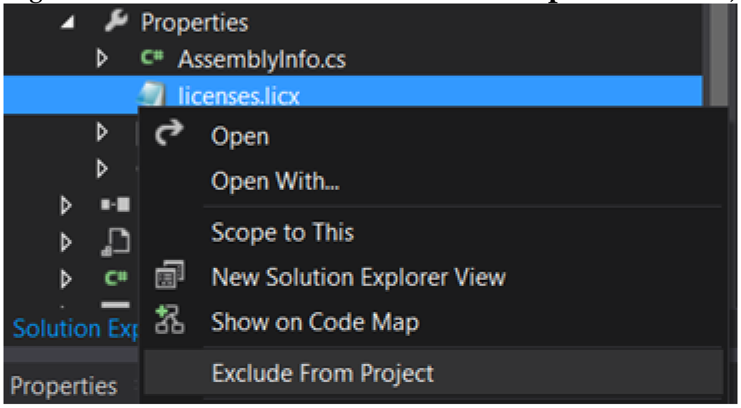
The GrapeCity licensing information, including the GrapeCity end-user license agreements, frequently asked licensing questions, and the GrapeCity licensing model, is available online at https://www.grapecity.com/en/licensing/spread and https://www.grapecity.com/en/legal/eula.

Creating a Build License

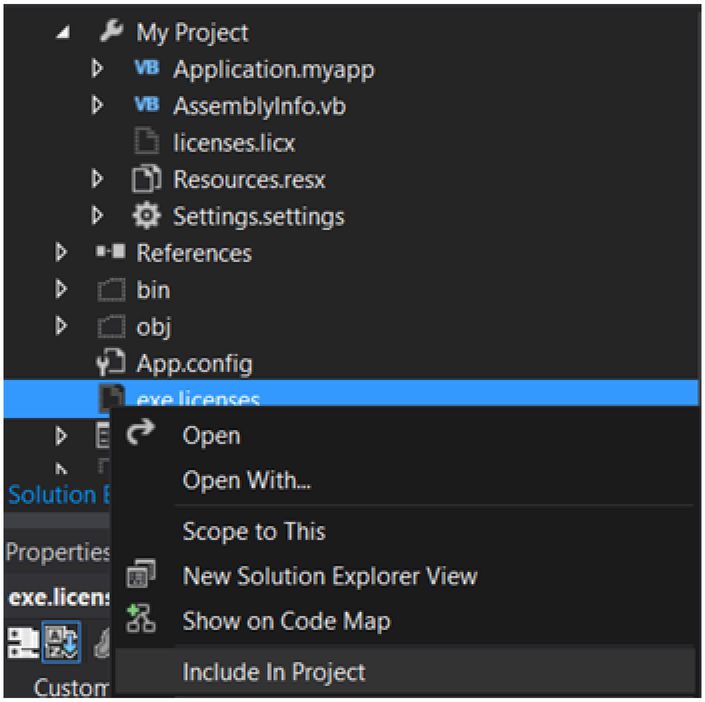
You can create a build license to use on a build machine.

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:

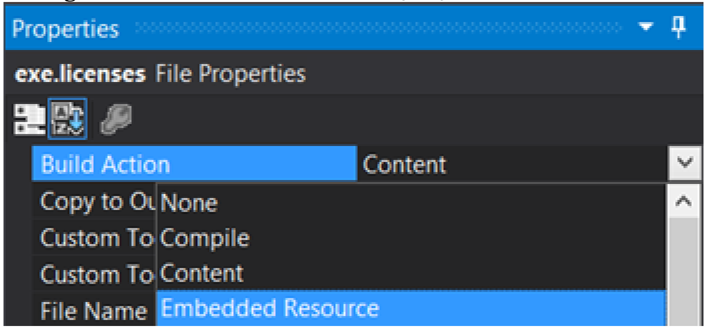
1. Build the project using the licensed components on a developer machine which is licensed for development with all the components referenced in the project (this creates the .licenses resource).
2. Find the licenses.licx in the Solution Explorer window. You can use the **Show All Files** toolbar button 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

Please review this information concerning redistribution of Spread for ASP.NET with your application.

Server Requirements

You must deploy to a Microsoft Internet Information server.

Server Files

Place the assemblies that come with Spread for ASP.NET in either your server's global assembly cache (GAC) or in your application directory's \bin folder under the **wwwroot** directory on your server.

Place the following assemblies on your server:

- FarPoint.Web.Spread.dll
- FarPoint.CalcEngine.dll
- FarPoint.Excel.dll
- FarPoint.PDF.dll
- FarPoint.Web.Chart.dll (if you use the Chart control or Sparklines)
- FarPoint.Web.Spread.Extender.dll (if you use the extender classes)
- System.Web.Extensions.dll (if you use the FarPoint.Web.Spread.Extender.dll)
- AjaxControlToolkit.dll (if you use the FarPoint.Web.Spread.Extender.dll)
- FarPoint.Mvc.Web.Spread.dll (if you use Spread in an MVC3 project)

Place the **fp_client folder** (installed in Spread.NET\ASP.NET\..\fp_client) and its subfolders provided with Spread for ASP.NET under your server's **wwwroot** directory, or, if you wish to put it elsewhere, set up a virtual directory in IIS Manager to point to the location of that folder's contents.

The fp_client folder can also be placed in the web application directory. The following code would need to be added to the web config file. For example:

```
<?xml version="1.0"?>
<configuration>
<system.web>
    ...
</system.web>
<appSettings>
<add key="fp_client" value="fp_client" />
</appSettings>
</configuration>
```

Be aware that Spread for ASP.NET creates a Web server control that serves up HTML pages for clients and it also puts HTC files in a directory on the client machine for client-side scripting capability.

Spread for ASP.NET uses jQuery 2.x. If the web page or web application uses jQuery 2.0 or higher, Spread uses that version of jQuery. If the web page uses jQuery 1.9 or earlier, Spread uses jQuery 2.x internally and does not conflict with the web page version of jQuery.

Permission Requirements

If you use the Spread control on medium trust web sites, you need to add SerializationFormatter and Reflection permissions to the machine config file, web_mediumtrust.config. The SecurityPermission needs the UnmanagedCode and SerializationFormatter flags. For example:

```
<IPermission class="SecurityPermission" version="1" Flags="Assertion, Execution, ControlThread, ControlPrincipal, RemotingConfiguration, UnmanagedCode, SerializationFormatter"/>
<IPermission class="ReflectionPermission" version="1" Unrestricted="true" Flags="ReflectionPermissionFlag.MemberAccess"/>
```

Product Requirements

For developing applications with the .NET 4.0 version of Spread for ASP.NET, you must have the following system items:

Operating System

One of the following:

- Microsoft Windows 2003 Server
- Microsoft Windows 2008 Server
- Microsoft Windows 2012 Server
- Microsoft Windows XP Professional
- Microsoft Windows Vista
- Microsoft Windows 7
- Microsoft Windows 8
- Microsoft Windows 8.1
- Microsoft Windows 10

Software

- Release version of the Microsoft .NET 4.0 Framework.
- Microsoft Internet Information Services (IIS)
- SQL Server or the SQL Server desktop engine that ships with Visual Studio .NET installed on your machine to be able to run some of the data binding samples
- The Spread extender requires the AJAX Control Toolkit
- The Spread Designer requires Microsoft Internet Explorer (IE) 7 or higher and the Microsoft.mshtml.dll.

Handling Variations in Windows Regional Settings

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.

Working with the Component

Here are the tasks involved with starting to work with the component.

- **Adding a Component to a Web Site using Visual Studio 2015 or 2017**
- **Adding a Component to a Web Site using Visual Studio 2013**
- **Adding a Component to a Web Site using Visual Studio 2012**
- **Adding a Component to a Web Site using Visual Studio 2010**
- **Adding and Using JavaScript IntelliSense**
- **Understanding Browser Support**
- **Understanding Parts of the Component Interface**
- **Working with Collection Editors**
- **Working with Web Parts**
- **Working with Windows Azure**
- **Working with Microsoft ASP.NET MVC 5**
- **Working with Microsoft ASP.NET MVC 3**
- **Copying Shared Assemblies to Local Folder**
- **Working with Strongly Typed Data Controls**

Adding a Component to a Web Site using Visual Studio 2015 or 2017

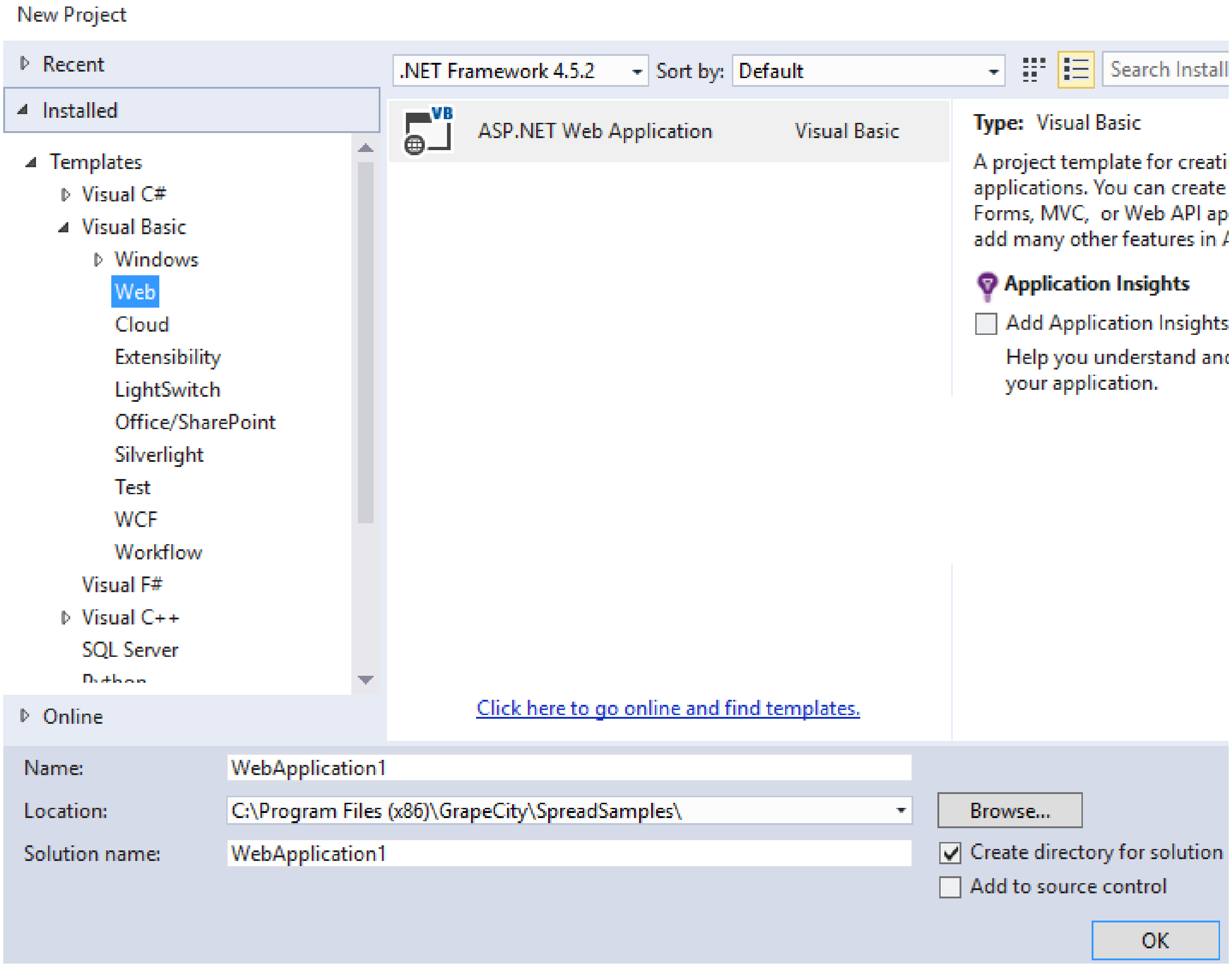
Use the following steps to add a Spread component to a Web Form in Visual Studio. You can either open an existing Web Site or create a new one.

> Spread, as a child control of the page, is affected by the style settings on that page (similar to placing a table on a web form, and setting a master CSS for everything inside the page). If you create a default web application with Visual Studio 2010 or higher, the default master page contains CSS style settings. Spread, once placed on this default page, can be affected by the style settings and the layout may change. Avoid the following HTML tags to prevent the layout change: TD, TH, TABLE, INPUT, and TEXTAREA.

Step 1. Start Visual Studio.

Step 2. Create a new Web site.

1. Select **New Project** or from the **File** menu, choose **New**, **Web Site**.
2. Under **Templates**, select **Web** under **Visual Basic** or **Visual C#.**



.NET Framework 4.5.2

Sort by: Default

Type: Visual Basic

A project template for creati
applications. You can create
Forms, MVC, or Web API ap
add many other features in A

□ Add Application Insights

Help you understand and
your application.

Name: WebApplication1

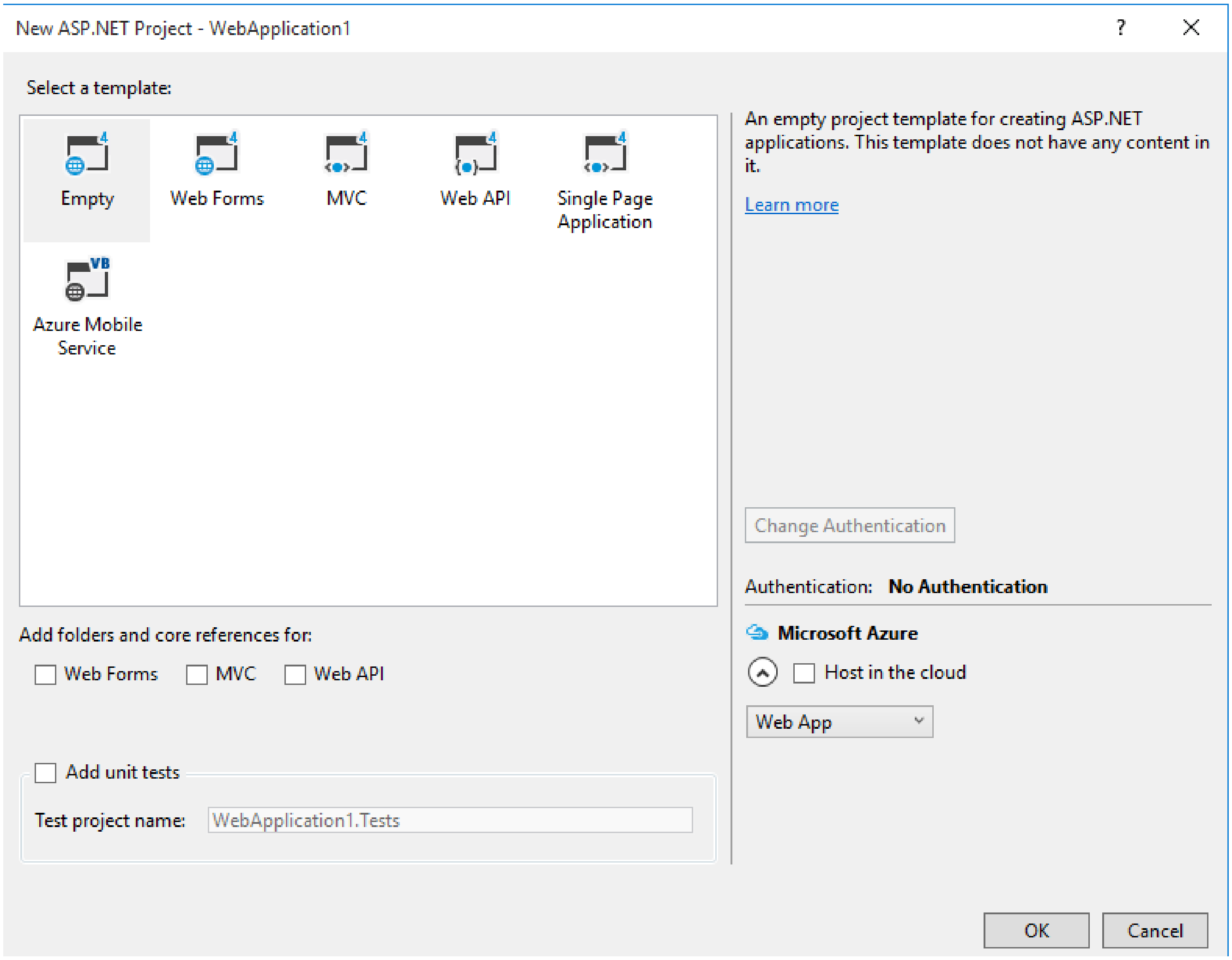
Location: C:\Program Files (x86)\GrapeCity\SpreadSamples\

Solution name: WebApplication1

☑ Create directory for solution

□ Add to source control

3. Select **ASP.NET Web Application**.
4. Specify a location and name for the project.
5. Select **OK**.
6. Select a template such as **Empty**.



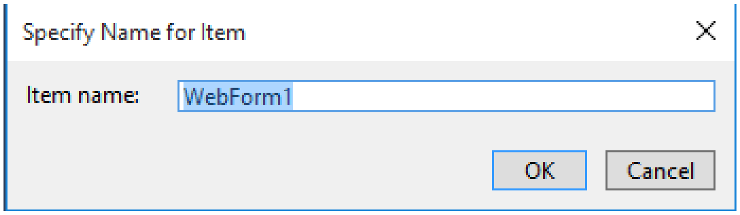
An empty project template for creating ASP.NET applications. This template does not have any content in it.

Authentication: **No Authentication**

Test project name: WebApplication1.Tests

7. Select **OK**.
   If your project does not display the **Solution Explorer**, from the **View** menu, choose **Solution Explorer**. If you used an empty site, you may wish to add a web form to the project (choose **Add**, **Web Form** after right-clicking on the project name in the **Solution Explorer**).

   Specify the **Item name**. Select **OK**.



Item name: WebForm1

Step 3. Add the FpSpread component to the toolbox if the component is not displayed in the toolbox.

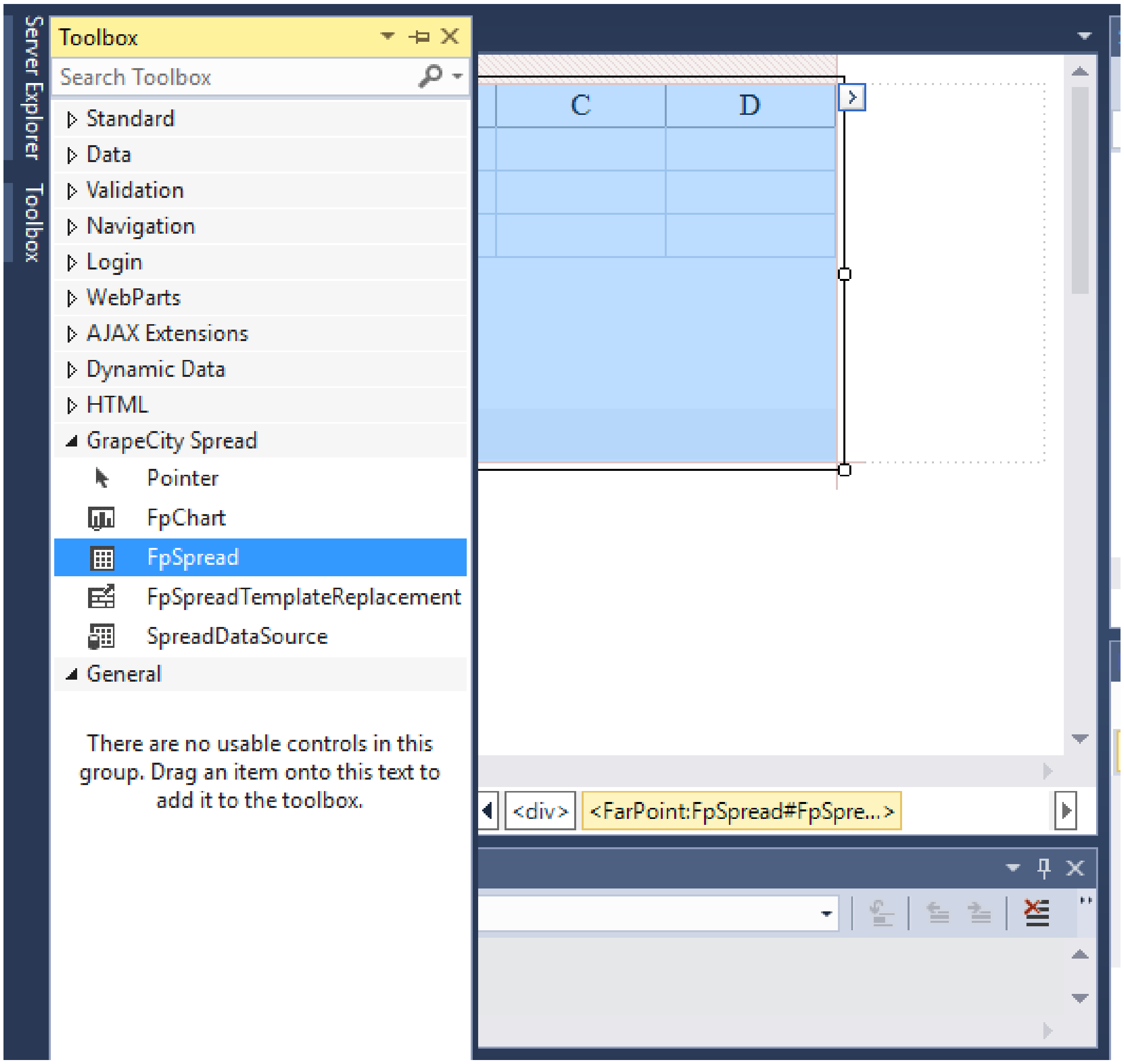
1. If the **Toolbox** is not displayed, from the **View** menu choose **Toolbox**.
2. Once the **Toolbox** is displayed, look in the **GrapeCity Spread** category (or in other categories if you have installed Spread and placed the toolbox icon in a different category).
3. If the FpSpread component is not in the **Toolbox**, right-click in the **Toolbox**, and from the pop-up menu choose **Choose Items**.
4. In the **Choose Toolbox Items** dialog, click the **.NET Framework Components** tab.
5. In the **.NET Framework Components** tab, the FpSpread component should be displayed in the list of components.

Select the FpSpread component check box and click **OK**.
If the FpSpread component is not displayed in the list of components, click **Browse** and browse to the installation path for the Spread component. Once there, select FarPoint.Web.Spread.dll and click **Open**. The FpSpread component is now displayed in the list of components. Select it and click **OK**. Select FarPoint.Web.Chart.dll if you wish to add FpChart at design time.

6. You can test that the component has been added by opening a project and inserting the component.

Step 4. Add the FpSpread component to the Web site.

1. With an open project, in the **Toolbox** under **Web Forms**, select the FpSpread component. Select FpChart if you wish to add the chart at design time.
2. On your Web Forms page, draw an FpSpread component by dragging a rectangle the size that you would like the initial component or simply double-click on the page.
3. The FpSpread component appears (as shown in this Visual Studio project).



Adding a Component to a Web Site using Visual Studio 2013

Adding an FpSpread component to a Web Form in Visual Studio 2013 involves the following steps of adding the component to a Web Site. You can either open an existing Web Site or create a new one.

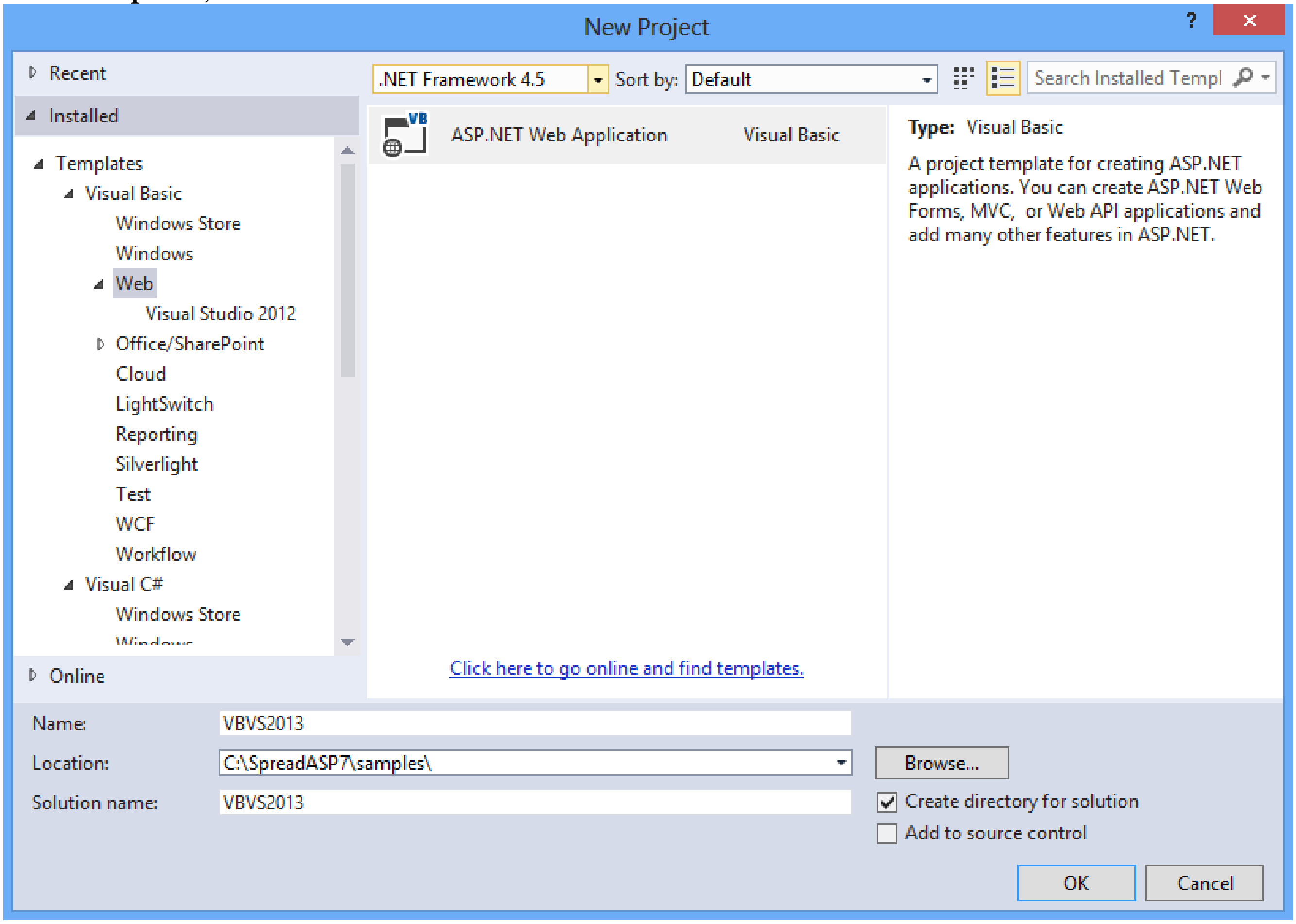
> Spread, as a child control of the page, is affected by the style settings on that page (similar to placing a table on a web form, and setting a master CSS for everything inside the page). If you create a default web application with Visual Studio

> 2010 or higher, the default master page contains CSS style settings. Spread, once placed on this default page, can be affected by the style settings and the layout may change. Avoid the following HTML tags to prevent the layout change: TD, TH, TABLE, INPUT, and TEXTAREA.

Step 1. Start Visual Studio 2013.

Step 2. Create a new Web site.

1. Select **New Project** or from the **File** menu, choose **New**, **Web Site**.
2. Under **Templates**, select **Web** under **Visual Basic** or **Visual C#.**



.NET Framework 4.5

Sort by: Default

ASP.NET Web Application Visual Basic

Type: Visual Basic

A project template for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Name: VBVS2013

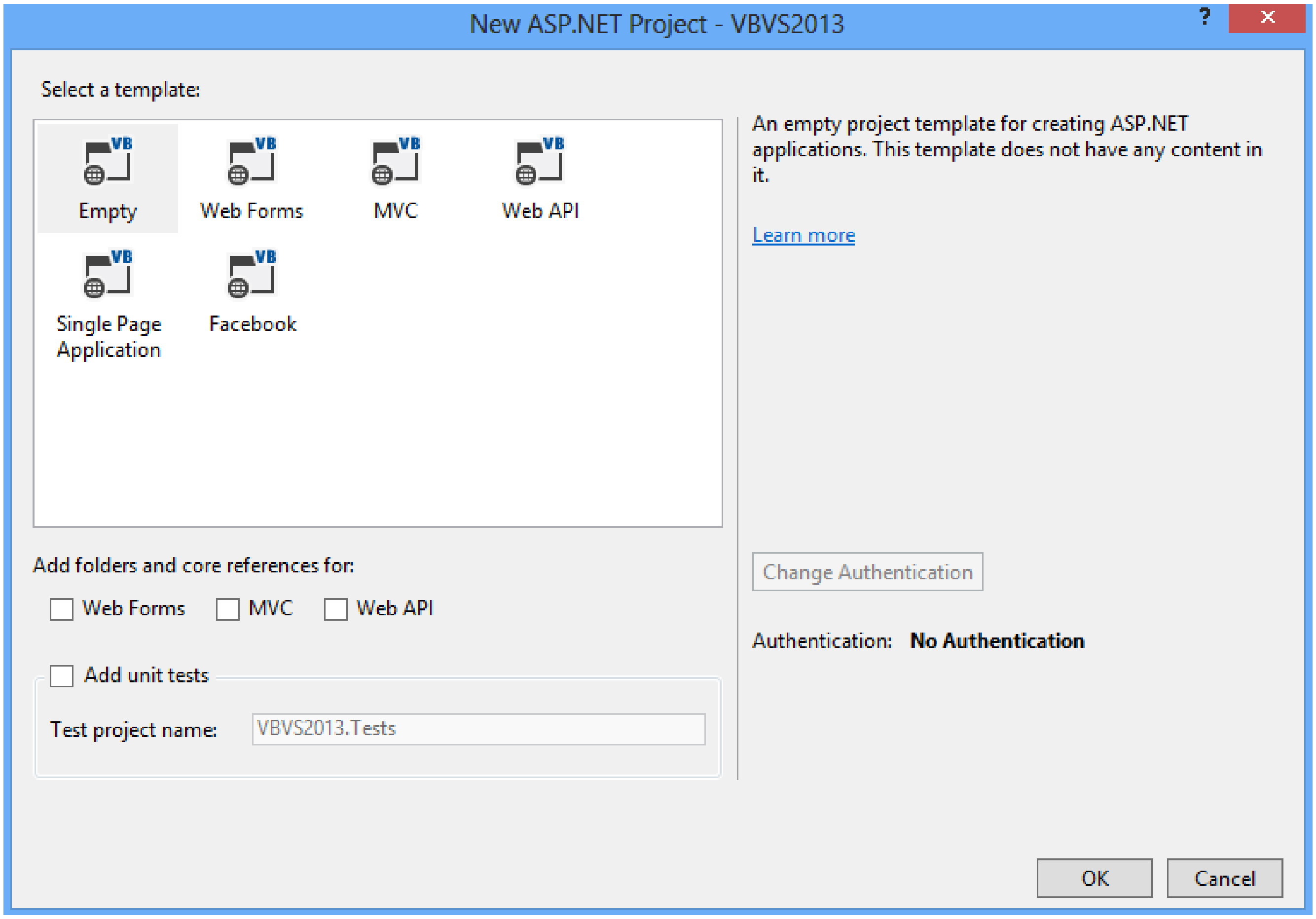
Location: C:\SpreadASP7\samples\

Solution name: VBVS2013

☑ Create directory for solution

□ Add to source control

3. Select **ASP.NET Web Application**.
4. Specify a location and name for the project.
5. Click **OK**.
6. Select a template such as **Empty**.



New ASP.NET Project - VBVS2013

Select a template:

An empty project template for creating ASP.NET applications. This template does not have any content in it.

Learn more

Add folders and core references for:

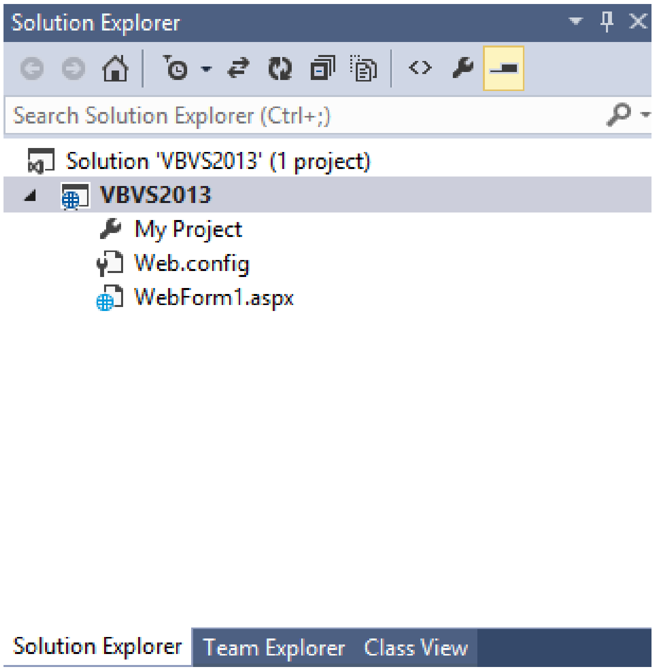
□ Web Forms □ MVC □ Web API

□ Add unit tests

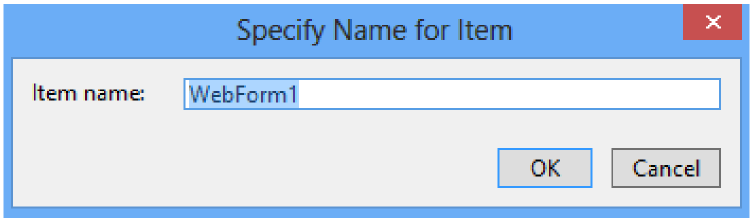
Test project name: VBVS2013.Tests

Authentication: **No Authentication**

7. Click **OK**.
   If your project does not display the **Solution Explorer**, from the **View** menu, choose **Solution Explorer**. If you used an empty site, you may wish to add a web form to the project (choose **Add**, **Web Form** after right-clicking on the project name in the **Solution Explorer**).



Specify the **Item name**. Select **OK**.



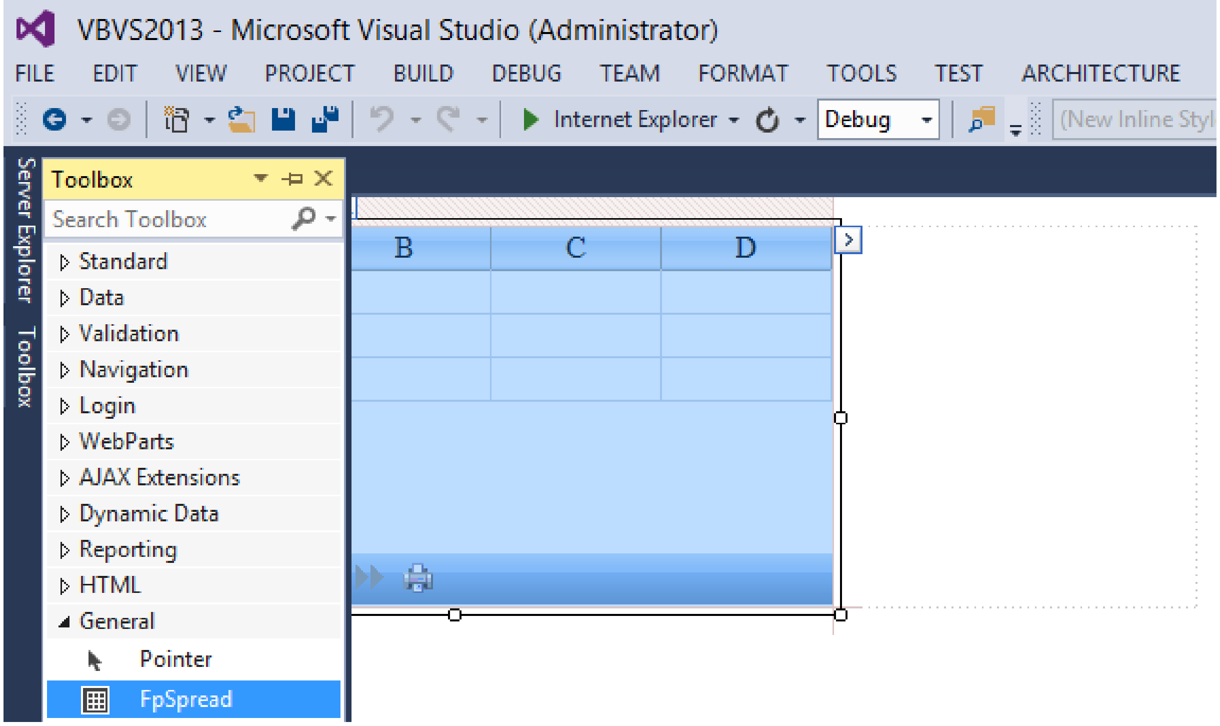
Item name: WebForm1

Step 3. Add the FpSpread 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 other categories if you have installed Spread and placed the toolbox icon in a different category).
3. If the FpSpread component is not in the **Toolbox**, right-click in the **Toolbox**, and from the pop-up menu choose **Choose Items**.
4. In the **Choose Toolbox Items** dialog, click the **.NET Framework Components** tab.
5. In the **.NET Framework Components** tab, the FpSpread component should be displayed in the list of components. Select the FpSpread component check box and click **OK**.
   If the FpSpread component is not displayed in the list of components, click **Browse** and browse to the installation path for the Spread component. Once there, select FarPoint.Web.Spread.dll and click **Open**. The FpSpread component is now displayed in the list of components. Select it and click **OK**. Select FarPoint.Web.Chart.dll if you wish to add FpChart at design time.
6. You can test that the component has been added by opening a project and inserting the component.

Step 4. Add the FpSpread component to the Web site.

1. With an open project, in the **Toolbox** under **Web Forms**, select the FpSpread component. Select FpChart if you wish to add the chart at design time.
2. On your Web Forms page, draw an FpSpread component by dragging a rectangle the size that you would like the initial component or simply double-click on the page.
3. The FpSpread component appears (as shown in this Visual Studio 2013 project).



Adding a Component to a Web Site using Visual Studio 2012

Adding an FpSpread component to a Web Form in Visual Studio 2012 involves the following steps of adding the component to a Web Site. You can either open an existing Web Site or create a new one.

> Spread, as a child control of the page, is affected by the style settings on that page (similar to placing a table on a web form, and setting a master CSS for everything inside the page). If you create a default web application with Visual Studio 2010 or higher, the default master page contains CSS style settings. Spread, once placed on this default page, can be affected by the style settings and the layout may change. Avoid the following HTML tags to prevent the layout change: TD, TH, TABLE, INPUT, and TEXTAREA.

Step 1. Start Visual Studio 2012.

Step 2. Create a new Web site.

1. From the **File** menu, choose **New**, **Web Site**.
2. In the **New Web Site** dialog, select a template. For example, from the list of Templates, choose **ASP.NET Web Forms Site** or **ASP.NET Empty Web Site**.
3. In the **Web** location area, select **HTTP** from the drop-down box, and type a location path, such as http://localhost/SpWebTest01. Alternatively, you could use the default location type as **FileSystem**, and then specify the complete path, but this requires some additional setup of copying the fp_client folder.
4. Click **OK**.
   If your project does not display the **Solution Explorer**, from the **View** menu, choose **Solution Explorer**. If you used an empty site, you may wish to add a web form to the project (choose **Add New Item** after right-clicking on the project name in the **Solution Explorer**).

   In the **Solution Explorer**, right-click on the form name, Default.aspx. You can rename it. Choose **Rename** from the pop-up menu, then type the new form name.

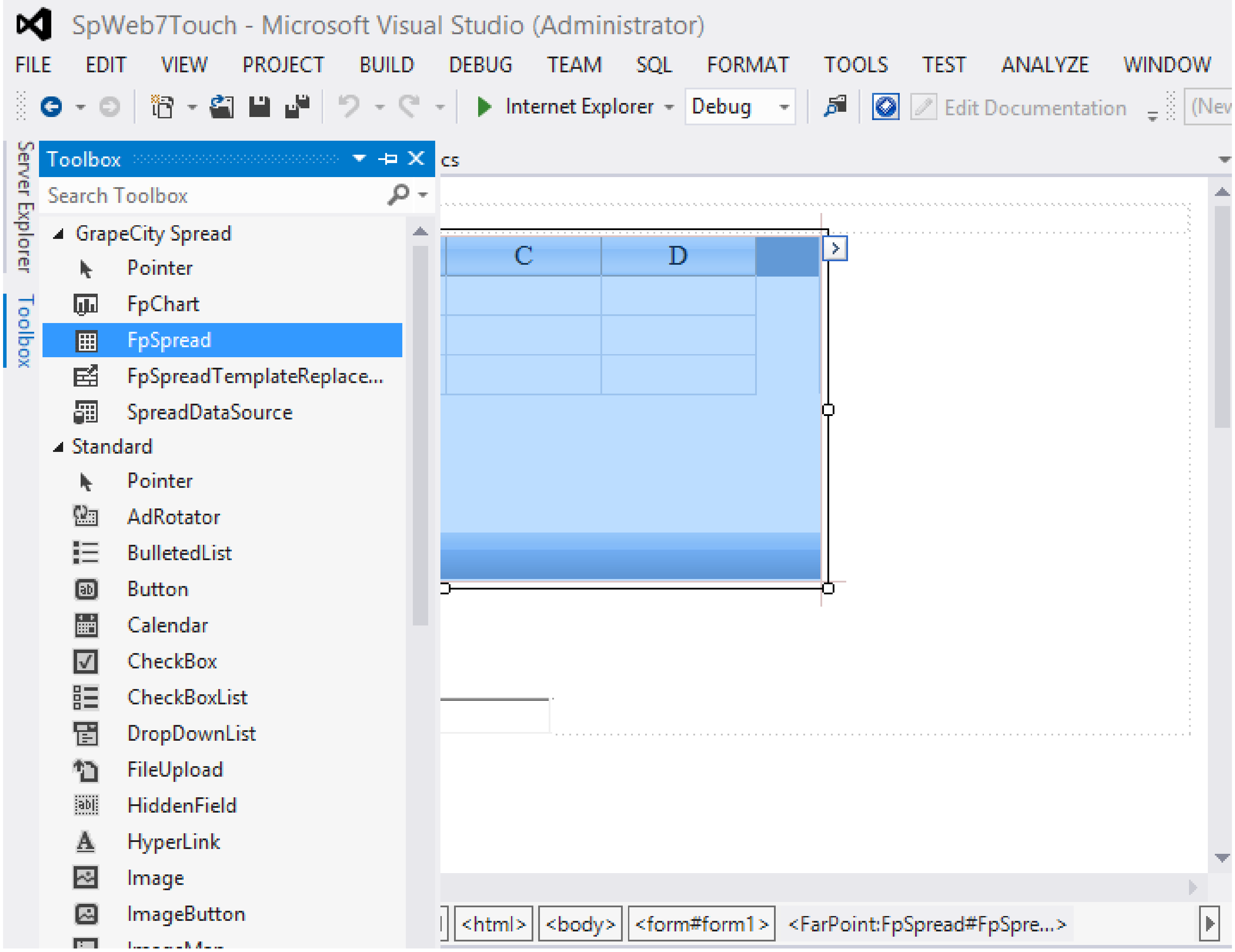
Step 3. Add the FpSpread component to the toolbox. This only has to be done once.

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 other categories if you have installed Spread and placed the toolbox icon in a different category).
3. If the FpSpread component is not in the **Toolbox**, right-click in the **Toolbox**, and from the pop-up menu choose **Choose Items**.
4. In the **Choose Toolbox Items** dialog, click the **.NET Framework Components** tab.
5. In the **.NET Framework Components** tab, the FpSpread component should be displayed in the list of components. Select the FpSpread component check box and click **OK**.
   If the FpSpread component is not displayed in the list of components, click **Browse** and browse to the installation path for the Spread component. Once there, select FarPoint.Web.Spread.dll and click **Open**. The FpSpread component is now displayed in the list of components. Select it and click **OK**. Select FarPoint.Web.Chart.dll if you wish to add FpChart at design time.
6. You can test that the component has been added by opening a project and inserting the component.

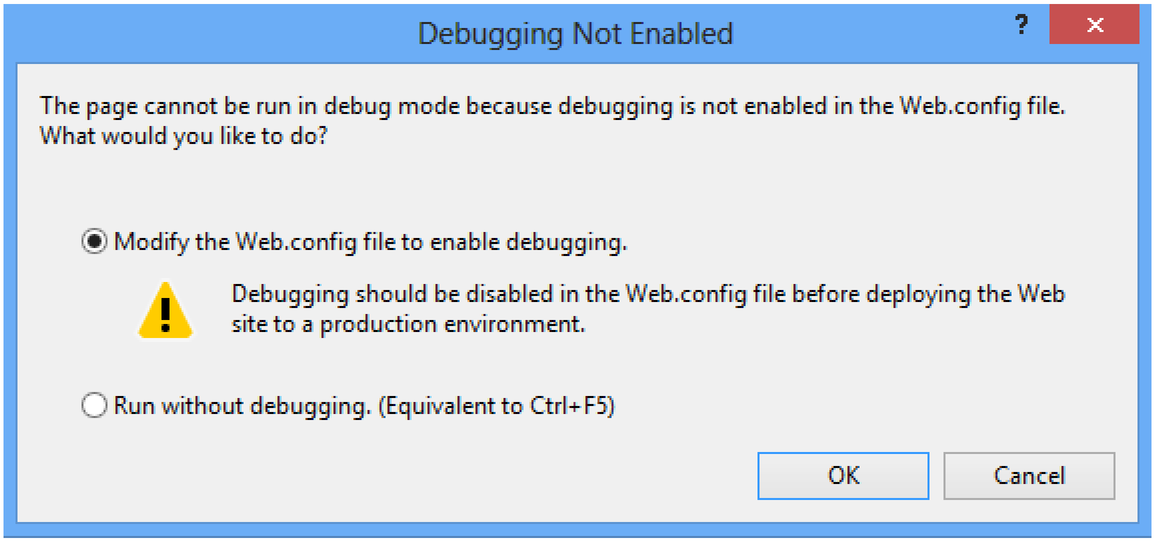
Step 4. Add the FpSpread component to the Web site.

1. With an open project, in the **Toolbox** under Web Forms, select the FpSpread component. Select FpChart if you wish to add the chart at design time.
2. On your Web Forms page, draw an FpSpread component by dragging a rectangle the size that you would like the initial component or simply double click on the page.
3. The FpSpread component appears (as shown in this Visual Studio 2012 project).



Step 5. Handle messages when running the Web site.

1. When you are ready to build and run the Web site, Visual Studio pops up an additional dialog to allow you to choose between whether to enable debugging or not to enable it. An example of the dialog appears here. Click **OK**, unless you want to select the other option to run without debugging before clicking **OK**.


Debugging Not Enabled

The page cannot be run in debug mode because debugging is not enabled in the Web.config file. What would you like to do?

● Modify the Web.config file to enable debugging.

Debugging should be disabled in the Web.config file before deploying the Web site to a production environment.

○ Run without debugging. (Equivalent to Ctrl+F5)

If you select the **File System** for the location, follow these additional instructions.

Place the fp_client folder (installed in Spread.NET\ASP.NET\..\fp_client) and its subfolders provided with Spread for ASP.NET under the folder for the Web site. Add the following code to the web.config file. For example:

XML

```
<?xml version="1.0"?>
<system.web>
...
</system.web>
<appSettings>
<add key="fp_client" value="fp_client" />
</appSettings>
</configuration>
```

If you add FpChart and you are using integrated managed pipeline mode, you may wish to set validateIntegratedModeConfiguration to false in web.config. For example:

XML

```
<system.webServer>
<validation validateIntegratedModeConfiguration="false"/>
<handlers>
...
<add name="chart" path="FpChart.axd" verb="*"
type="FarPoint.Web.Chart.ChartImageHttpHandler"/>
</handlers>
// If you are using integrated managed pipeline mode,
//set validateIntegratedModeConfiguration to false.
<validation validateIntegratedModeConfiguration="false"/>
```

Adding a Component to a Web Site using Visual Studio 2010

Adding an FpSpread component to a Web Form in Visual Studio 2010 involves the following steps of adding the component to a Web Site. You can either open an existing Web Site or create a new one.

> Spread, as a child control of the page, is affected by the style settings on that page (similar to placing a table on a web form, and setting a master CSS for everything inside the page). If you create a default web application with Visual Studio 2010 or higher, the default master page contains CSS style settings. Spread, once placed on this default page, can be affected by the style settings and the layout may change. Avoid the following HTML tags to prevent the layout change: TD, TH, TABLE, INPUT, and TEXTAREA.

Step 1. Start Visual Studio 2010.

Step 2. Create a new Web site.

1. From the **File** menu, choose **New**, **Web Site**.
2. In the **New Web Site** dialog, select a template. For example, from the list of Templates, choose **ASP.NET Web Site** or **ASP.NET Empty Web Site**.
3. In the **Web** location area, select **HTTP** from the drop-down box, and type a location path, such as http://localhost/SpWebTest01. Alternatively, you could use the default location type as **FileSystem**, and then specify the complete path, but this requires some additional setup of copying the fp_client folder.
4. Click **OK**.
   If your project does not display the **Solution Explorer**, from the **View** menu, choose **Solution Explorer**. If you used an empty site, you may wish to add a web page to the project (choose **Add New Item** after right-clicking on the project name in the **Solution Explorer**).

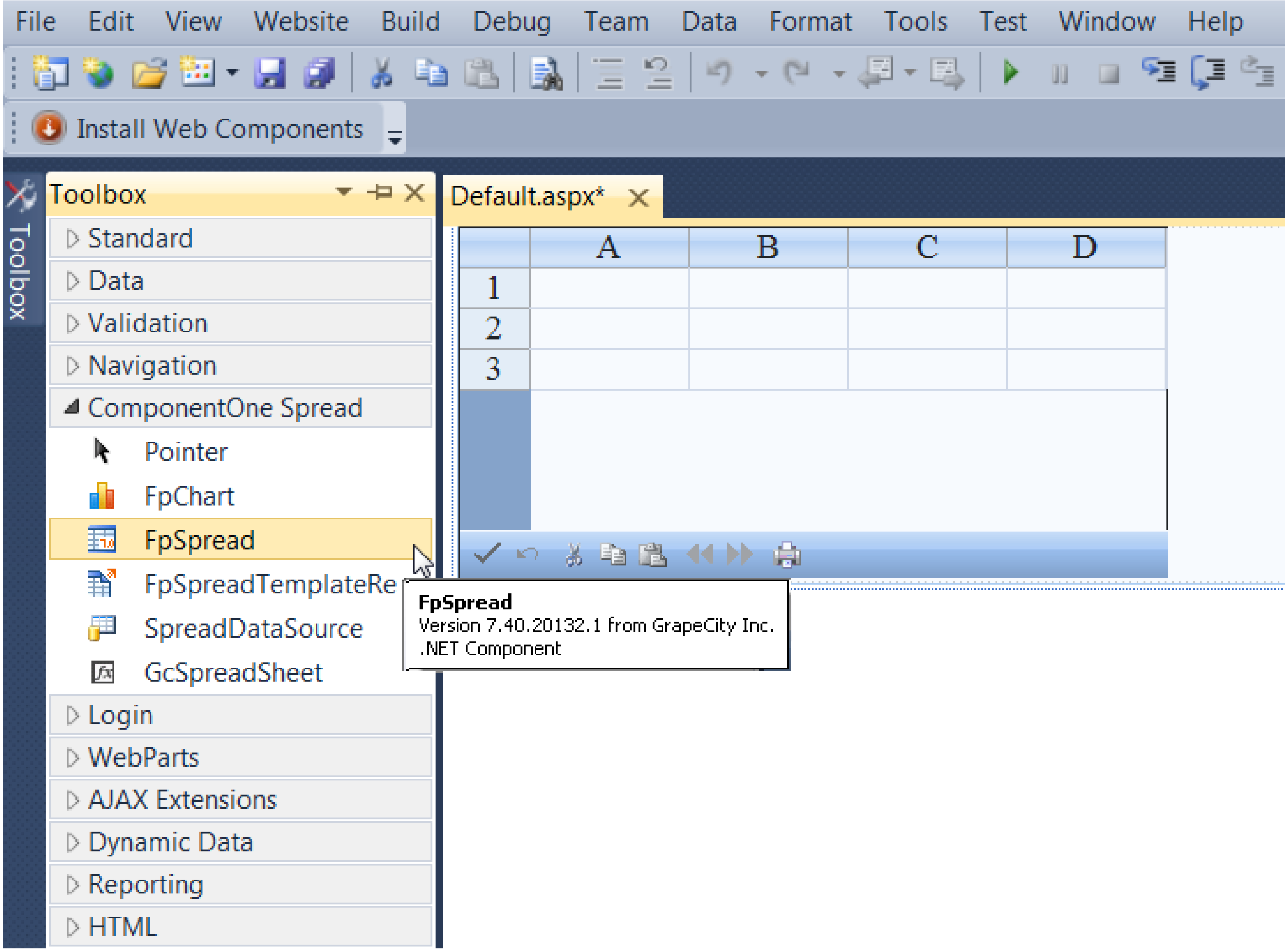
   In the **Solution Explorer**, right-click on the form name, Default.aspx. You can rename it. Choose **Rename** from the pop-up menu, then type the new form name.

Step 3. Add the FpSpread 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 other categories if you have installed Spread and placed the toolbox icon in a different category).
3. If the FpSpread 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**.
4. In the **Customize Toolbox** dialog, click the **.NET Framework Components** tab.
5. In the **.NET Framework Components** tab, the FpSpread component should be displayed in the list of components. Select the FpSpread component check box and click **OK**.
   If the FpSpread component is not displayed in the list of components, click **Browse** and browse to the installation path for the Spread component. Once there, select SpreadWeb.dll and click **Open**. The FpSpread component is now displayed in the list of components. Select it and click **OK**.
6. You can test that the component has been added by opening a project and inserting the component.

Step 4. Add the FpSpread component to the Web site.

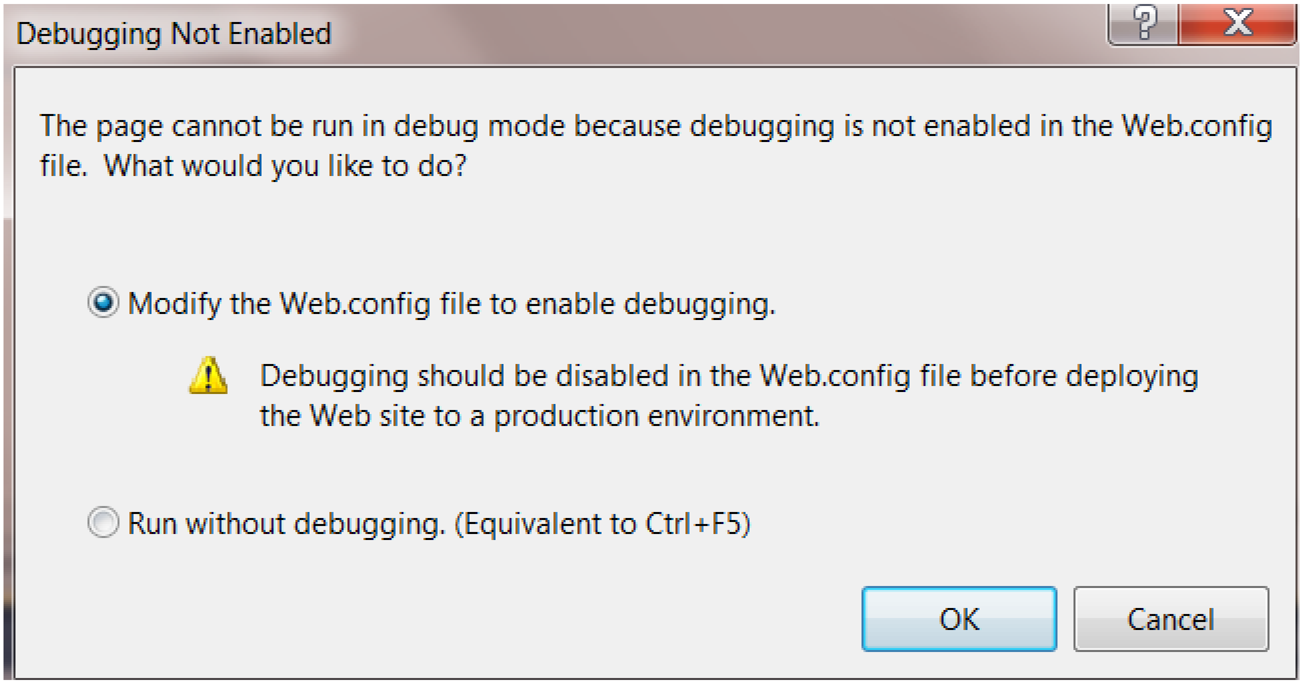
1. With an open project, in the **Toolbox** under Web Forms, select the FpSpread component.
2. On your Web Forms page, draw an FpSpread component by dragging a rectangle the size that you would like the initial component or simply double click on the page.
3. The FpSpread component appears (as shown in this Visual Studio 2010 project).



FpSpread
Version 7.40.20132.1 from GrapeCity Inc.
.NET Component

Step 5. Handle messages when running the Web site.

1. When you are ready to build and run the Web site, Visual Studio pops up an additional dialog to allow you to choose between whether to enable debugging or not to enable it. An example of the dialog appears here. Click **OK**, unless you want to select the other option to run without debugging before clicking **OK**.


Debugging Not Enabled

The page cannot be run in debug mode because debugging is not enabled in the Web.config file. What would you like to do?

● Modify the Web.config file to enable debugging.

Debugging should be disabled in the Web.config file before deploying the Web site to a production environment.

○ Run without debugging. (Equivalent to Ctrl+F5)

If you select the **File System** for the location, follow these additional instructions.

Place the fp_client folder (installed in Spread.NET\ASP.NET\..\fp_client) and its subfolders provided with Spread for ASP.NET under the folder for the Web site. Add the following code to the web.config file. For example:

XML

```xml
<?xml version="1.0"?>
<system.web>
...
</system.web>
<appSettings>
<add key="fp_client" value="fp_client" />
</appSettings>
</configuration>
```

Adding and Using JavaScript IntelliSense

The Spread component can support client-side code IntelliSense. This requires a minimum of Visual Studio 2010. This feature allows you to type the name of the control and get a list of available methods and properties. Some browsers may not support certain properties and methods.

After support has been added, type the control name followed by a dot to see the list.

The following topics contain detailed information based on the version of Visual Studio:

- **Adding JavaScript IntelliSense for Visual Studio 2012**
- **Adding JavaScript IntelliSense for Visual Studio 2010**

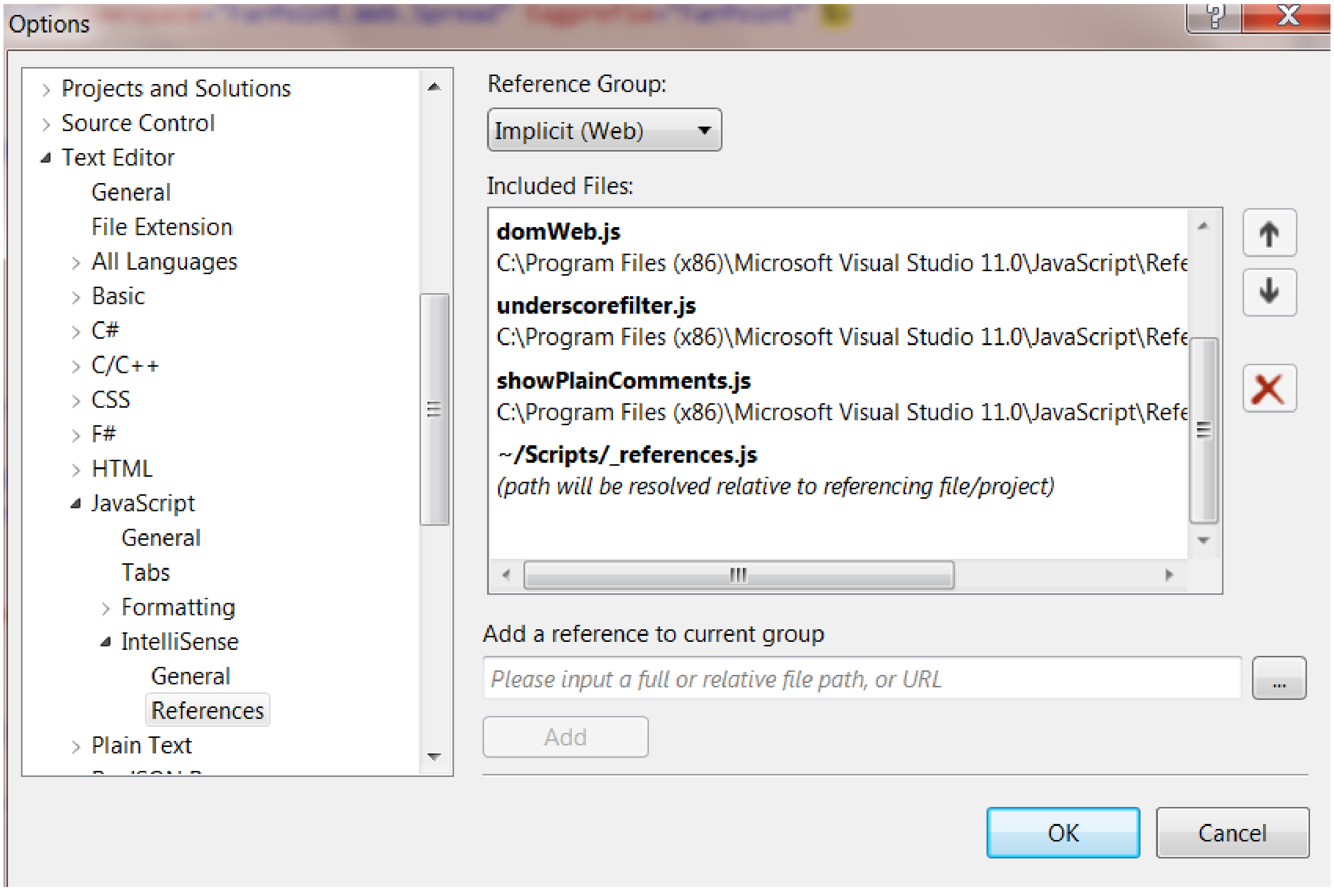
Adding JavaScript IntelliSense for Visual Studio 2012

The Spread component can support client-side code IntelliSense in Visual Studio 2012. This feature allows you to type the name of the control and get a list of available methods and properties. Some browsers may not support certain properties and methods.

After support has been added, type the control name followed by a dot to see the list.

This features requires the FpSpreadJsIntellisense.js file located in the fp_client folder. Use the following steps:

1. Verify that the ~/Scripts/_references.js reference exists in your IntelliSense settings. Select the **Options** menu under the **Tools** menu to see this dialog. Click **OK**.



Reference Group: Implicit (Web)

Included Files:

domWeb.js
C:\Program Files (x86)\Microsoft Visual Studio 11.0\JavaScript\Refe

underscorefilter.js
C:\Program Files (x86)\Microsoft Visual Studio 11.0\JavaScript\Refe

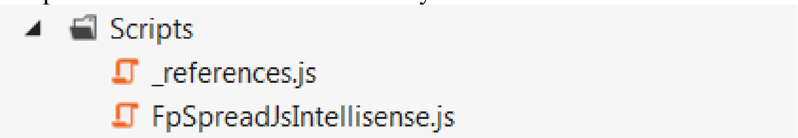
showPlainComments.js
C:\Program Files (x86)\Microsoft Visual Studio 11.0\JavaScript\Refe

~/Scripts/_references.js
(path will be resolved relative to referencing file/project)

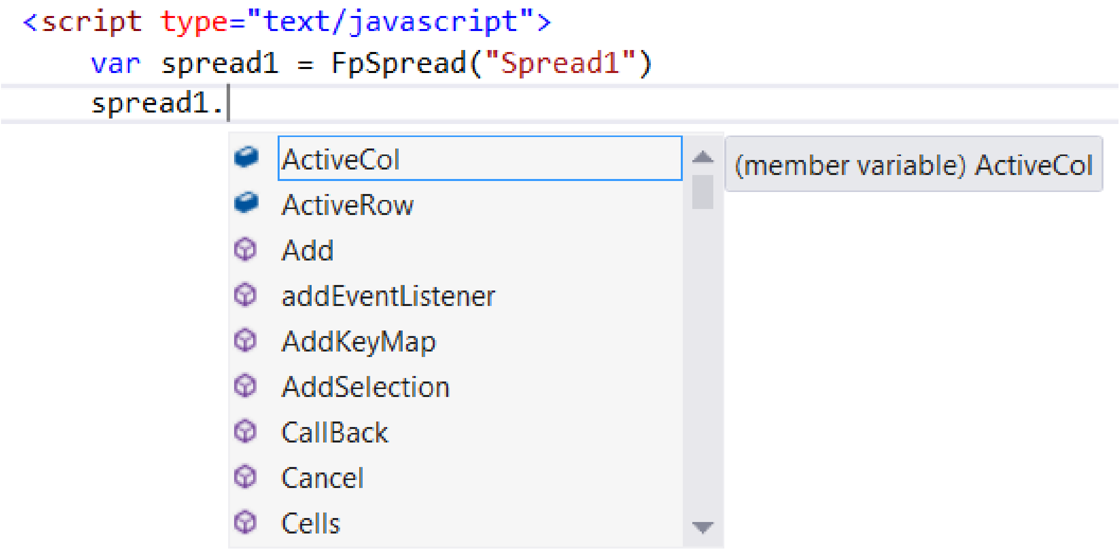
Add a reference to current group

Please input a full or relative file path, or URL

2. Add the FpSpreadJsIntellisense.js file to the same folder where the _references.js file is located. You may need to create a Scripts folder in your project if the folder does not exist. You may also need to create the _references.js file under the Scripts folder if the file does not already exist.



3. Add the following line to _references.js:
   /// <reference path="FpSpreadJsIntellisense.js" />
4. Open and close the FpSpreadJsIntellisense.js file.
5. Spread client-side methods and properties should now be displayed when you type the control name followed by a dot.



```
<script type="text/javascript">
    var spread1 = FpSpread("Spread1")
    spread1.
```

Adding JavaScript IntelliSense for Visual Studio 2010

The Spread component can support client-side code IntelliSense. This requires a minimum of Visual Studio 2010. This feature allows you to type the name of the control and get a list of available methods and properties. Some browsers may not support certain properties and methods.

After support has been added, type the control name followed by a dot to see the list.

Adding Support for IntelliSense

This features requires the FpSpreadJsIntellisense.js file located in the fp_client folder. Use the following steps:

- Create a folder named ClientResources and put the folder in the project folder (or root path of the application or web site).
- Put the FpSpreadJsIntelliSense.js file in the ClientResources folder.
- Add the following code (after the title) to the aspx page:

Code

```
<%If (False) Then%>
<script type="text/javascript"
src="./ClientResources/FpSpreadJsIntellisense.js"></script>
<% End If%>
function SomeFunction() {
var spread = FpSpread("FpSpread1");
// This variable declaration is necessary for the autocomplete.
// Type spread. here to see the autocomplete.
}
```

The final aspx page might appear as follows:

Code

```
<title>Untitled Page</title>
<%If (False) Then%>
<script type="text/javascript"
src="./ClientResources/FpSpreadJsIntellisense.js"></script>
<% End If%> <script language="javascript" type ="text/javascript" >
```

```
window.onload = function () {
var ss = document.getElementById("<%=FpSpread1.ClientID %>");
if (document.all) {
// IE
if (ss.addEventListener) {
// IE9
ss.addEventListener("DataChanged", DataChanged, false);
} else {
// Other versions of IE and IE9 quirks mode (no doctype set)
ss.onDataChanged = DataChanged;
}
}
else {
// Firefox
ss.addEventListener("DataChanged", DataChanged, false);
}
}

function DataChanged(event) {
var spread = FpSpread("FpSpread1");
> // TYPE spread. here to see the auto-complete.
}
</SCRIPT>
```

The <% if %> block will evaluate to false at run time since this code is only used for code autocomplete.

Client-side autocomplete support can also be used in a stand-alone js file with the following code (this line must be before any script):

Code

```
<reference name="FarPoint.Web.Spread.htc.FpSpreadJsIntellisense.js"
assembly="FarPoint.Web.Spread" />
```

Understanding Browser Support

The Spread component resides on the server and generates HTML pages when it is accessed by end users on the client side. The appearance and amount of interactivity of the Web page depends on the browser used on the client side. The view of the HTML pages generated by the Spread component depends on the browser being used to view the page. The component also downloads some HTML component (HTC) files to the client side. This topic summarizes some browser-specific behaviors of the product. These aspects of browser support, discussed below, include:

Browser Level

The appearance and amount of interactivity of the Web page depends on the level of browser. Broadly, an uplevel browser is one that can support client-side JavaScript, HTML version 4.0, the Microsoft Document Object Model, and cascading style sheets (CSS). A downlevel browser is one that does not. For a more detailed definition of uplevel and downlevel browser and for a list of capabilities of those browsers, refer to the browser capability information in the Microsoft .NET documentation.

Mozilla Firefox Support

While most features work in Mozilla Firefox, not all do. All features work in the latest version of Microsoft Internet Explorer (IE). Here is a list of features that are not supported in Firefox.

- Scroll bar properties (see **Customizing the Scroll Bar Colors**)

For other affects, see the discussion on the DOCTYPE Affect on Rendering.

Apple Safari Support

While most features work in Apple Safari, not all do. Here is a list of features that are not supported.

- Frozen rows and columns (**FrozenRowCount ('FrozenRowCount Property' in the on-line documentation)** property and **FrozenColumnCount ('FrozenColumnCount Property' in the on-line documentation)** property)
- ImeMode for editable cell types
- **UIVirtualization ('UIVirtualization Property' in the on-line documentation)** property

Apple Safari Support with IPad

Here is a list of features that are not supported.

- Panning mode is not supported.
- Custom toolbar above the system keyboard is not supported.
- Scroll bars are not displayed.

Google Chrome

Here is a list of features that are not supported.

- ImeMode for editable cell types
- Scroll bar properties (see **Customizing the Scroll Bar Colors**)

Client-Side Scripting

For other browsers, besides Microsoft Internet Explorer (IE) and Mozilla Firefox, the Spread client-side scripting is not supported.

In your scripting code, you will need to check the browser to see if it is Firefox or IE before calling this code, so you can call it correctly based on the browser that is viewing the page. Client-side scripting for the Firefox browser is a little different than it is for IE. You need to use Firefox's way to attach events. For example

Code

```
<HEAD>
...
<script language="javascript" type="text/javascript">
window.onload = function ()
        {
            var spread1 = document.getElementById("<%=FpSpread1.ClientID %>");
            if (document.all) {
                // IE
                if (spread1.addEventListener) {
                    // IE9
                    spread1.addEventListener("DataChanged", dataChanged, false);
                } else {
                    // Other versions of IE and IE9 quirks mode (no doctype set)
                    spread1.onDataChanged = dataChanged;
                }
            } else {
                // Firefox
                spread1.addEventListener("DataChanged", dataChanged, false);
            }
         }
         function dataChanged() {
             alert("The data has changed!");
         }

</SCRIPT>
</HEAD>
```

Firefox does not support JavaScript properties as IE does; everything is accessed with methods. For example, to get the active row and active column, you use the **GetActiveRow (on-line documentation)** method and the **GetActiveCol (on-line documentation)** method respectively. In code,

Code

```
var row = ss.GetActiveRow();
alert(row);
```

For browser-level issues specific to certain members, refer to these:

- FpSpread.**EnableClientScript ('EnableClientScript Property' in the on-line documentation)** Property
- SheetView.**MessageRowStyle ('MessageRowStyle Property' in the on-line documentation)** Property
- SheetSkin.**SelectionForeColor ('SelectionForeColor Property' in the on-line documentation)** Property

and refer to **Understanding Effects of Client-Side Validation**.

AJAX Support

Spread for ASP.NET supports AJAX in Microsoft Internet Explorer (IE) and Mozilla Firefox browsers.

DOCTYPE Affect on Rendering

The DOCTYPE settings can affect the rendering of Spread. Consider the following:

- Column widths slightly differ from what you see in Spread Designer (IE and Firefox).
- Spread inside a DIV element does not scroll as expected (IE and Firefox)
- Row height expands to show the wrapping text (IE and Firefox)

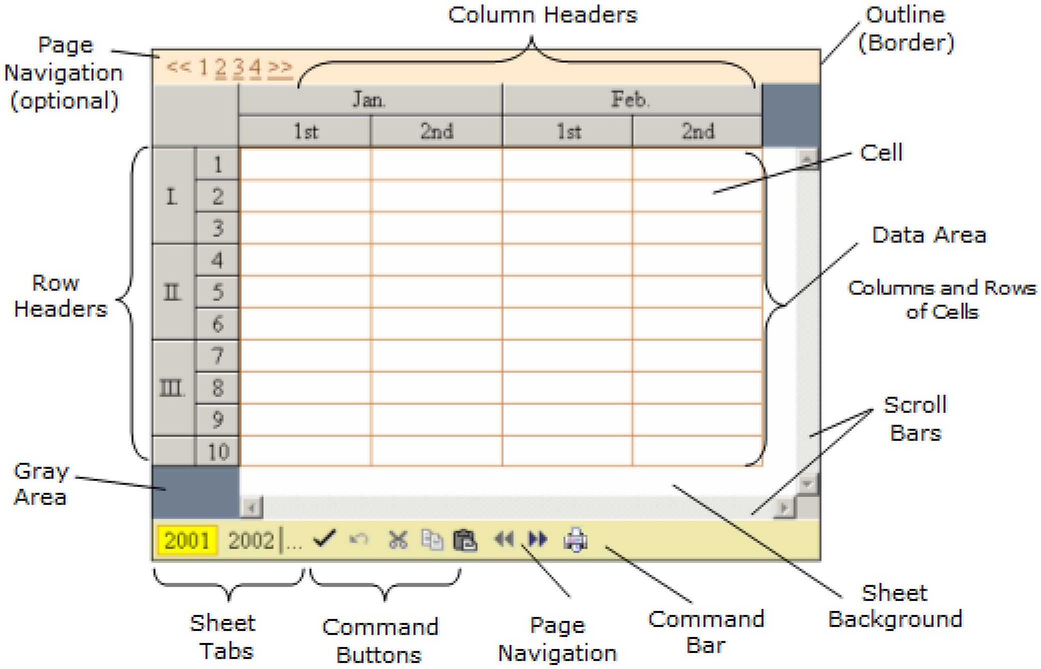
The column widths may appear narrower in Firefox than in IE. This has to do with the document type (DOCTYPE) of the HTML page. In IE, with compliant mode, you will see column widths with margins set to be larger that what you set them to. If you change the DOCTYPE of the page to Transitional or remove the margins for the cells, you should see the same column widths using either browser. For more information, refer to the **IsStrictMode** method.

Note: Spread requires that the XML name space be declared as follows:
<html xmlns="http://www.w3.org/1999/xhtml">

Spread uses HTML tables for the display on the client side. When you define a row span, it defines the span for the HTML table in the page. The default behavior in Internet Explorer for a spanned row in an HTML table is to resize to fit the text. In Spread, the row is resized to avoid layout issues. There is little documentation outlining this behavior, but you can test this behavior in Firefox where you do not see the cell resize itself to display the full text. However, it does not force the horizontal scroll bar to remain.

Understanding Parts of the Component Interface

The generated Spread component interface is made up of the tool bars (which can appear above and below the spreadsheet) and the sheet that displays the data. The figure below shows the major parts of the component interface that can be customized.



More information about the component is available in **Customizing the Appearance of the Overall Component** and **Customizing Interaction with the Overall Component**.

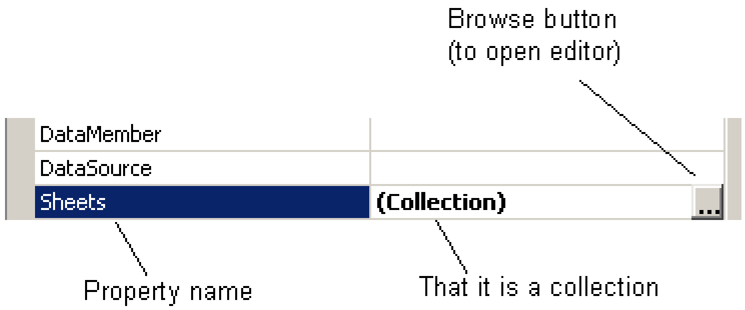
The command bar, the optional page navigation bar, and the scroll bars are described in more detail in **Customizing the Tool Bars**.

The row and column headers, considered part of the sheet, are described in more detail in **Customizing the Appearance of Headers**.

For more information on the data area, including the sheet, the rows and columns, and the cells, refer to **Customizing the Appearance of the Sheet**, **Customizing the Appearance of Rows and Columns**, and **Customizing the Appearance of a Cell**.

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 FpSpread component.



DataMember	
DataSource	
Sheets	(Collection) ...

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

Working with Web Parts

You can allow Spread for ASP.NET work as a Web Part in a Microsoft SharePoint environment. To do so, follow these steps:

1. Set the trust level to full. Set the TrustLevel in SharePoint web.config to Full.
2. Mark the Spread DLLs safe in web.config (FarPoint.CalcEngine.dll, FarPoint.Excel.dll and FarPoint.Web.Spread.dll).

This assumes you are using the latest version of SharePoint (WSS or MOSS). For earlier versions of SharePoint you also had the additional step of excluding the fp_client folder from the SharePoint server. This is no longer necessary.

Spread can be used inside the Web Part you are developing.

In earlier versions of the product, the FpSpread.RenderWebPart method was used for the .NET Framework 1.x when you created a Web Part based on Microsoft.SharePoint.WebPartPages.WebPart class. In .NET Framework 2.x, the Web Part becomes part of the framework. If you use the framework WebPart class, System.Web.UI.WebControls.WebParts.WebPart, the FpSpread.RenderWebPart method is no longer needed.

See a good introduction to Web Parts at:

- https://msdn.microsoft.com/en-us/library/ee231579.aspx

Working with Windows Azure

You can use Spread for ASP.NET in a Windows Azure project. Use the following steps:

1. Copy the fp_client folder to your WebRole project folder.
2. Include this fp_client folder in your WebRole project and add the following setting to the web.config file.

Code

```
<appSettings>
<add key="fp_client" value="fp_client"/>
</appSettings>
```

You do not need to edit web.config if you use the development fabric in Visual Studio.

You can also use the Chart control in a Windows Azure project. You would need to add the ChartImageHttpHandler to the web server section of the web.config file. If you are using integrated managed pipeline mode, set **validateIntegratedModeConfiguration** to False. For example:

Code

```
<system.webServer>
<validation validateIntegratedModeConfiguration="false"/>
<handlers>
...
<add name="chart" path="FpChart.axd" verb="*"
type="FarPoint.Web.Chart.ChartImageHttpHandler"/>
```

Working with Microsoft ASP.NET MVC 5

You can use Spread for ASP.NET in an MVC 5 project. MVC support in Spread for ASP.NET requires Microsoft ASP.NET MVC 5, Microsoft Visual Studio 2013 with .NET 4.0 Framework, and the Microsoft ADO.NET Entity 4.1 Framework.

The Razor view generally uses @ in front of the name and the ASPX view generally uses <% %>around the name. Use the following steps to create a project with Spread:

1. Reference FarPoint.Mvc.Spread.dll and FarPoint.Web.Spread.dll in the project.
2. Add the Spread information to the Licenses.licx file:

Code

```
FarPoint.Web.Spread.FpSpread, FarPoint.Web.Spread, Version=10.40.20162.0,
Culture=neutral, PublicKeyToken=327c3516b1b18457
FarPoint.Mvc.Spread.FpSpread, FarPoint.Mvc.Spread, Version=10.40.20162.0,
Culture=neutral, PublicKeyToken=327c3516b1b18457
```

3. Open Global.asax.cs, go to the Application_Start function and add the following registration code:

> If you use Spread MVC on .NET Framework 4.0 or above, remove FarPoint.Mvc.Spread.MvcSpreadVirtualPathProvider.AppInitialize(); from Application_Start().

C#

```
protected void Application_Start()
{
//FarPoint.Mvc.Spread.MvcSpreadVirtualPathProvider.AppInitialize();
AreaRegistration.RegisterAllAreas();
RegisterGlobalFilters(GlobalFilters.Filters);
RegisterRoutes(RouteTable.Routes);
//ModelBinders.Binders.DefaultBinder =
ModelBinders.Binders[typeof(FarPoint.Mvc.Spread.FpSpread)];
//ModelBinders.Binders.Add(typeof(FarPoint.Mvc.Spread.FpSpread), new
FarPoint.Mvc.Spread.MvcSpreadModelBinder());
}
```

4. Declare Spread with the MVC Spread namespace:

Code

```
@using FarPoint.Mvc.Spread; or <%@ Import Namespace="Farpoint.Mvc.Spread"
%>
```

This allows you to have an MVC Spread with the following code:

Code

```
@Html.FpSpread("FpSpread1"); or <%=Html.FpSpread("FpSpread1")%>
```

FpSpread1 is the Spread ID. It should be unique.

5. Provide access to MVC Spread from the Controller. When the user posts back data to the server, the developer can access the declared MVC Spread as an argument (the Spread has full ViewState and new postback data). For example:

C#

```
public ActionResult
Index([FarPoint.Mvc.Spread.MvcSpread]FarPoint.Mvc.Spread.FpSpread FpSpread1)
{
ViewBag.Message = "Welcome to GrapeCity";
if (FpSpread1 != null)
{
var value = FpSpread1.ActiveSheetView.Cells[0, 0].Value;
}
return View();
}
```

If you do not want to use an attribute, open the Global.asax.cs and uncomment one of the lines in step 3.

Make sure the Spread ID is the same in the view code and in the controller action parameter.

6. Attach Spread events:
Spread supports attaching events from the Controller only. If there is an AJAX postback, the Spread events will not be handled. MvcSpread allows attaching 3 main events: Init, Load, and PreRender. Events can be grouped or ungrouped. Use one of the following methods to handle the event:

Create a function with a special name.

The special name indicates that "I want to bind this function to a Spread event". For example, to attach to the Load event of FpSpread1, the function looks like the following:

C#

```
public void FpSpread1_Load(object sender, EventArgs e)
{
}
Use MvcSpreadEventAttribute.
In some cases, you may want to reserve a special name (like "FpSpread1_Load").
This can only be done by using the second method: MvcSpreadEventAttribute. The
event handler can be shared globally or in a group.
This example handles the Init event for all FpSpreads with the ID of FpSpread1 in
any view, globally:
[FarPoint.Mvc.Spread.MvcSpreadEvent("Init", "FpSpread1")]
private void _init(object sender, EventArgs e)
{
}
```

This solution also provides the ability to bind one function to many different Spreads. The following example handles the Init event for all FpSpreads with the ID of FpSpread1 or FpSpread2 in any view, globally:

C#

```
[FarPoint.Mvc.Spread.MvcSpreadEvent("Init", new string[] {"FpSpread1",
"FpSpread2"})]
private void _init(object sender, EventArgs e)
{
}
```

The second solution requires that you indicate implicitly that the function (with special name) should not be attached automatically:

C#

```
[FarPoint.Mvc.Spread.NoMvcSpreadEvent]
private void FpSpread1_Init(object sender, EventArgs e)
{
}
```

By attaching to the Init or Load event, you can attach to other custom Spread events such as TopRowChange, UpdateCommand, and so on.

Additional information about global, grouped, and ungrouped events:

Attaching events with a grouped MvcSpread:

If the MvcSpread is grouped, use a group name. For example:

C#

```
// groupName is "GroupName" -> grouped
```

```
public ActionResult Index([MvcSpread("GroupName", false)] FpSpread FpSpread1)
// no groupName specified -> not grouped
public ActionResult Index([MvcSpread(false)] FpSpread FpSpread1)
```

The following examples show how to handle group events:

C#

```
// method Func3() is used to handle the Load event for all FpSpreads with ID of
FpSpread1 in any view, inside the group called "GroupName"
[MvcSpreadEvent("Load", "GroupName", "FpSpread1")]
private void Func3(object sender, EventArgs e)
{
}
// method Func4() is used to handle the Load event for all FpSpreads with ID of
FpSpread1, or FpSpread2 in any view, inside the group called "GroupName"
[MvcSpreadEvent("Load", "GroupName", new string[] { "FpSpread1", "FpSpread2" })]
private void Func4(object sender, EventArgs e)
{
}
```

Any function named "[Spread ID]_[Event Name]" is treated as a global event handler. The event name is indicated by [Event Name] and only an MvcSpread with an ID the same as [Spread ID] can handle this event handler:

C#

```
// this is a global handler for all FpSpread1 Load events
public void FpSpread1_Load(object sender, EventArgs e)
{
}
```

You can exclude some groups (that contain FpSpread controls that are not handled by this method) from global event handlers. These groups can be referenced by their names. For example:

C#

```
// method abc() is used to handle the Load event for all FpSpread1 controls,
excluding the ones that belong to GroupName1 or GroupName2
[MvcSpreadEvent("Load", "FpSpread1")]
[MvcSpreadEventExclude("GroupName1", "GroupName2")]
public void abc(object sender, EventArgs e)
{
}
// this method is used to handle the Load event for all FpSpread1 controls,
excluding the ones that belong to GroupName1 or GroupName2
[MvcSpreadEventExclude("GroupName1", "GroupName2")]
public void FpSpread1_Load(object sender, EventArgs e)
{
}
```

Developer does not declare MvcSpread inside controller:

The MvcSpread is ungrouped if the MvcSpread is declared inside the view. Using a group name with MvcSpread in the view may cause issues since there is a case where the group name in the controller and the group name in the view are different. If MvcSpread is not declared in the controller, the events are attached to global event handlers only.

Developer declares MvcSpread inside controller:

If the event handlers are global (group name not declared in MvcSpreadEventAttribute), they attach to all Spread controls whether grouped or not.

If the event handlers are private (group name explicitly declared in MvcSpreadEventAttribute), they attach only to Spread controls with the same group name.

Spread controls without a group name are attached by global event handlers.

If there are two event handlers for the same event, the first one is private and the second one is global.

The current Spread looks for the global event handler first when binding events. If found, binding for the current event happens first. The private one with the same group name happens next.

Note: Using the create parameter to notify MvcSpreadModelBinder to create a new instance of MvcSpread the first time does not affect attaching events.An event handler is attached to MvcSpread if the event ID list contains the MvcSpread ID, regardless of whether the event handler is private or global.

7. Pass MVC Spread from Controller to View:
   If an instance of FpSpread is created by a model, it is applied to the view automatically. The code that renders FpSpread with the same ID, renders the current FpSpread to the client browser:

   C#

   ```
   public ActionResult Index([MvcSpread(true)] FpSpread FpSpread1)
   {
   FpSpread1.ActiveSheetView.Rows.Count = 30;
   return View();
   }
   ```

   In the parameter list Index() action the FpSpread1 is declared specifically as [MvcSpread(true)] FpSpread FpSpread1. When running this code, a new instance of FpSpread is created by our ModelBinder. The ActiveSheetView.Rows.Count is set to 30, and then this instance is applied to the view automatically. The FpSpread control with the ID of "FpSpread1" in the view receives these changes.

Working with Microsoft ASP.NET MVC 3

You can use Spread for ASP.NET in an MVC 3 project. MVC support in Spread for ASP.NET requires Microsoft ASP.NET MVC3, Microsoft Visual Studio 2010 with .NET 4.0 Framework, and the Microsoft ADO.NET Entity 4.1 Framework.

The Razor view generally uses @ in front of the name and the ASPX view generally uses <% %>around the name. Use the following steps to create a project with Spread:

1. Reference FarPoint.Mvc.Spread.dll and FarPoint.Web.Spread.dll in the project.
2. Add the Spread information to the Licenses.licx file:

   Code

   ```
   FarPoint.Web.Spread.FpSpread, FarPoint.Web.Spread, Version=8.40.20143.0,
   Culture=neutral, PublicKeyToken=327c3516b1b18457
   FarPoint.Mvc.Spread.FpSpread, FarPoint.Mvc.Spread, Version=8.40.20143.0,
   Culture=neutral, PublicKeyToken=327c3516b1b18457
   ```

3. Open Global.asax.cs, go to the Application_Start function and add the following registration code:

   C#

   ```
   protected void Application_Start()
   {
   FarPoint.Mvc.Spread.MvcSpreadVirtualPathProvider.AppInitialize();
   AreaRegistration.RegisterAllAreas();
   RegisterGlobalFilters(GlobalFilters.Filters);
   RegisterRoutes(RouteTable.Routes);
   //ModelBinders.Binders.DefaultBinder =
   ```

```
ModelBinders.Binders[typeof(FarPoint.Mvc.Spread.FpSpread)];
//ModelBinders.Binders.Add(typeof(FarPoint.Mvc.Spread.FpSpread), new
FarPoint.Mvc.Spread.MvcSpreadModelBinder());
}
```

4. Declare Spread with the MVC Spread namespace:

Code

```
@using FarPoint.Mvc.Spread; or <%@ Import Namespace="Farpoint.Mvc.Spread"
%>
```

This allows you to have an MVC Spread with the following code:

Code

```
@Html.FpSpread("FpSpread1"); or <%=Html.FpSpread("FpSpread1")%>
```

FpSpread1 is the Spread ID. It should be unique.

5. Provide access to MVC Spread from the Controller. When the user posts back data to the server, the developer can access the declared MVC Spread as an argument (the Spread has full ViewState and new postback data). For example:

C#

```
public ActionResult
Index([FarPoint.Mvc.Spread.MvcSpread]FarPoint.Mvc.Spread.FpSpread FpSpread1)
{
ViewBag.Message = "Welcome to GrapeCity";
if (FpSpread1 != null)
{
var value = FpSpread1.ActiveSheetView.Cells[0, 0].Value;
}
return View();
}
```

If you do not want to use an attribute, open the Global.asax.cs and uncomment one of the lines in step 3.

Make sure the Spread ID is the same in the view code and in the controller action parameter.

6. Attach Spread events:
Spread supports attaching events from the Controller only. If there is an AJAX postback, the Spread events will not be handled. MvcSpread allows attaching 3 main events: Init, Load, and PreRender. Events can be grouped or ungrouped. Use one of the following methods to handle the event:

Create a function with a special name.

The special name indicates that “I want to bind this function to a Spread event”. For example, to attach to the Load event of FpSpread1, the function looks like the following:

C#

```
public void FpSpread1_Load(object sender, EventArgs e)
{
}
Use MvcSpreadEventAttribute.

In some cases, you may want to reserve a special name (like “FpSpread1_Load”).
This can only be done by using the second method: MvcSpreadEventAttribute. The
event handler can be shared globally or in a group.
This example handles the Init event for all FpSpreads with the ID of FpSpread1 in
any view, globally:
```

```
[FarPoint.Mvc.Spread.MvcSpreadEvent("Init", "FpSpread1")]
private void _init(object sender, EventArgs e)
{
}
```

This solution also provides the ability to bind one function to many different Spreads. The following example handles the Init event for all FpSpreads with the ID of FpSpread1 or FpSpread2 in any view, globally:

C#

```
[FarPoint.Mvc.Spread.MvcSpreadEvent("Init", new string[] {"FpSpread1",
"FpSpread2"})]
private void _init(object sender, EventArgs e)
{
}
```

The second solution requires that you indicate implicitly that the function (with special name) should not be attached automatically:

C#

```
[FarPoint.Mvc.Spread.NoMvcSpreadEvent]
private void FpSpread1_Init(object sender, EventArgs e)
{
}
```

By attaching to the Init or Load event, you can attach to other custom Spread events such as TopRowChange, UpdateCommand, and so on.

Additional information about global, grouped, and ungrouped events:

Attaching events with a grouped MvcSpread:

If the MvcSpread is grouped, use a group name. For example:

C#

```
// groupName is "GroupName" -> grouped
public ActionResult Index([MvcSpread("GroupName", false)] FpSpread FpSpread1)
// no groupName specified -> not grouped
public ActionResult Index([MvcSpread(false)] FpSpread FpSpread1)
```

The following examples show how to handle group events:

C#

```
// method Func3() is used to handle the Load event for all FpSpreads with ID of
FpSpread1 in any view, inside the group called "GroupName"
[MvcSpreadEvent("Load", "GroupName", "FpSpread1")]
private void Func3(object sender, EventArgs e)
{
}
// method Func4() is used to handle the Load event for all FpSpreads with ID of
FpSpread1, or FpSpread2 in any view, inside the group called "GroupName"
[MvcSpreadEvent("Load", "GroupName", new string[] { "FpSpread1", "FpSpread2" })]
private void Func4(object sender, EventArgs e)
{
}
```

Any function named "[Spread ID]_[Event Name]" is treated as a global event handler. The event name is indicated by [Event Name] and only an MvcSpread with an ID the same as [Spread ID] can handle this event handler:

C#

```
// this is a global handler for all FpSpread1 Load events
public void FpSpread1_Load(object sender, EventArgs e)
{
}
```

You can exclude some groups (that contain FpSpread controls that are not handled by this method) from global event handlers. These groups can be referenced by their names. For example:

C#

```
// method abc() is used to handle the Load event for all FpSpread1 controls,
excluding the ones that belong to GroupName1 or GroupName2
[MvcSpreadEvent("Load", "FpSpread1")]
[MvcSpreadEventExclude("GroupName1", "GroupName2")]
public void abc(object sender, EventArgs e)
{
}
// this method is used to handle the Load event for all FpSpread1 controls,
excluding the ones that belong to GroupName1 or GroupName2
[MvcSpreadEventExclude("GroupName1", "GroupName2")]
public void FpSpread1_Load(object sender, EventArgs e)
{
}
```

Developer does not declare MvcSpread inside controller:

The MvcSpread is ungrouped if the MvcSpread is declared inside the view. Using a group name with MvcSpread in the view may cause issues since there is a case where the group name in the controller and the group name in the view are different. If MvcSpread is not declared in the controller, the events are attached to global event handlers only.

Developer declares MvcSpread inside controller:

If the event handlers are global (group name not declared in MvcSpreadEventAttribute), they attach to all Spread controls whether grouped or not.

If the event handlers are private (group name explicitly declared in MvcSpreadEventAttribute), they attach only to Spread controls with the same group name.

Spread controls without a group name are attached by global event handlers.

If there are two event handlers for the same event, the first one is private and the second one is global.

The current Spread looks for the global event handler first when binding events. If found, binding for the current event happens first. The private one with the same group name happens next.

Note: Using the create parameter to notify MvcSpreadModelBinder to create a new instance of MvcSpread the first time does not affect attaching events.An event handler is attached to MvcSpread if the event ID list contains the MvcSpread ID, regardless of whether the event handler is private or global.

7. Pass MVC Spread from Controller to View:
If an instance of FpSpread is created by a model, it is applied to the view automatically. The code that renders FpSpread with the same ID, renders the current FpSpread to the client browser:

C#

```
public ActionResult Index([MvcSpread(true)] FpSpread FpSpread1)
{
FpSpread1.ActiveSheetView.Rows.Count = 30;
return View();
}
```

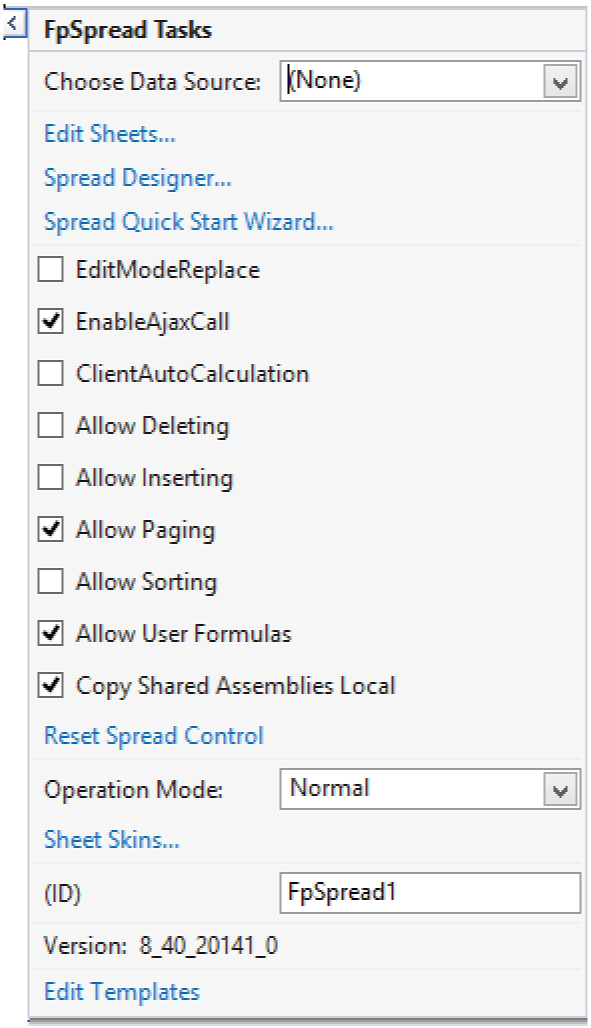
In the parameter list Index() action the FpSpread1 is declared specifically as [MvcSpread(true)] FpSpread FpSpread1. When running this code, a new instance of FpSpread is created by our ModelBinder. The ActiveSheetView.Rows.Count is set to 30, and then this instance is applied to the view automatically. The FpSpread control with the ID of "FpSpread1" in the view receives these changes.

Copying Shared Assemblies to Local Folder

FarPoint.CalcEngine.dll, FarPoint.Excel.dll, and FarPoint.PDF.dll are installed to the GAC by default when installing Spread for ASP.NET.

You can use the smart tag verb "Copy Shared Assemblies Local" to copy FarPoint.CalcEngine.dll, FarPoint.Excel.dll, and FarPoint.PDF.dll to the local bin folder on the web site when deploying.

The smart tag verb appears as follows:



FpSpread Tasks

Choose Data Source: (None)

Edit Sheets...

Spread Designer...

Spread Quick Start Wizard...

☐ EditModeReplace

☑ EnableAjaxCall

☐ ClientAutoCalculation

☐ Allow Deleting

☐ Allow Inserting

☑ Allow Paging

☐ Allow Sorting

☑ Allow User Formulas

☑ Copy Shared Assemblies Local

Reset Spread Control

Operation Mode: Normal

Sheet Skins...

(ID) FpSpread1

Version: 8_40_20141_0

Edit Templates

The following entry is added to the web.config file:

Code

```
<appSettings>
<add key="fp_CopySharedAssembliesLocal" value="True" />
```

```
</appSettings>
```

Working with Strongly Typed Data Controls

Strong-typed data controls support binding using a new syntax that is available in ASP.NET 4.5. This allows you to directly reference properties on the data source object in the markup. For example, in order to bind data to PreviewRowTemplate and RowEditTemplate before ASP.NET 4.5, code to bind to the **FirstName** property on your data source item looked like this:

Code

```
<FarPoint:FpSpread ID="FpSpread1">
  <Sheets>
    <FarPoint:SheetView SheetName="Sheet1">
      <PreviewRowTemplate>
      First Name is: <%#DataBinder.Eval(Item, "FirstName") %>
      </PreviewRowTemplate>
      <RowEditTemplate>
        First Name is: <%#DataBinder.Eval(Item, "FirstName") %>
      </RowEditTemplate>
    </FarPoint:SheetView>
  </Sheets>
</FarPoint:FpSpread>
```

With ASP.NET 4.5 you can use code like the following:

Code

```
<FarPoint:FpSpread ID="FpSpread1">
  <Sheets>
    <FarPoint:SheetView SheetName="Sheet1">
      <PreviewRowTemplate>
      First Name is: <%#Item.FirstName %>
      </PreviewRowTemplate>
      <RowEditTemplate>
        First Name is: <%#Item.FirstName %>
      </RowEditTemplate>
    </FarPoint:SheetView>
  </Sheets>
</FarPoint:FpSpread>
```

Using code in your markup this way allows you to get the added benefits of IntelliSense listing of your data source object's properties and validation of the property name at design time.

Getting More Practice

Here are the tasks for getting more practice with the product.

- **Understanding Procedures in the Documentation**
- **Getting Technical Support (on-line documentation)**

Understanding Procedures in 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 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. Typically, for either way of writing code, there is an example given.

Documentation Provided

The Spread for ASP.NET 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 component or one of its members has focus, press F1 to display the Spread for ASP.NET help.

You can also access the help file in a stand-alone window by choosing **Start**->**Programs**->**GrapeCity**->...->**Product**->**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 commonly used procedures covered in the documentation:

- **Adding a Note to a Cell**
- **Adding a Row or Column**
- **Adding a Sheet**
- **Applying a Skin to a Sheet**
- **Customizing the Dimensions of the Component**
- **Customizing the Number of Rows or Columns**
- **Creating a Custom Function**
- **Creating a Custom Name**
- **Creating a Skin for Sheets**
- **Creating Alternating Rows**
- **Customizing the Outline of the Component**
- **Customizing the Sheet Corner**
- **Displaying Grid Lines on the Sheet**
- **Displaying Scroll Bars**
- **Locking a Cell**
- **Nesting Functions in a Formula**
- **Opening Existing Files**
- **Placing a Formula in Cells**
- **Removing a Row or Column**
- **Removing a Sheet**
- **Saving Data to a File**

- **Setting the Background Color of the Sheet**
- **Setting the Row Height or Column Width**
- **Specifying a Cell Reference Style in a Formula**
- **Using a Circular Reference in a Formula**
- **Working with Editable Cell Types**
- **Working with Graphical Cell Types**

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 files, see **Understanding Procedures in the Documentation**.
- Product forum at https://www.grapecity.com/en/forums#spread

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

Web site:	https://www.grapecity.com/en/forums
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 5:30 p.m. Eastern time, Monday through Friday.

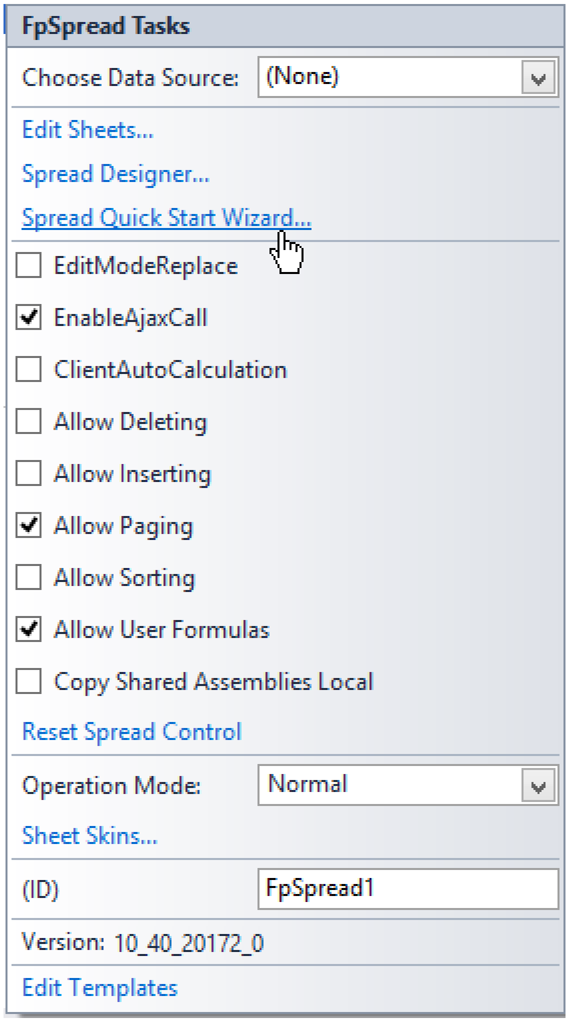
Understanding the Spread Wizard

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 Web Form in Visual Studio as shown in this figure.



FpSpread Tasks

Choose Data Source: (None)

Edit Sheets...

Spread Designer...

Spread Quick Start Wizard...

☐ EditModeReplace

☑ EnableAjaxCall

☐ ClientAutoCalculation

☐ Allow Deleting

☐ Allow Inserting

☑ Allow Paging

☐ Allow Sorting

☑ Allow User Formulas

☐ Copy Shared Assemblies Local

Reset Spread Control

Operation Mode: Normal

Sheet Skins...

(ID) FpSpread1

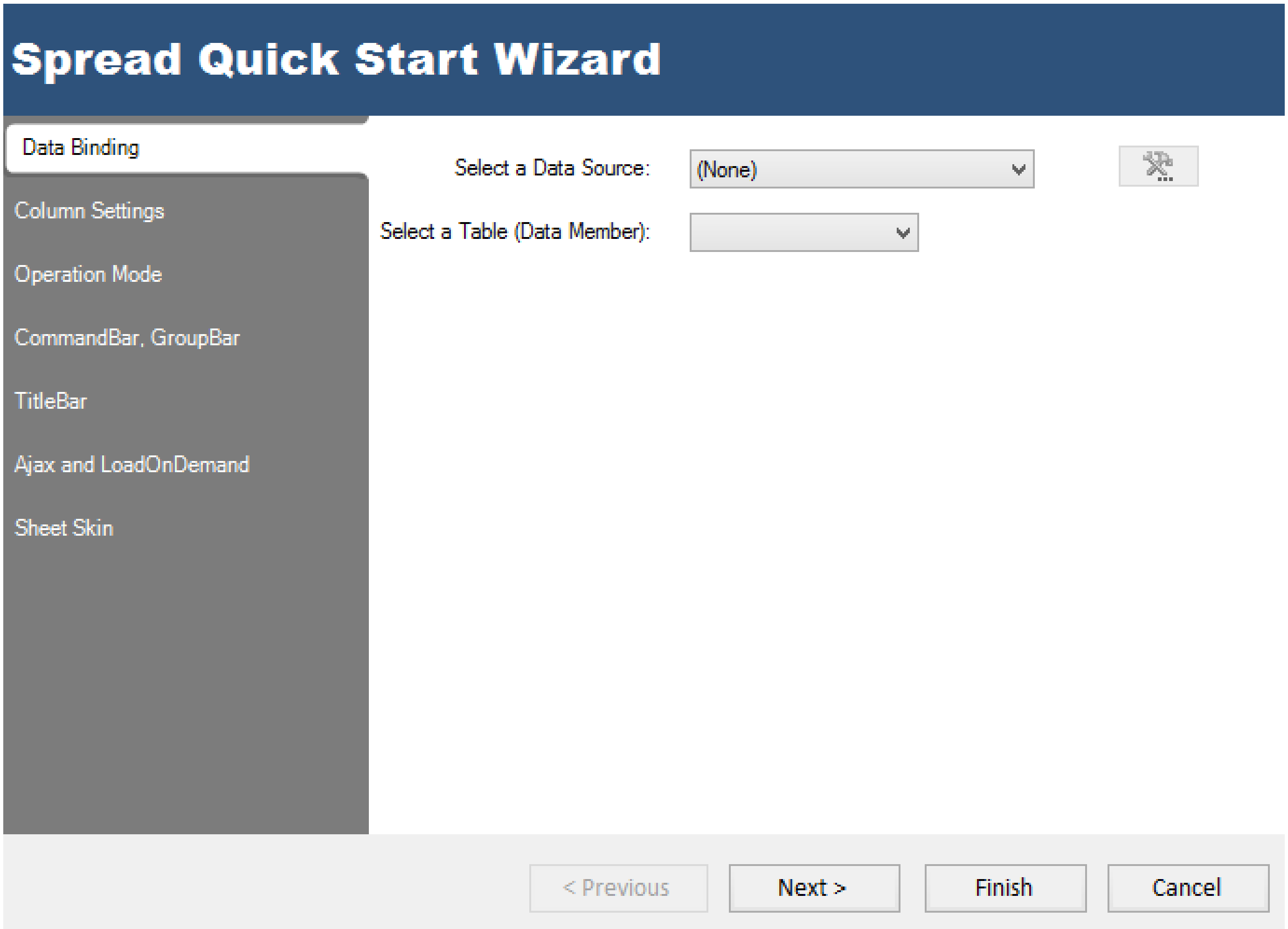
Version: 10_40_20172_0

Edit Templates

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 sheet skin, and many others.

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):

Tutorial: Creating a Checkbook Register

The following tutorial walks you through creating an ASP.NET project in Visual Studio .NET using the Spread for ASP.NET 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**
- **Adding Spread to a 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

Start a new Visual Studio .NET project. Name the project checkbook. Name the form in the project register.aspx. 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 Spread to a Project**.

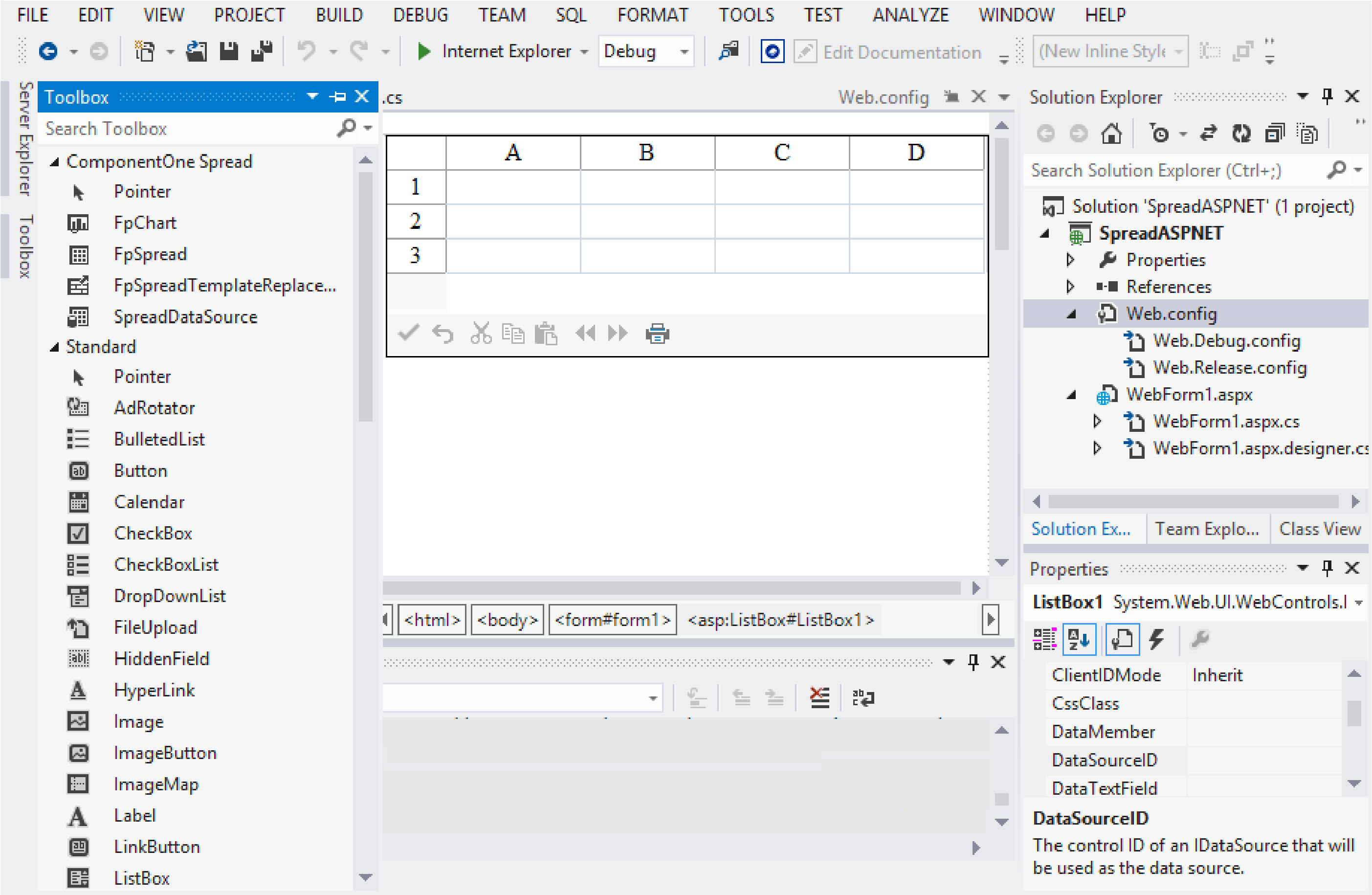
Adding Spread to a Project

This and all the tutorials assume that you have Visual Studio .NET installed on your system, and Internet Information Services (IIS) installed and running on your system as the local host server.

Perform the steps in this tutorial to set up an ASP.NET Visual Studio .NET project that contains the Spread for ASP.NET component.

1. Start Visual Studio .NET.
2. From the **File** menu, choose **New**, **Project**.
3. In the **New Project dialog**,
4. In the **Project Types** list, choose either **Visual Basic Projects** or **Visual C# Projects** depending on the language you are using.
5. In the **Templates** list, choose **ASP.NET Web Application**.
6. In the **Location** box, leave the location path as http://localhost/ unless you prefer to save this project to another server. Change the project name from WebApplication1 to the name of your choice.
7. Click **OK**.
8. In the **Solution Explorer**, right-click on the form name, WebForm1.aspx. Choose **Rename** from the pop-up menu, then type the new form name you prefer for the new form name.
9. If your project does not display the **Solution Explorer**, from the **View** menu, choose **Solution Explorer**.
10. If the **Toolbox** is not displayed, from the **View** menu choose **Toolbox**.
11. In the **Toolbox**, look in the **Web Forms** category (or in other categories if you have installed Spread and placed the toolbox icon in a different category). If the Spread component is not in the **Toolbox**, perform steps 12 through 14. Otherwise, proceed to step 15.
12. Right-click in the **Toolbox**, and from the pop-up menu choose Customize **Toolbox**.
13. In the **Customize Toolbox** dialog, click the **.NET Framework Components** tab.
14. In the **.NET Framework Components** tab,
    a. Click **Browse**.
    b. Browse to the installation path for the Spread for ASP.NET component. Once there, select FarPoint.Web.Spread.dll and click **Open**.
    c. The FpSpread component is now displayed in the list of components. Click **OK**.
15. In the **Toolbox** under **Web Forms** or another tab, select the FpSpread component.
16. On your form, draw an FpSpread component.
17. Save your project.

Your project should now look similar to the following image:



You have added the Spread component to the project.

Setting Up the Rows and Columns of the Register

The Spread component 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.

1. Double-click on the form in your project to open the code window.
2. Select the line of code

C#

```
// Put user code to initialize the page here.
```

VB

```
'Put user code to initialize the page here.
```

and type the following code to replace it:

C#

```
if (this.IsPostBack) return;
// Set up component and rows and columns in sheet.
FpSpread1.Height = Unit.Pixel(300);
FpSpread1.Width = Unit.Pixel(763);
FpSpread1.Sheets[0].ColumnCount = 8;
```

```
FpSpread1.Sheets[0].RowCount = 100;
```

VB

```
If (IsPostBack) Then
Return
End If
' Set up component and rows and columns in sheet.
FpSpread1.Height = Unit.Pixel(300)
FpSpread1.Width = Unit.Pixel(763)
FpSpread1.Sheets(0).ColumnCount = 8
FpSpread1.Sheets(0).RowCount = 100
```

This code sets up the component to be 300 pixels high and 763 pixels wide, and the sheet to have 8 columns and 100 rows.

3. Now we need to set up the columns to have custom headings. Add the following code below the code you added in Step 2:

   C#

```
// Add text to column heading.
FpSpread1.ColumnHeader.Cells[0, 0].Text = "Check #";
FpSpread1.ColumnHeader.Cells[0, 1].Text = "Date";
FpSpread1.ColumnHeader.Cells[0, 2].Text = "Description";
FpSpread1.ColumnHeader.Cells[0, 3].Text = "Tax?";
FpSpread1.ColumnHeader.Cells[0, 4].Text = "Cleared?";
FpSpread1.ColumnHeader.Cells[0, 5].Text = "Debit";
FpSpread1.ColumnHeader.Cells[0, 6].Text = "Credit";
FpSpread1.ColumnHeader.Cells[0, 7].Text = "Balance";
```

   VB

```
' Add text to column heading.
FpSpread1.ColumnHeader.Cells(0, 0).Text = "Check #"
FpSpread1.ColumnHeader.Cells(0, 1).Text = "Date"
FpSpread1.ColumnHeader.Cells(0, 2).Text = "Description"
FpSpread1.ColumnHeader.Cells(0, 3).Text = "Tax?"
FpSpread1.ColumnHeader.Cells(0, 4).Text = "Cleared?"
FpSpread1.ColumnHeader.Cells(0, 5).Text = "Debit"
FpSpread1.ColumnHeader.Cells(0, 6).Text = "Credit"
FpSpread1.ColumnHeader.Cells(0, 7).Text = "Balance"
```

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

   C#

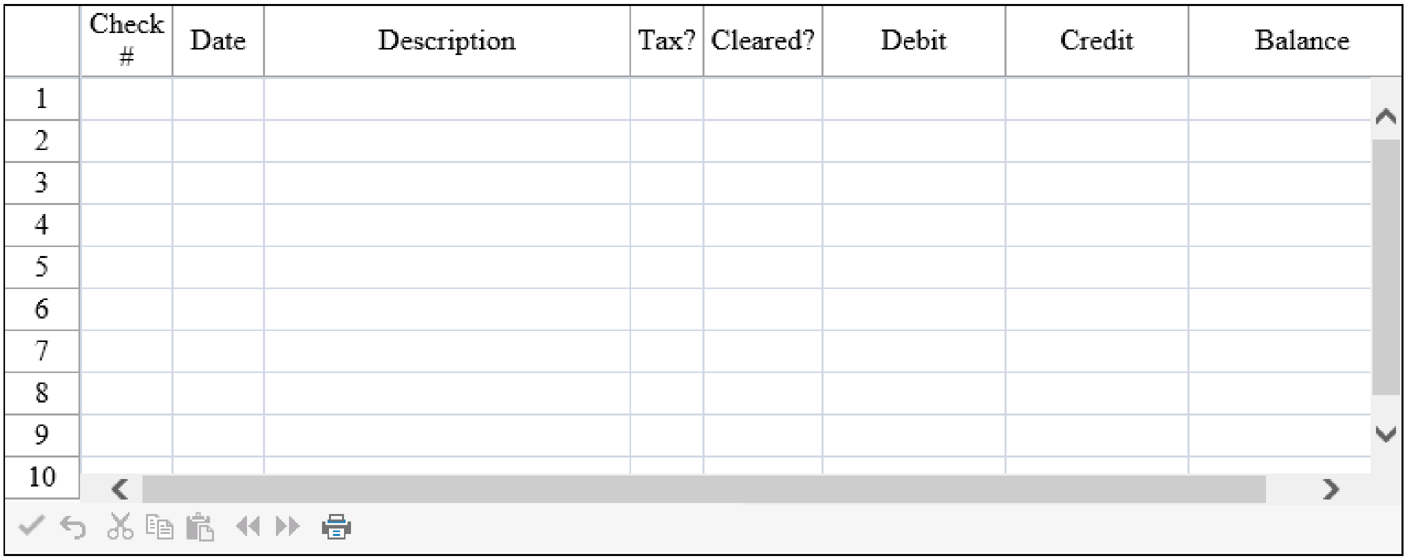
```
// Set column widths.
FpSpread1.Sheets[0].Columns[0].Width = 50;
FpSpread1.Sheets[0].Columns[1].Width = 50;
FpSpread1.Sheets[0].Columns[2].Width = 200;
FpSpread1.Sheets[0].Columns[3].Width = 40;
FpSpread1.Sheets[0].Columns[4].Width = 65;
FpSpread1.Sheets[0].Columns[5].Width = 100;
FpSpread1.Sheets[0].Columns[6].Width = 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 = 200
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 from the **Debug** menu choose **Start** to run your project.

Your ASP.NET page should look similar to the following picture.



	Check #	Date	Description	Tax?	Cleared?	Debit	Credit	Balance
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								

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:

   C#

```
// Create Check # column of integer cells.
FarPoint.Web.Spread.IntegerCellType objIntCell = new
FarPoint.Web.Spread.IntegerCellType();
FpSpread1.Sheets[0].Columns[0].CellType = objIntCell;
```

   VB

```
' Create Check # column of integer cells.
Dim objIntCell As New FarPoint.Web.Spread.IntegerCellType()
FpSpread1.Sheets(0).Columns(0).CellType = objIntCell
```

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

   C#

```
// Create Date column of date-time cells.
```

```
FarPoint.Web.Spread.DateTimeCellType objDateCell = new
FarPoint.Web.Spread.DateTimeCellType();
objDateCell.FormatString = "M/dd/yyyy";
FpSpread1.Sheets[0].Columns[1].CellType = objDateCell;
```

VB

```
' Create Date column of date-time cells.
Dim objDateCell As New FarPoint.Web.Spread.DateTimeCellType()
objDateCell.FormatString ="M/dd/yyyy"
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:

   C#

   ```
   // Create Description column of general cells.
   FarPoint.Web.Spread.GeneralCellType objGenCell = new
   FarPoint.Web.Spread.GeneralCellType();
   FpSpread1.Sheets[0].Columns[2].CellType = objGenCell;
   ```

   VB

   ```
   ' Create Description column of general cells.
   Dim objGenCell As New FarPoint.Web.Spread.GeneralCellType()
   FpSpread1.Sheets(0).Columns(2).CellType = objGenCell
   ```

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

   C#

   ```
   /// Create Tax? and Cleared? columns of check box cells.
   FarPoint.Web.Spread.CheckBoxCellType objCheckCell = new
   FarPoint.Web.Spread.CheckBoxCellType();
   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.Web.Spread.CheckBoxCellType()
   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:

   C#

   ```
   // Create the Debit, Credit, and Balance columns of currency cells.
   FarPoint.Web.Spread.CurrencyCellType objCurrCell = new
   FarPoint.Web.Spread.CurrencyCellType();
   FpSpread1.Sheets[0].Columns[5].CellType = objCurrCell;
   FpSpread1.Sheets[0].Columns[6].CellType = objCurrCell;
   FpSpread1.Sheets[0].Columns[7].CellType = objCurrCell;
   ```

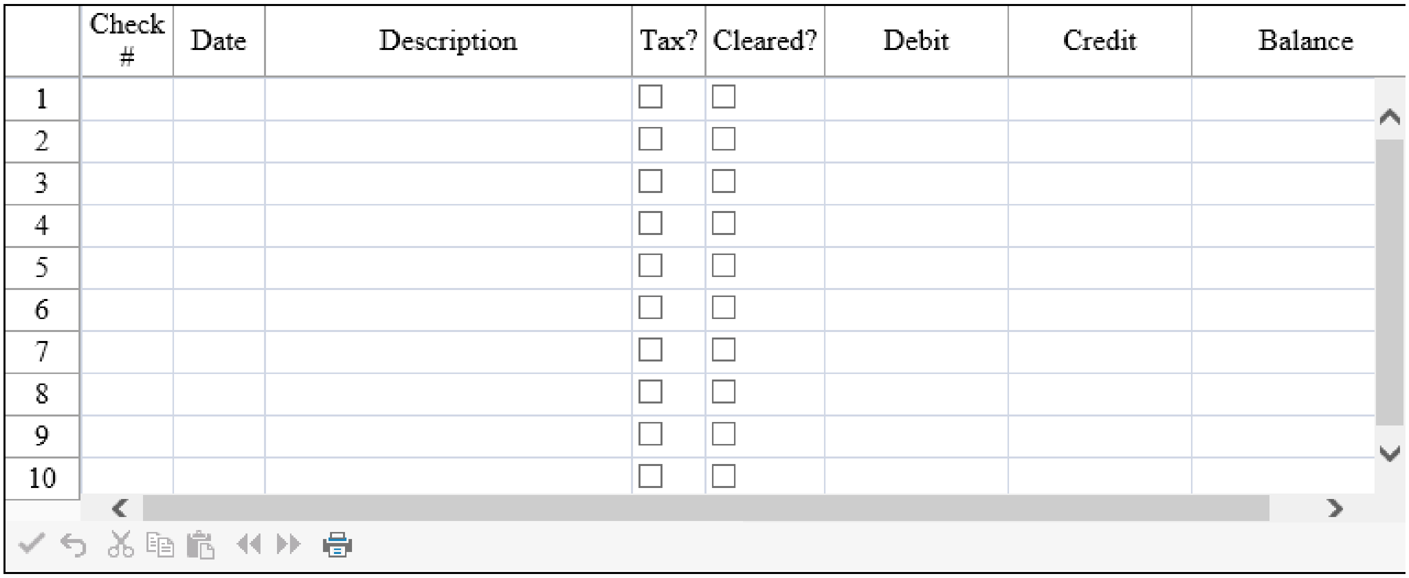
   VB

   ```
   ' Create the Debit, Credit, and Balance columns of currency cells.
   Dim objCurrCell As New FarPoint.Web.Spread.CurrencyCellType()
   ```

```
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 from the **Debug** menu choose **Start** to run your project.

Your ASP.NET page should look similar to the following picture.



	Check #	Date	Description	Tax?	Cleared?	Debit	Credit	Balance
1				□	□			
2				□	□			
3				□	□			
4				□	□			
5				□	□			
6				□	□			
7				□	□			
8				□	□			
9				□	□			
10				□	□			

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. Below the code you have already added, add the following code:

C#

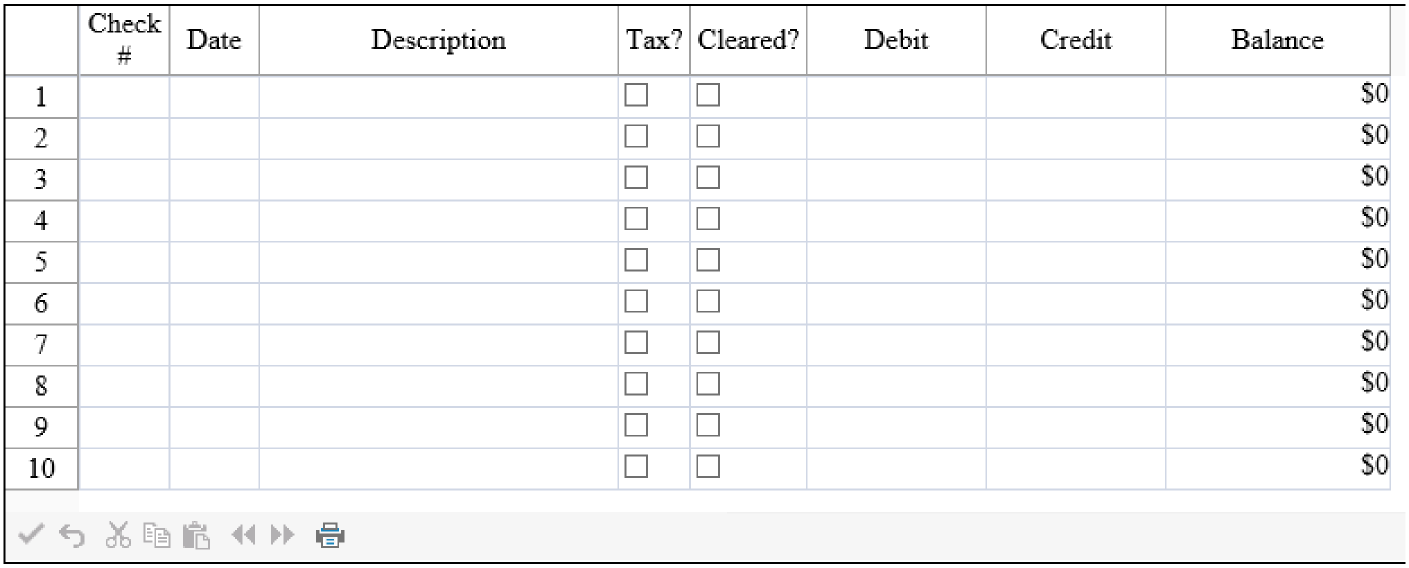
```
// Set formula for calculating balance.
FpSpread1.Sheets[0].ReferenceStyle =
FarPoint.Web.Spread.Model.ReferenceStyle.R1C1;
FpSpread1.Sheets[0].Cells[0, 7].Formula = "RC[-1]-RC[-2]";
for (int i = 1; i < 99; i++)
{
  FpSpread1.Sheets[0].Cells[i, 7].Formula = "R[-1]C-RC[-2]+RC[-1]";
}
```

VB

```
' Set formula for calculating balance.
FpSpread1.Sheets(0).ReferenceStyle = FarPoint.Web.Spread.Model.ReferenceStyle.R1C1
FpSpread1.Sheets(0).Cells(0, 7).Formula = "RC[-1]-RC[-2]"
Dim i As Integer
For i = 1 To 99
  FpSpread1.Sheets(0).Cells(i, 7).Formula = "R[-1]C-RC[-2]+RC[-1]"
Next
```

2. Save your project, then from the **Debug** menu choose **Start** to run your project.

Your ASP.NET page should look similar to the following picture. Type data into your checkbook register to test it and see how it operates. Click on the checkmark icon to save the changes or set the **ClientAutoCalculation** property for the control to **True**.



	Check #	Date	Description	Tax?	Cleared?	Debit	Credit	Balance
1				□	□			$0
2				□	□			$0
3				□	□			$0
4				□	□			$0
5				□	□			$0
6				□	□			$0
7				□	□			$0
8				□	□			$0
9				□	□			$0
10				□	□			$0

Your checkbook register is complete! You have completed this tutorial.

Understanding the Product

Spread for ASP.NET provides a completely new, object-oriented spreadsheet component for use in the Microsoft .NET framework. The following topics provide an introduction to this unique and powerful product and explain some of the underlying concepts.

- **Product Overview**
- **Features Overview**
- **Concepts Overview**
- **Namespaces Overview**

Product Overview

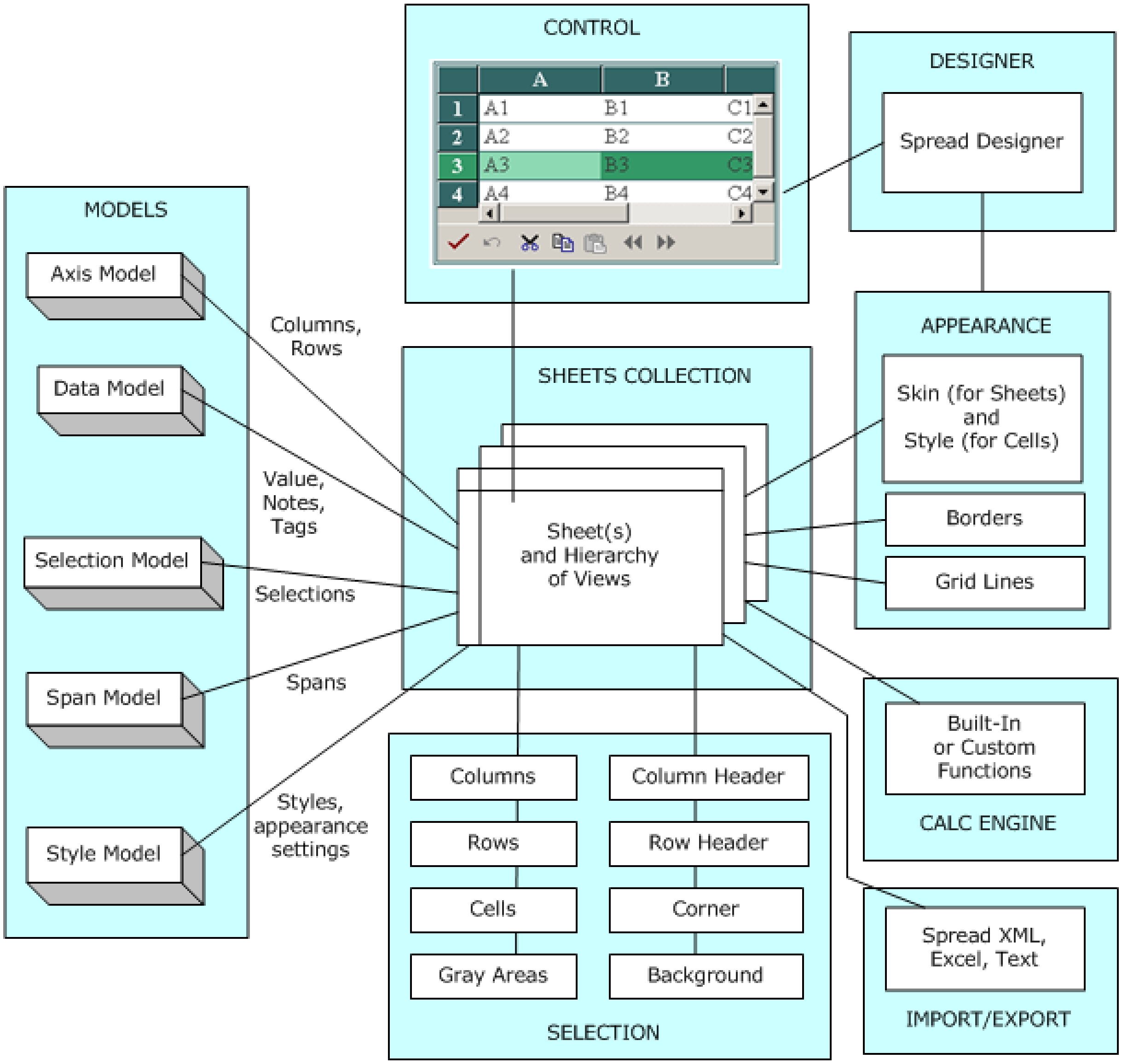
Spread for ASP.NET is a comprehensive spreadsheet component for Web applications that combines grid capabilities, spreadsheet functionality, and includes the ability to bind to data sources. A single component can contain many sheets, columns, and rows. Cross-sheet referencing allows calculations to make use of data and formulas on a variety of sheets. Spread for ASP.NET uses dot notation for object-oriented coding in .NET.

A Spread component may be dropped on a Web Form and customized for a range of applications. You can control 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.

The component resides on the server and generates HTML pages when it is accessed by end users on the client side. The amount of interactivity of the Web page depends on the level of browser used on the client side. The view of the HTML pages generated by the Spread component depends on the browser being used to view the page. Broadly, an uplevel browser is one that can support client-side JavaScript, HTML version 4.0, the Microsoft Document Object Model, and cascading stylesheets (CSS). For a more detailed definition of uplevel and downlevel browser and for a list of capabilities of those browsers, refer to the browser capability information in the Microsoft .NET documentation. The component also downloads some HTML component (HTC) files to the client side.

Import and export capabilities provide another source of flexibility when developing and exchanging designs. Spread for ASP.NET can handle data from comma-delimited text files as well as multiple spreadsheets from Microsoft Excel files.

The following figure provides a conceptual overview of Spread for ASP.NET.



In Spread, you can use the default models or extend them through inheritance. Refer to **Underlying Models** for more information on models. Styles and named styles provide ways to save customized appearances that can be applied to other sheets.

The Spread component contains toolbars and navigation aids, and a collection of sheets that contain row, column, cell, and header objects. The contents of the component may be saved as a BIFF8 or XLSX file compatible with Microsoft Excel or a text file or as a Spread XML file. For more information on exporting to (and importing from) a file, refer to the Spread for ASP.NET Import and Export Reference.

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

Features Overview

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

- **AJAX Support**
- **ASP.NET AJAX Extenders**

- **Built-In Functions**
- **Cell Types**
- **Chart Control**
- **Client-Side Scripting**
- **Conditional Formatting**
- **Context Menu**
- **Corner Customization**
- **Customized Appearance (Skins)**
- **Data Binding**
- **Floating Images (on-line documentation)**
- **Footers for Columns or Groups**
- **Formula Extender Control (on-line documentation)**
- **Frozen Rows and Columns**
- **Goal Seeking**
- **Grouping**
- **Headers with Multiple Columns and Rows**
- **Hierarchical Display**
- **Import and Export Capabilities**
- **Load on Demand**
- **Multiple-Line Columns**
- **Multiple Sheets**
- **PDF Support**
- **Printing**
- **Row Filtering**
- **Row Preview**
- **Row Template Editor**
- **Searching Features**
- **Sorting Capabilities**
- **Spannable Cells**
- **Sparklines**
- **Spread Designer**
- **Spread Wizard**
- **Theme Roller**
- **Title and Subtitle**
- **Touch Support**
- **Validation Controls**

AJAX Support

You can allow support for AJAX (Asynchronous JavaScript and XML) to make your applications more responsive on the client side.

For more information, refer to **Enabling AJAX Support**.

ASP.NET AJAX Extenders

You can use the many cell types in the FarPoint.Web.Spread.Extender assembly to provide controls that are available as ASP.NET AJAX extenders.

For more information, refer to **Using ASP.NET AJAX Extenders**.

Built-In Functions

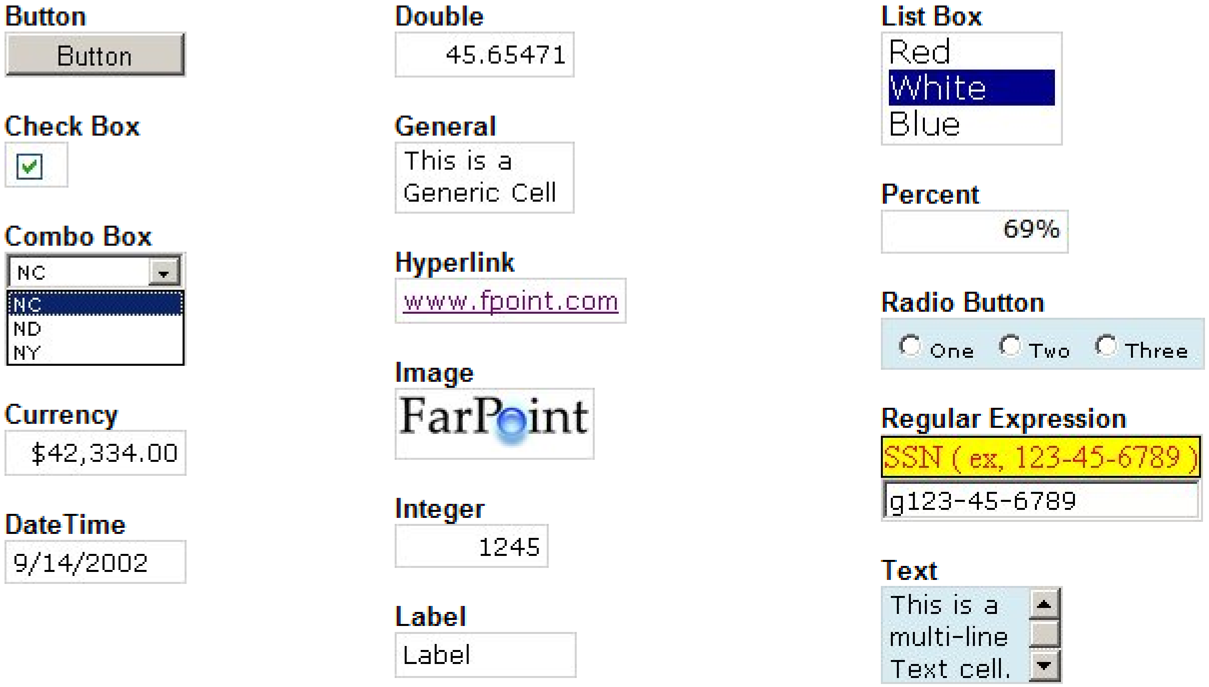
You can use built-in functions and operators to develop formulas and perform calculations. Add calculations quickly to your applications by using any of over 300 pre-defined algorithms or add your own custom functions.

For more information, refer to **Managing Formulas** and to the Formula Reference.

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.

The following image displays many of the cell types that are available. You can also create multi-column combo, tag cloud, and ajax extender cells.



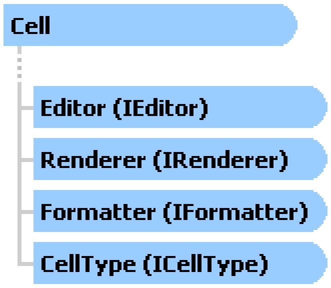
For a complete list of cell types, refer to **Customizing with Cell Types**.

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.

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 (on-line documentation)**. For more general information about cell types and applying them to cells, columns, rows, or whole sheets, refer to **Customizing with Cell Types**.

Chart Control

You can add a chart control to the Spread control or use the chart control stand alone. The chart control supports many types of charts as well as 2D or 3D views.

For more information, refer to **Working with the Chart Control**.

Client-Side Scripting

With scripting you can run small programs, or scripts, in your Web pages on the client after the pages are downloaded from the server. Client-side scripting allows the user to interact with the page on the client without the page having to be reloaded from the server. Scripts can be written in any scripting language supported by the browser on the client, such as VBScript or JavaScript. Most often scripts are written in JavaScript (also referred to as ECMAScript).

Spread for ASP.NET gives you client-side scripting capabilities to enhance the power of your Web Forms. In addition, you can also create your own HTC files for specific purposes.

Spread for ASP.NET client side features are enabled for IE 5.5 and above. For other browsers, there may be no client side scripting support or it may be limited.

For more information, refer to the Spread for ASP.NET **Client-Side Scripting Reference (on-line documentation)**.

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 or rule.

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

Context Menu

You can create a Spread context menu that is displayed when right-clicking on the Spread control. For more information, see **Adding a Context Menu**.

Corner Customization

You can customize the sheet corner with text, colors, and various style settings.

For more information, refer to **Customizing the Sheet Corner**.

Customized Appearance (Skins)

Easily and quickly configure the appearance of Spread using predefined skins or create and save your own custom skins. Custom skins can be shared with everyone in your development team, allowing a consistent look of the component across applications.

For more information about skins, refer to **Applying a Skin to a Sheet** and **Creating a Skin for Sheets**.

Data Binding

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 bind a range of cells to a data source.

For more information about data binding, refer to **Managing Data Binding**.

Footers for Columns or Groups

You can display a footer at the bottom of the sheet that allows you to enter formulas or instructions with a footer cell for each column. This feature also works with the grouping feature.

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

Frozen Rows and Columns

You can freeze columns and rows (keep them non-scrollable) and keep them displayed regardless of where the user navigates in the sheet.

For more information, refer to **Freezing Rows and Columns**.

Goal Seeking

You can use goal seeking capability to iterate toward a desired formula result. Use this if you know the result of a formula but not the input value required to obtain the result.

For more information, refer to **Finding a Value with Goal Seeking**.

Grouping

You can set up the spreadsheet to group rows of data. This is useful for displaying large amounts of data in organized groups.

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

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 data.

For more information, refer to **Customizing the Appearance of Headers**.

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 **Displaying Data as a Hierarchy**.

Import and Export Capabilities

You can import data from and export data and formatting to Microsoft Excel, both individual spreadsheets and entire workbooks. You can import and export entire spreadsheet(s) with data and formatting to and from XML.

For more information about opening and saving files, refer to **Managing File Operations**.

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

Load on Demand

You can allow the Web page to load on demand. As the user scrolls further down the spreadsheet the Spread component on the client loads another page of rows from the server as needed.

- FpSpread.AllowLoadOnDemand
- SheetView.AllowLoadOnDemand

For more information about this feature, refer to **Allowing Load on Demand**.

Multiple-Line Columns

For a more compressed format of displaying information, you may want to display columns with multiple lines of information.

For more information on multiple-line columns, refer to **Creating Row Templates (Multiple-Line Columns)**.

Multiple Sheets

Spread 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 multiple rows and columns. You can define styles for sheets and apply those styles across multiple sheets.

For more information about sheets, refer to **Working with Multiple Sheets**.

PDF Support

You can allow the user to save the spreadsheet to a PDF (Portable Document Format) file.

For more information, refer to **Saving to a PDF File**.

Printing

You can allow the user to print. You can also add headers and footers to the printed pages and customize the printed page.

For more information, refer to **Managing Printing**.

Row Filtering

You can allow row filtering by hiding or changing the color of the filtered rows. You can also use a filter bar, simple filtering, or enhanced filtering.

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

Row Preview

You can add a preview row that contains extra information about a row.

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

Row Template Editor

You can use the row editor or template editor to type text.

For more information, refer to **Setting up Row Edit Templates**.

Searching Features

You can programmatically search the cell text, headers, notes, or tags. You can also specify the starting row and column and the ending row and column.

Use the **Search ('Search Method' in the on-line documentation)** method or the **SearchHeaders ('SearchHeaders Method' in the on-line documentation)** method of the **FpSpread ('FpSpread Class' in the on-line documentation)** object. For more information, refer to **Searching for Data with Code**.

Sorting Capabilities

You can sort rows or columns or a range of cells. You can programmatically sort data by rows or columns or allow your users to sort rows automatically by clicking on the column header.

For more information, refer to **Customizing 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.

For more information, refer to **Spanning Cells**.

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 component at design time using an intuitive, easy-to-use interface.

For more information refer to **Working with the Spread Designer**.

Spread Wizard

You can use the Spread Wizard to design your component and to create a prototype quickly.

For more information refer to **Using the Spread Wizard**.

Theme Roller

Spread supports using a Theme Roller theme in the Spread control.

For more information, refer to **Using the jQuery Theme Roller with Spread**.

Title and Subtitle

You can add a title to the Spread component and subtitle to the sheet.

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

Touch Support

The Spread control supports touch gestures in many areas.

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

Validation Controls

You can prevent a user from entering invalid characters in a cell by using a validation control in the Spread cell.

For more information, refer to **Using Validation Controls**.

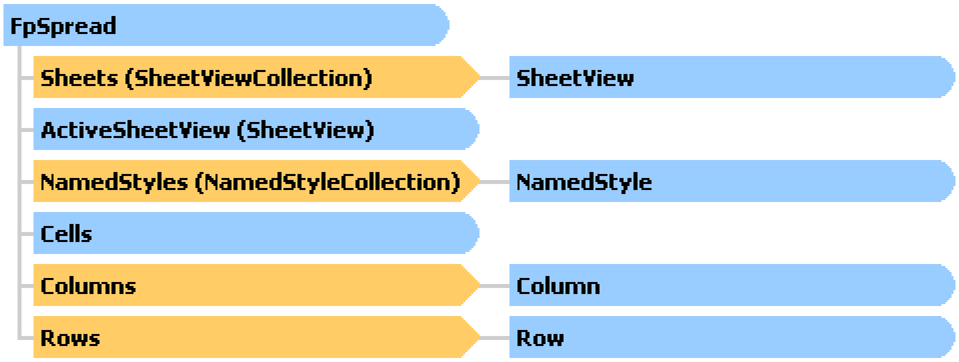
Concepts Overview

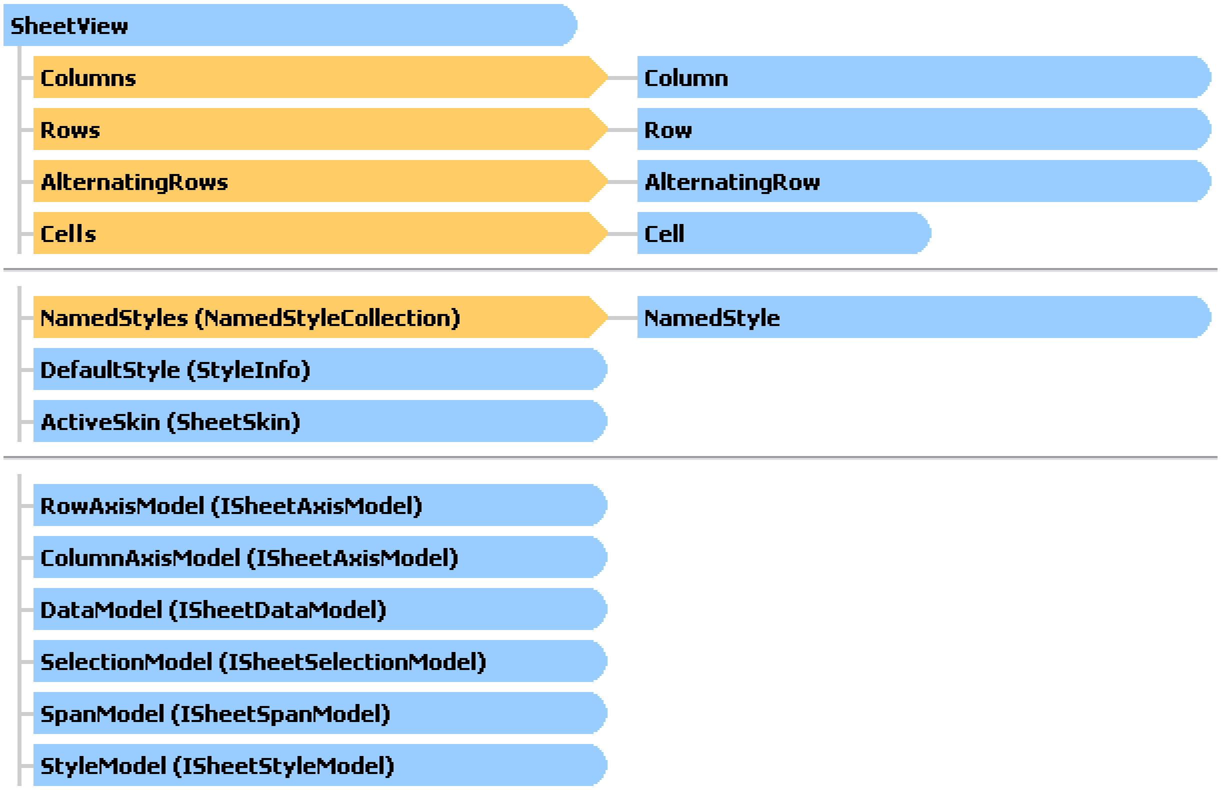
While those familiar with previous generations of Spread products understand the object oriented nature of Spread for ASP.NET, there are several concepts that are worth reviewing for new users.

- **Shortcut Objects**
- **Object Parentage**
- **Underlying Models**
- **Cell Types**
- **SheetView versus FpSpread**
- **Formatted versus Unformatted Data**
- **Zero-Based Indexing**
- **Client-Side Scripting**
- **Maintaining State**

Shortcut Objects

The spreadsheet objects in the Spread namespace, which represent various parts of the spreadsheet, can be accessed through a set of built-in shortcut objects. The shortcut objects help you interact with the parts of the spreadsheet in a way that is probably familiar to you from working with other components or applications. Cells, rows, columns and others are wrappers to other objects, and make customization that much easier by allowing you to manipulate them. There are objects that 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
Sheets (SheetViewCollection)
SheetView
ActiveSheetView (SheetView)
NamedStyles (NamedStyleCollection)
NamedStyle
Cells
Columns
Column
Rows
Row


SheetView
Columns
Column
Rows
Row
AlternatingRows
AlternatingRow
Cells
Cell
NamedStyles (NamedStyleCollection)
NamedStyle
DefaultStyle (StyleInfo)
ActiveSkin (SheetSkin)
RowAxisModel (ISheetAxisModel)
ColumnAxisModel (ISheetAxisModel)
DataModel (ISheetDataModel)
SelectionModel (ISheetSelectionModel)
SpanModel (ISheetSpanModel)
StyleModel (ISheetStyleModel)

Spread for ASP.NET provides the following shortcut objects in the Spread namespace:

Shortcut Object	Corresponding Classes	
cell	**Cell ('Cell Class' in the on-line**	**Cells ('Cells Class' in the on-line**

	documentation)	**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)**

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

The shortcut objects help you interact with the Spread for ASP.NET component in a way that is probably familiar to you from working with other components or applications. They are shortcuts for working with more conceptual objects, the more abstract objects, that are referred to as "models." These models are responsible for managing the style information, formatting, and data in the Spread component. They are what give the product its power and flexibility for customization. For more information on the underlying models, refer to **Underlying Models**.

The shortcut objects call the model objects. However, the shortcut objects allow you to interact with the Spread component without dealing too much with the underlying object models if you are doing routine development. 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 the features of Spread for ASP.NET. However, intensive use of the shortcut objects can degrade your application’s performance. As you get familiar with the workings of the product and as you want more control over the spreadsheet, you may begin to work more with the models.

For more information on the skins that can applied to a sheet, refer to **Applying a Skin to a Sheet**. For more information on the styles refer to **Creating and Applying a Custom Style for Cells**.

Object Parentage

For the objects in a 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**. For information on cell types, which is set in a different way than inheriting from a parent, refer to **Customizing with Cell Types**.

Underlying Models

Spread for ASP.NET provides the following underlying models for each sheet and each set of headers in the spreadsheet:

Sheet Model	Classes and Interface	Description
Axis model	**BaseSheetAxisModel ('BaseSheetAxisModel Class' in the on-line documentation)** **DefaultSheetAxisModel ('DefaultSheetAxisModel Class' in the on-line documentation)** **ISheetAxisModel ('ISheetAxisModel Interface' in the on-line documentation)**	Basis for how the sheet of cells is structured in terms of rows and columns.
Data model	**BaseSheetDataModel ('BaseSheetDataModel Class' in the on-line documentation)** **DefaultSheetDataModel ('DefaultSheetDataModel Class' in the on-line documentation)** **ISheetDataModel ('ISheetDataModel Interface' in the on-line documentation)**	Basis for the manipulation of data in the cells in the sheet.
Selection model	**BaseSheetSelectionModel ('BaseSheetSelectionModel Class' in the on-line documentation)** **DefaultSheetSelectionModel ('DefaultSheetSelectionModel Class' in the on-line documentation)** **ISheetSelectionModel ('ISheetSelectionModel Interface' in the on-line documentation)**	Basis for the behavior of and interaction of selected cells in the sheet.
Span model	**BaseSheetSpanModel ('BaseSheetSpanModel Class' in the on-line documentation)** **DefaultSheetSpanModel ('DefaultSheetSpanModel Class' in the on-line documentation)** **ISheetSpanModel ('ISheetSpanModel Interface' in the on-line documentation)**	Basis for how cells in the sheet are spanned.
Style model	**BaseSheetStyleModel ('BaseSheetStyleModel Class' in the on-line documentation)** **DefaultSheetStyleModel ('DefaultSheetStyleModel Class' in the on-line documentation)** **ISheetStyleModel ('ISheetStyleModel Interface' in the on-line documentation)**	Basis for the appearance of the cells in the sheet.

Each model object is provided as a base model and a default model. The base model is the base on which the default model is created. The base model has the fewest built-in features, and the default model extends the base model.

The default models are provided as the models with which you will most likely want to work in Spread. They provide the default features that the Spread component offers.

However, if you want to provide different features, you might want to extend the base models yourself, creating new classes. One reason you might do this is if you want to create a "template" component for all the developers in your organization to use. By creating your own class based on one of the Spread base models, you could provide such a template.

The shortcut objects access the interfaces in the model namespace. When you work with shortcut objects, you are actually working with the interfaces of the models in the component. For example, if you change the background color of a cell using the Cells shortcut object, the interface for the sheet style model is updated with that information. For more information on the shortcut objects, refer to **Shortcut Objects**.

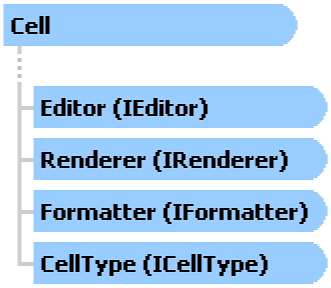
For more in-depth information on the models, refer to **Using Sheet Models**.

Cell Types

There are several different cell types that can be set for cells in a sheet. 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.

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.

A cell has both an editor and a renderer. The editor is an actual control instance that we create and place 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 (on-line documentation)**. For more general information about cell types and applying them to cells, columns, rows, or whole sheets, refer to **Customizing with Cell Types**.

SheetView versus FpSpread

In Spread, there are several objects that should not be confused. The sheet in a multiple-sheet workbook corresponds to a SheetView object. In a hierarchical data display, each child (expanded) sheet of the parent sheet corresponds to a separate SheetView object. The FpSpread object is somewhat like a workbook.

For more detailed information on these objects, refer to the FarPoint.Web.Spread.**SheetView ('SheetView Class' in the on-line documentation)** object and the FarPoint.Web.Spread.**FpSpread ('FpSpread Class' in the on-line documentation)** object in the API reference documentation.

Formatted versus Unformatted Data

Spread for ASP.NET 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 ('Text Property' in the on-line documentation)** property would return the entire formatted string with currency symbol and thousand separator. The **Value ('Value Property' in the on-line documentation)** property could be used in formulas or other calculations. Remember that each cell has both properties. For cell types that have buttons or check boxes, the distinction is also important.

For more detailed information on the text and value data for specific cell types in Spread for ASP.NET, refer to the

FarPoint.Web.Spread assembly and namespace, the **Cell ('Cell Class' in the on-line documentation)** class, the **Text ('Text Property' in the on-line documentation)** and **Value ('Value Property' in the on-line documentation)** properties, and **Understanding How Cell Types Display Data**. Refer to **Placing and Retrieving Data** for more information about data in cells.

Zero-Based Indexing

For most objects in Spread, zero-based indexing is used. Rows and columns are numbered starting with zero (0, 1, 2, and so on). Note that the default header labels are numbered starting with one (1, 2, 3, and so on).

Client-Side Scripting

Scripting refers to running small programs, or scripts, in your pages. Scripts are written in a scripting language, such as VB Script or JavaScript. For more information on how to perform client-side scripting, refer to the Spread for ASP.NET **Client-Side Scripting Reference (on-line documentation)**.

Maintaining State

You can and should maintain the session state when the page is refreshed, so that user data remains in the page. You have probably experienced pages that do not maintain state; when the page is refreshed, such as to remind you to complete part of a form, your information is lost, and you must complete the entire page again. Understandably, users prefer pages that maintain the state.

You need to set up your application’s state management to optimize performance while maintaining the state. For more information about the best ways to optimize performance, refer to **Maintaining State**.

Namespaces Overview

In Spread for ASP.NET, namespaces are organized to contain objects according to how they are used in the Spread component. The Spread component comprises six namespaces. The objects in the component fall into these categories:

- objects that represent parts of the spreadsheet, like column, rows, and cells
- objects that represent parts of the component for editing cells in the spreadsheet
- objects that represent parts of the component for the underlying template or model of the spreadsheet
- objects that represent parts of the component for rendering or displaying the spreadsheet
- objects that represent the chart control
- objects that represent data binding

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

Namespace	Description
FarPoint.Web.Spread	Provides the base classes, interfaces, enumerations, and delegates for Spread.
FarPoint.Web.Spread.Chart	Provides the base classes, interfaces, and enumerations for the Spread Chart component.
FarPoint.Web.Spread.Data	Provides the base classes and interfaces for data binding for the Spread spreadsheet component.
FarPoint.Web.Spread.Editor	Provides the base classes and interface for the controls used to edit cells.
FarPoint.Web.Spread.Model	Provides the base classes, interfaces, and enumerations for the Spread models.
FarPoint.Web.Spread.Renderer	Provides the base classes and interfaces for the controls used to render cells.

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

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. 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.

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 for ASP.NET, you must use the model objects to do so.

Working with the Spread Designer

You can quickly customize a spreadsheet component using the Spread Designer. Whether you are prototyping a complete spreadsheet component or simply customizing some aspect of your spreadsheet component, the dedicated graphical interface offers many features to save time and effort. It also provides a way for you to add data to and set properties for the component easily, including properties that are not available at design time in Visual Studio. You can set both design-time and run-time properties. In most cases you can preview changes before applying them to the spreadsheet.

The Spread Designer requires Microsoft Internet Explorer (IE) 7 or higher.

The Spread Designer creates a snapshot of the spreadsheet component. Once all the changes are made, you apply the changes to the spreadsheet component on your form. You can also open files from within Spread Designer and save your design as a file.

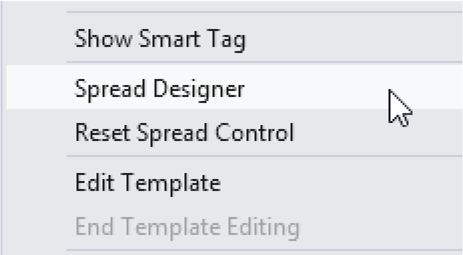
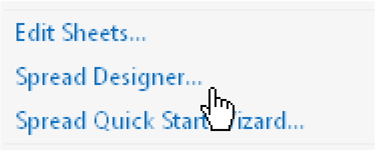
Throughout the rest of the documentation, where there is a procedure that could be done in code or in the Spread Designer, a brief procedure for using Spread Designer is given. Because Spread Designer has so many features, a topic dedicated to the designer is given here. This topic provides information about general tasks specific to the designer and about the designer user interface in general. It is not a comprehensive explanation of all the dialogs within the Spread Designer.

The following topics provide information about using Spread Designer:

- **Starting the Spread Designer**
- **Understanding the Spread Designer Interface**
- **Using the Spread Designer**

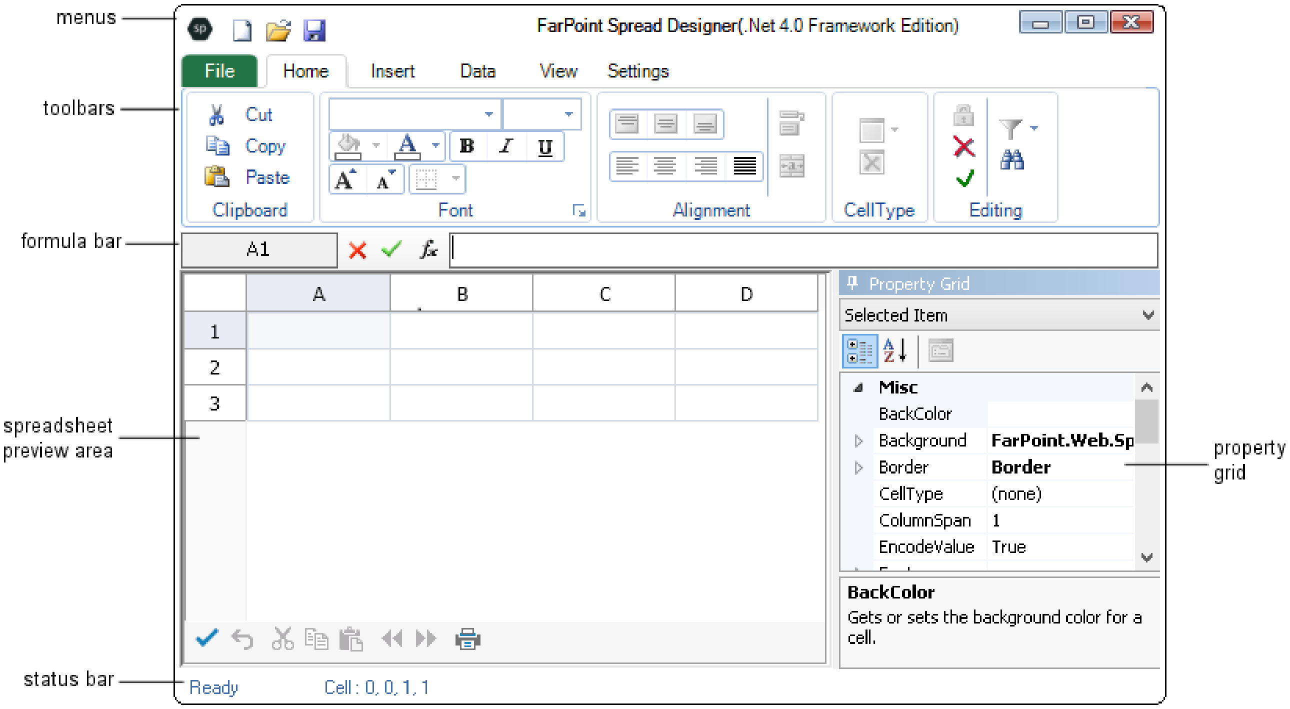
Starting the Spread Designer

You can start Spread Designer from inside your Visual Studio .NET project by performing either of these steps: right-clicking on the control and selecting the context menu or selecting the designer verb area of the Smart Tags, both shown below.

Description	Picture
Context (right-click) menu of the FpSpread component selected on the form in Visual Studio	 Show Smart Tag Spread Designer Reset Spread Control Edit Template End Template Editing
Verb area of the Smart Tags (that shows up after clicking on the arrow to the right) of the FpSpread component selected on the design form in Visual Studio	 Edit Sheets... Spread Designer... Spread Quick Start Wizard...

Understanding the Spread Designer Interface

Use the Spread Designer as a way to quickly set properties of the FpSpread component in the Microsoft .NET framework by accessing the properties in an organized and easy-to-use interface. There are several places where the Spread Designer offers additional capabilities beyond the **Properties** window in the .NET framework. The parts of the user interface of the Spread Designer are shown in this figure.



BackColor	
Background	FarPoint.Web.Sp
Border	Border
CellType	(none)
ColumnSpan	1
EncodeValue	True

BackColor
Gets or sets the background color for a cell.

Ready Cell : 0, 0, 1, 1

To set the properties for a part of the spreadsheet, select the item in the **Property Grid** (Spread, Sheet, etc.) that corresponds with that item. The figure above shows what appears when you click on the Spread item. The properties related to this item are displayed in the **Property Grid** where you can set them as you would in Visual Studio. You can also use the menus and toolbars to make changes to properties quickly and easily. The preview area offers a quick visual indication of the results of your changes.

In this example, the Spread menu includes those properties of the overall component, including the border or outline of the entire component, the command bar, the page navigation bar, the scroll bars, and many other design time properties. You can also add styles (that apply to an entire Spread or an individual sheet) and you can add sheets. The drop-down combo box allows you to select the **Spread**, **Sheet**, or **Selected Item** menus. You can then set properties for the component, a specific sheet, or a selected item such as a column or a row.

The Spread Designer offers a quick and easy way to change appearance and functionality of an FpSpread component at design time. Remember that this changes properties at design time in the component itself and no code is added for these changes. Some properties cannot be seen at design time and the effect is not seen until you preview the spreadsheet or until you run the spreadsheet.

The Spread Designer has several areas where you can change designer settings or change properties of the Spread component. The following topics describe the different areas:

- **Spread Designer Menus**
- **Spread Designer Toolbars**
- **Spread Designer Editors**
- **Spread Designer Context Menus**

Spread Designer Menus

You can use the designer menus to save or load a designer file, show or hide toolbars, edit, set various formatting properties for the component, or bring up the help.

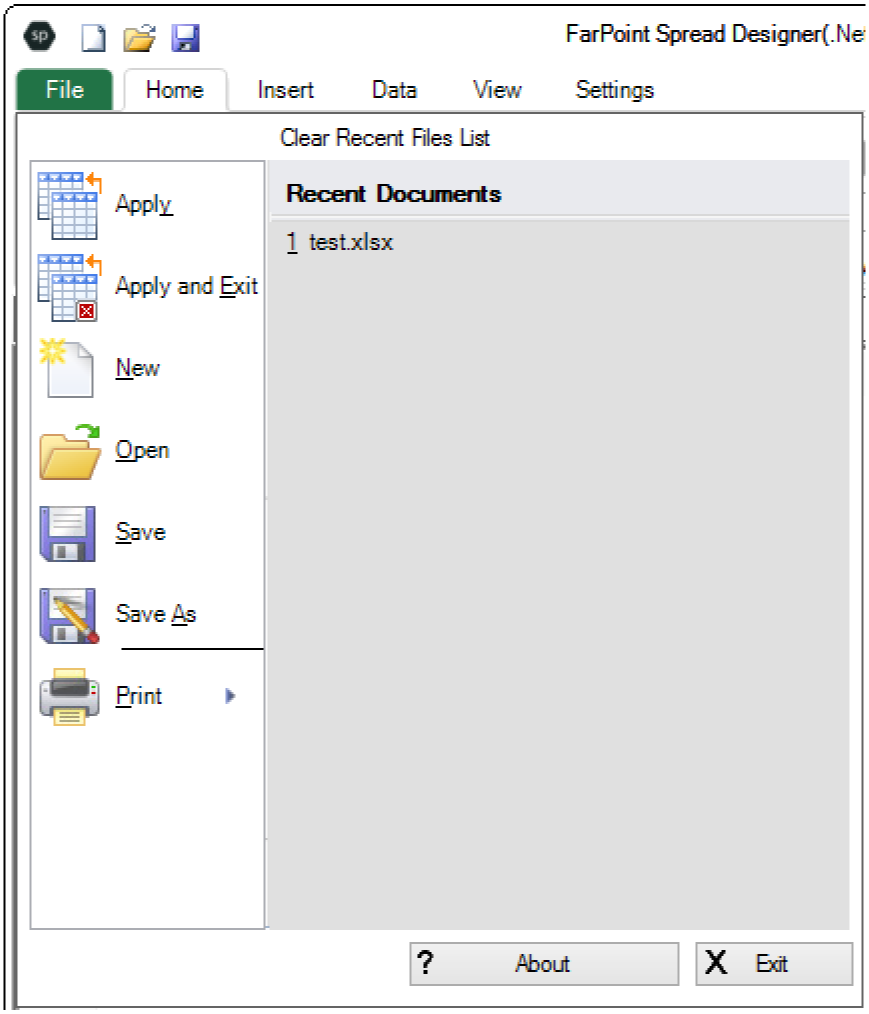
The following menus are available:

- **File Menu**
- **Home Menu**
- **Insert Menu**
- **Data Menu**
- **View Menu**
- **Settings Menu**
- **Chart Tools Menu**
- **Sparklines Menu**

For more tasks within Spread Designer, return to **Understanding the Spread Designer Interface**.

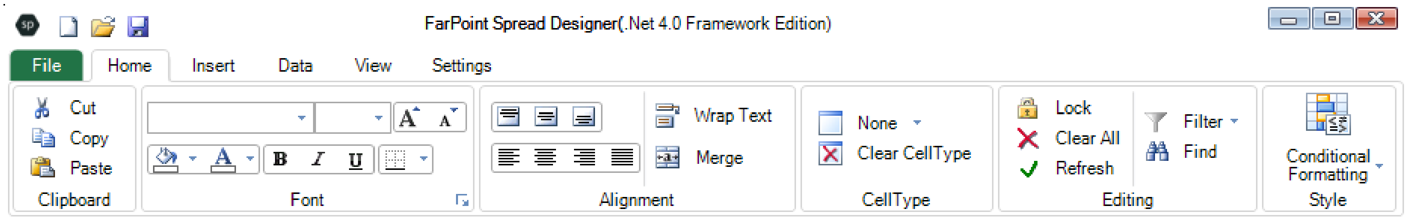
File Menu

The **File** menu or **Spread** button can be used to open and save files, apply changes to the designer or apply and exit, reset the settings, print, preview the printing, various save options, or exit the Designer. The **File** menu is shown in the following figure:



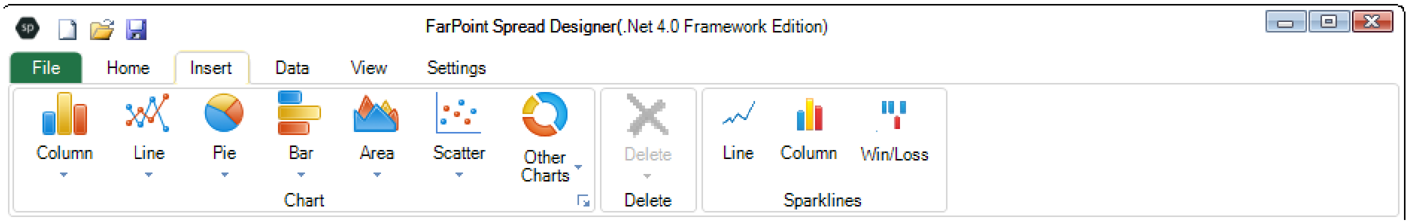
Home Menu

The **Home** menu can be used to cut, copy, or paste cell data, select cells, set cell fonts and alignment, lock cells, set cell types, find text or formatting, set conditional formatting, and clear or refresh the control. The **Home** menu is shown in the following figure:



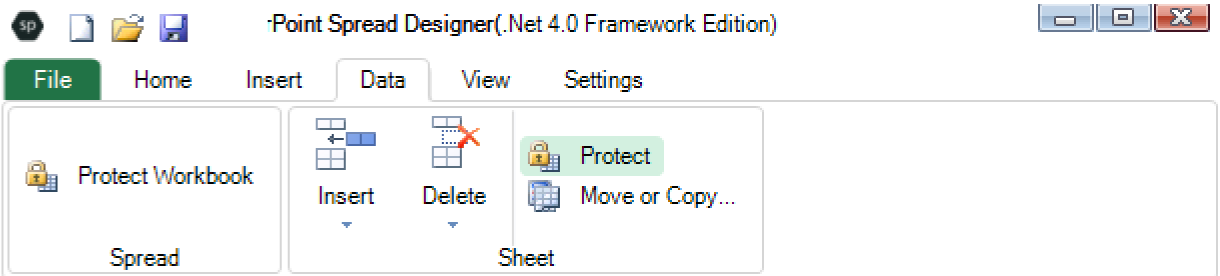
Insert Menu

The **Insert** menu can be used to add or delete a chart control or a sparkline. The **Insert** menu is shown in the following figure:



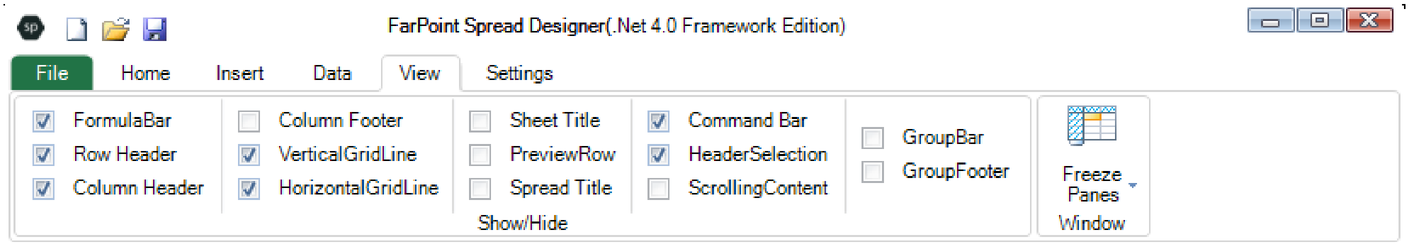
Data Menu

The **Data** menu can be used to insert or delete rows, columns, or sheets. You can also use it to protect, move, or copy a sheet. The Protect Workbook option allows you to protect the structure of the workbook (changes such as moving or adding sheets). The **Data** menu is shown in the following figure:



View Menu

The **View** menu can be used to show or hide headers, grid lines, the formula bar, column footers, the group bar, group footers, the header selection, scrolling content, Spread or sheet titles, or the preview row. The View menu can also be used to freeze columns or rows. The **View** menu is shown in the following figure:



Settings Menu

The **Settings** menu can be used to format cells, the sheet, or the Spread component. The **Settings** menu has a **Spread Setting** section that contains options for creating titles, setting the focus rectangle color, setting edit properties, paging, command bar options, scrollbar options, and tab settings.

The **Sheet Setting** section can be used to set the column and row count, freeze columns and rows, set paging and sorting options, set operation mode, set the locked and selection colors, set header and footer properties (including the sheet corner), set grid lines (type and color), and set formula properties such as reference style.

The **Appearance Settings** section can be used to create sheet skins and named styles.

The **Other Settings** section has editors for the row template, header and groups, alternating rows, and cells, columns, and rows.

The **Designer Settings** section allows you to make changes to the designer such as showing on start up and resizing.

The **Settings** menu is shown in the following figure:

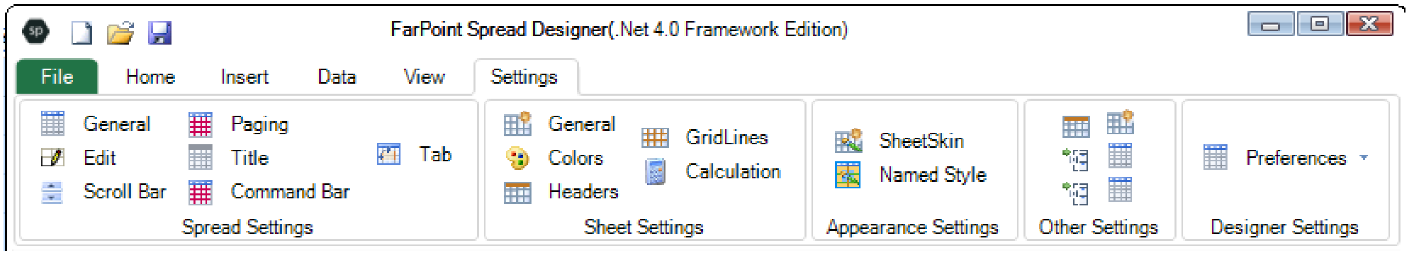
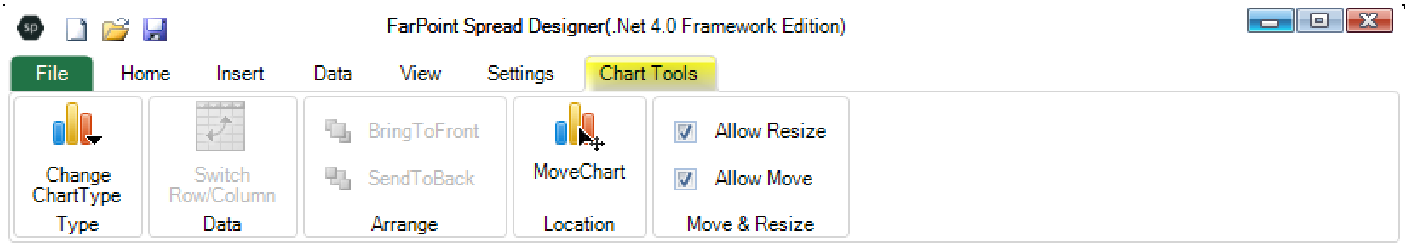


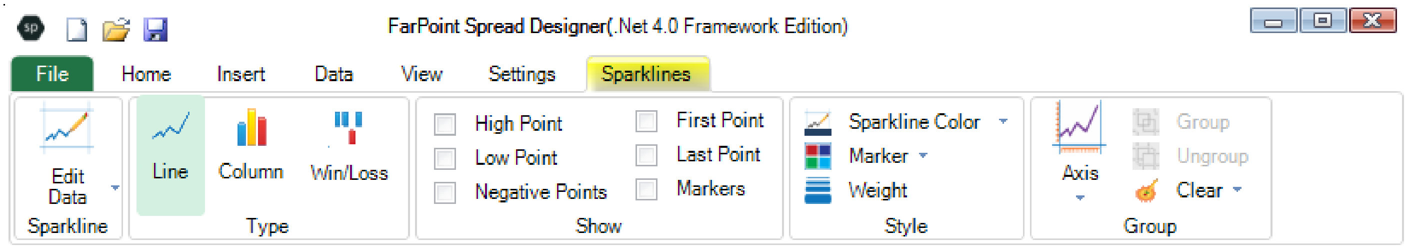
Chart Tools Menu

The **Chart Tools** menu can be used to make changes to a chart control. You can change the chart type, switch the column and row data, move to the front or back, move the chart, or allow the user to move or resize the chart. The **Chart Tools** menu is shown in the following figure:



Sparklines Menu

The **Sparklines** menu can be used to make changes to a sparkline. This menu is displayed after you add a sparkline to a cell. The **Sparklines** menu is shown in the following figure:



The **Sparklines** menu can be used to edit an existing sparkline, display markers and specific points, set colors and weights, show an axis, and group and ungroup sparklines.

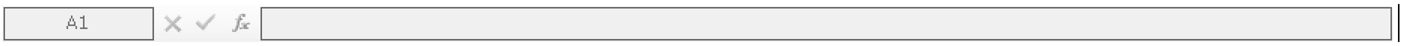
Spread Designer Toolbars

You can use the toolbars to provide quick access to some features of the Spread Designer such as saving or loading a file and creating formulas.

The following toolbars are available:

Formula Toolbar

The **Formula** toolbar allows you to enter formulas directly or open the **Formula Editor** to specify a formula for a cell or range of cells. This toolbar is shown in the following figure.



General Toolbar

The **General** toolbar can be used to open and save files. This toolbar is shown in the following figure (next to the file button icon):

For more tasks within Spread Designer, return to **Understanding the Spread Designer Interface**.

Spread Designer Editors

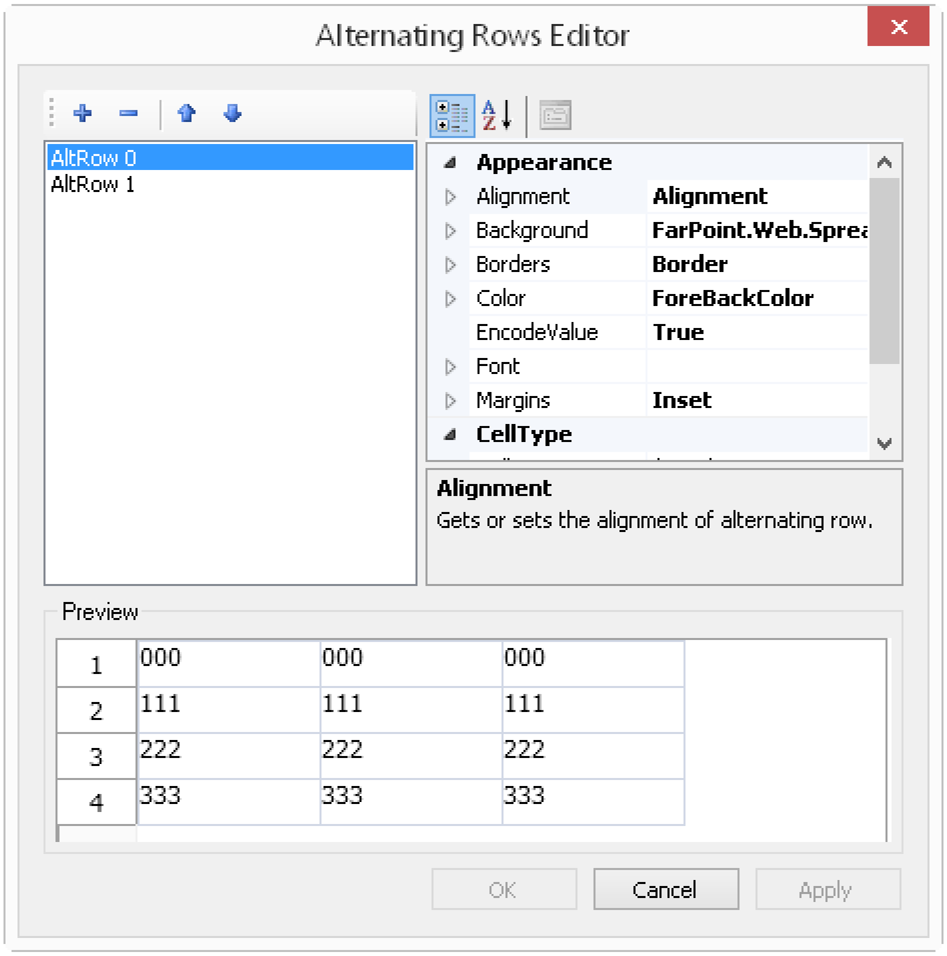
You can use the collection editors to format specific areas of the component such as groups, sheet views, and named styles. The following editors are available:

- **Alternating Rows Editor**
- **Cells, Columns, and Rows Editor**
- **ContextMenu Collection Editor**
- **DataKey Names (String Collection) Editor**
- **Formula Editor**
- **GroupInfo Collection Editor**
- **Header Editor**
- **NamedStyle Collection Editor**
- **Row Template Editor**
- **SheetSkin Editor**
- **SheetView Collection Editor**

For more tasks within Spread Designer, return to **Understanding the Spread Designer Interface**.

Alternating Rows Editor

You can customize alternating rows. You can specify borders, colors, cell types, and other options with the **Alternating Rows** editor in the Spread Designer. From the **Settings** menu, select the **AlternatingRow** icon under the **Other Settings** section.



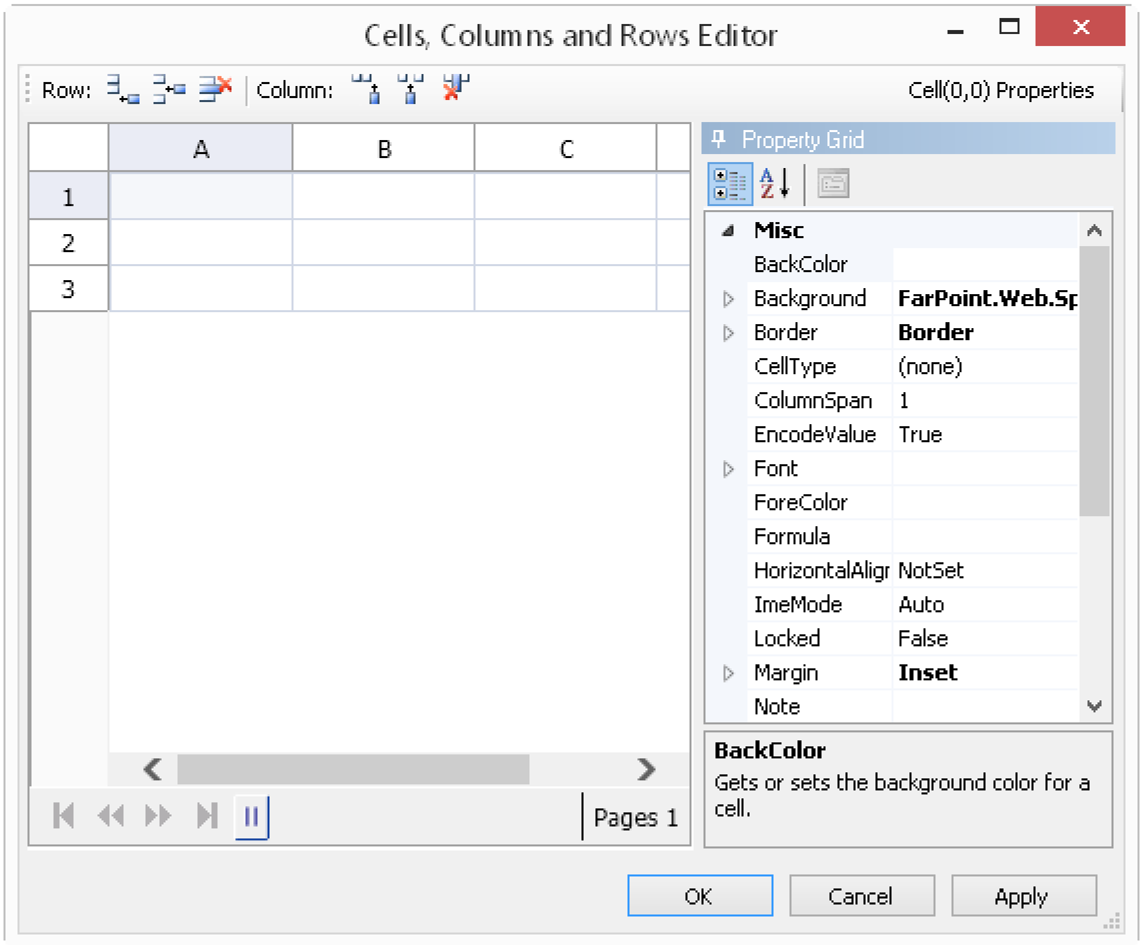
Appearance	
Alignment	**Alignment**
Background	**FarPoint.Web.Sprea**
Borders	**Border**
Color	**ForeBackColor**
EncodeValue	**True**
Font	
Margins	**Inset**
CellType	

Alignment
Gets or sets the alignment of alternating row.

1	000	000	000
2	111	111	111
3	222	222	222
4	333	333	333

Cells, Columns, and Rows Editor

You can customize the appearance of cells, columns or rows with the **Cells, Columns, or Rows Editor** of the Spread Designer. This editor is launched from the **Properties** window by selecting sheet in the drop-down box on the right side of the designer and then clicking on the button for the **Cells, Columns, or Rows** property.



Misc	
BackColor	
Background	FarPoint.Web.Sp
Border	Border
CellType	(none)
ColumnSpan	1
EncodeValue	True
Font	
ForeColor	
Formula	
HorizontalAligr	NotSet
ImeMode	Auto
Locked	False
Margin	Inset
Note	

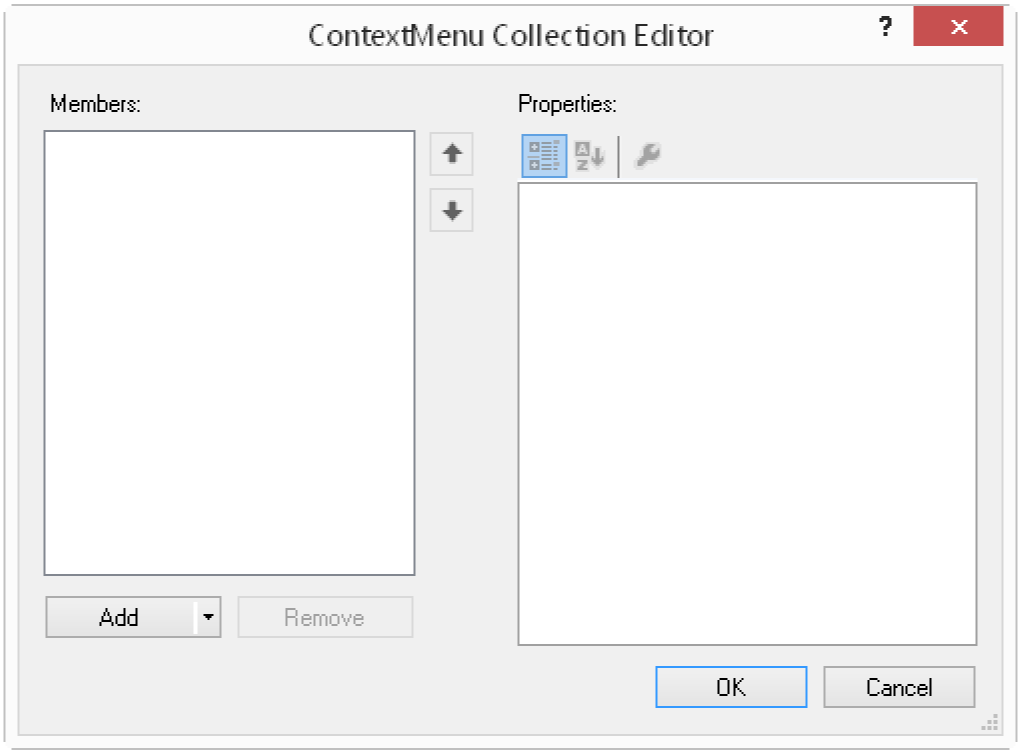
BackColor

Gets or sets the background color for a cell.

For more information about customizing the appearance of cells, refer to **Customizing the Appearance of a Cell**. For more information on rows and columns, refer to **Customizing the Appearance of Rows and Columns**.

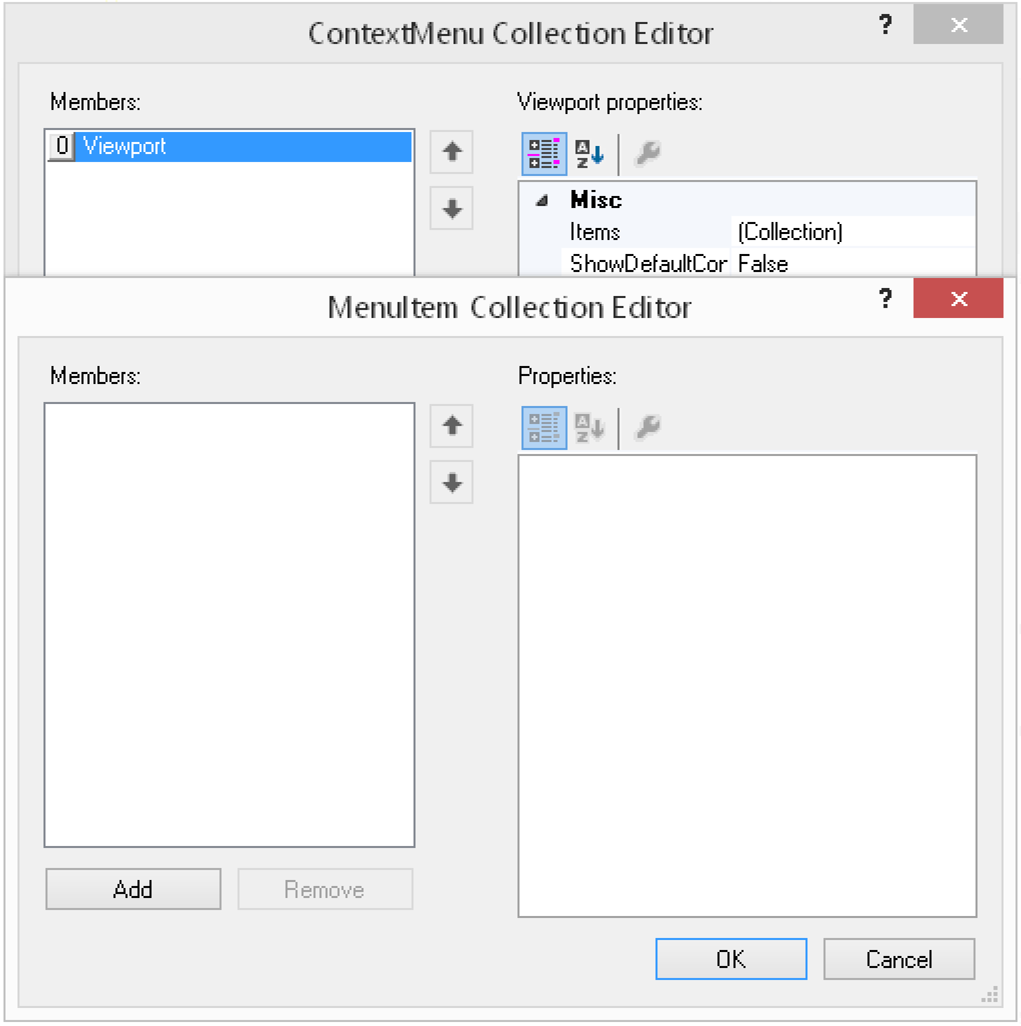
ContextMenu Collection Editor

You can use the **ContextMenu Collection Editor** to create menus for the row header, column header, or viewport area. You can also create menu items and set menu properties. From the **Properties** window, select **ContextMenus** to see this editor.

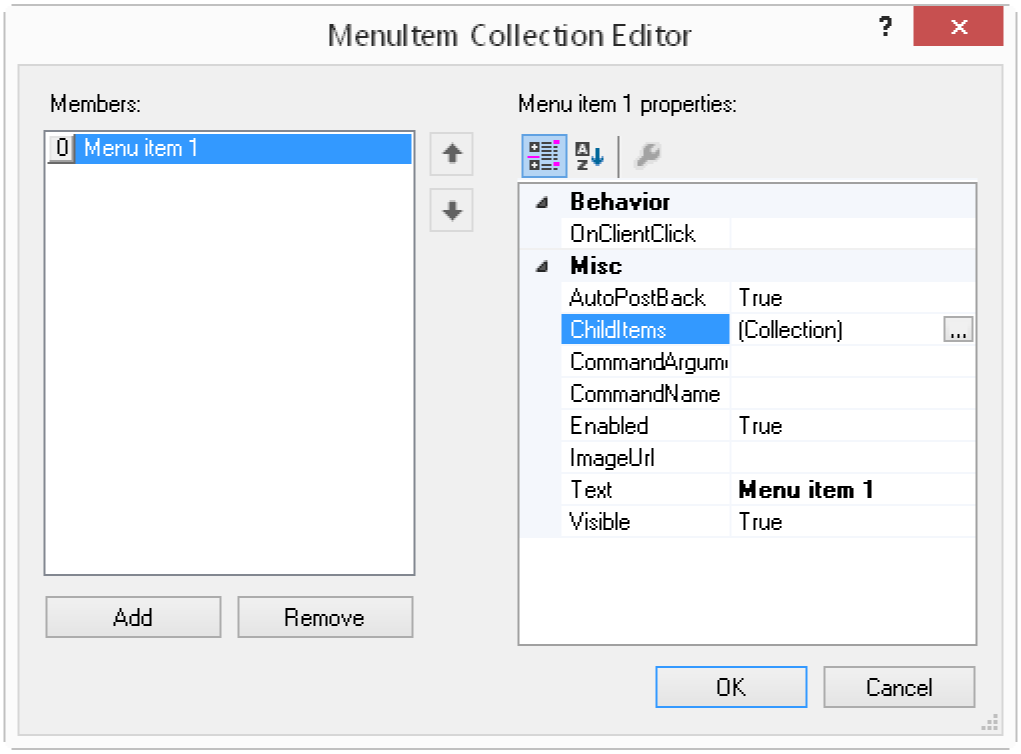


Use the **Add** button to add the type of menu (row header, column header, or viewport).

You can use the **MenuItem Collection Editor** to add menu items after you add a menu type.



After you add a menu item, you can specify menu properties such as visible or add submenu items with the **ChildItems** collection. The **CommandArgument** and **CommandName** properties are used to separate which menu item is clicked in code. The **ImageUrl** property is a small image displayed to the left of the menu item. **Text** is the title of the menu item.



Behavior

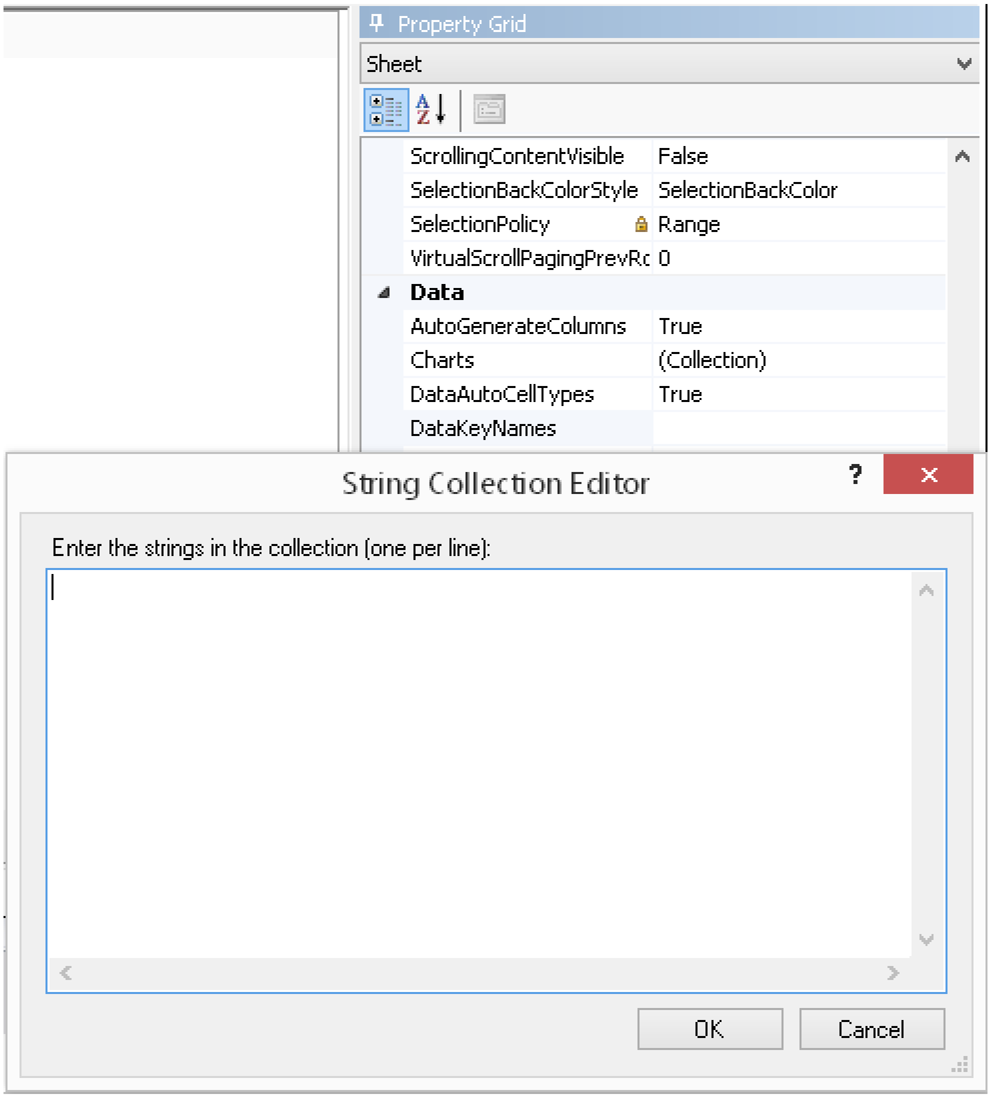
OnClientClick	

Misc

AutoPostBack	True
ChildItems	(Collection) ...
CommandArgum	
CommandName	
Enabled	True
ImageUrl	
Text	**Menu item 1**
Visible	True

DataKey Names (String Collection) Editor

You can set the names of key fields for the data source. Select **DataKeyNames** under the **Data** section after selecting **Sheet** in the drop-down box to the right side of the designer.

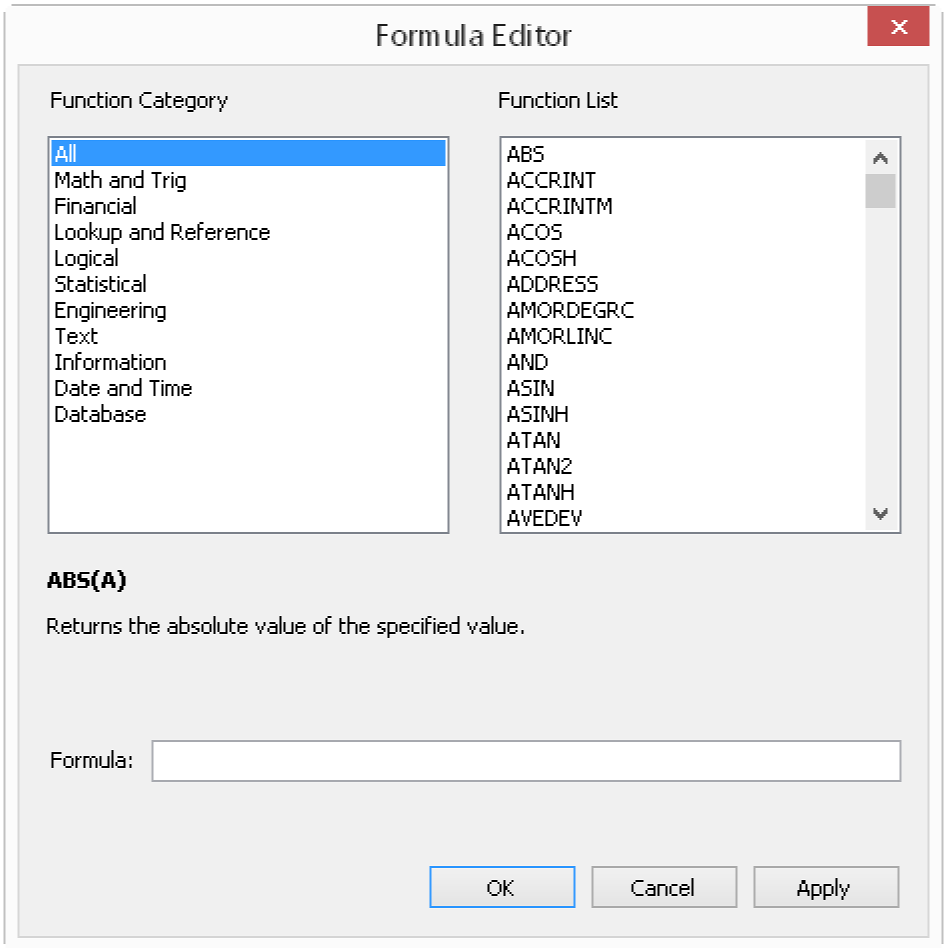


ScrollingContentVisible	False
SelectionBackColorStyle	SelectionBackColor
SelectionPolicy	Range
VirtualScrollPagingPrevRc	0
Data	
AutoGenerateColumns	True
Charts	(Collection)
DataAutoCellTypes	True
DataKeyNames	

Enter the strings in the collection (one per line):

Formula Editor

In the **Formula** bar, when you type an equals sign, ("=") then the drop-down list displays all the built-in functions for you to choose from. Or if you click on the **Choose Formula** button on the **Formula** bar, the **Formula Editor** is displayed. The **Formula Editor** allows you to select any of the built-in functions.



ABS(A)

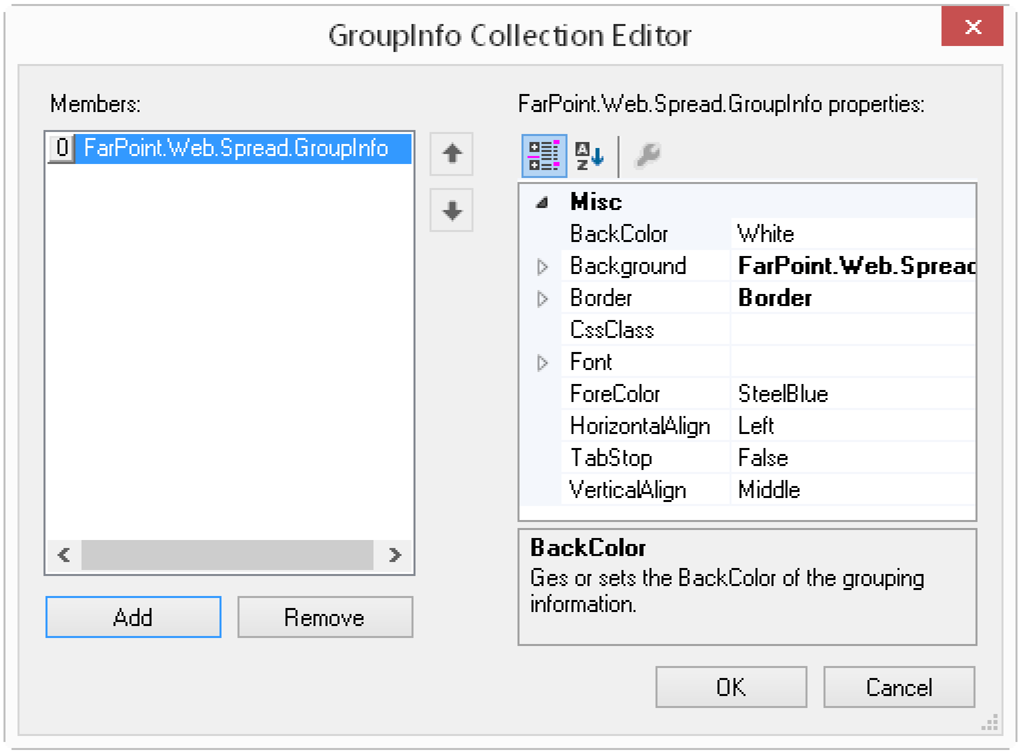
Returns the absolute value of the specified value.

The **Formula Editor** gives you a list of the built-in functions that you can use and displays a brief description and syntax of the selected function. To choose a function, double-click on the function name and it appears in the Formula field. You may also type operators and constants to construct your formula.

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

GroupInfo Collection Editor

You can set basic formatting for group headers with the **GroupInfo Collection Editor** of the Spread Designer. You can launch the **GroupInfo Collection Editor** from the Spread Designer by selecting the sheet from the drop down on the right side of the designer and choosing **GroupInfos** under the **Misc** section.


GroupInfo Collection Editor
Members:
FarPoint.Web.Spread.GroupInfo properties:
Add
Remove
OK
Cancel

0 FarPoint.Web.Spread.GroupInfo

Misc

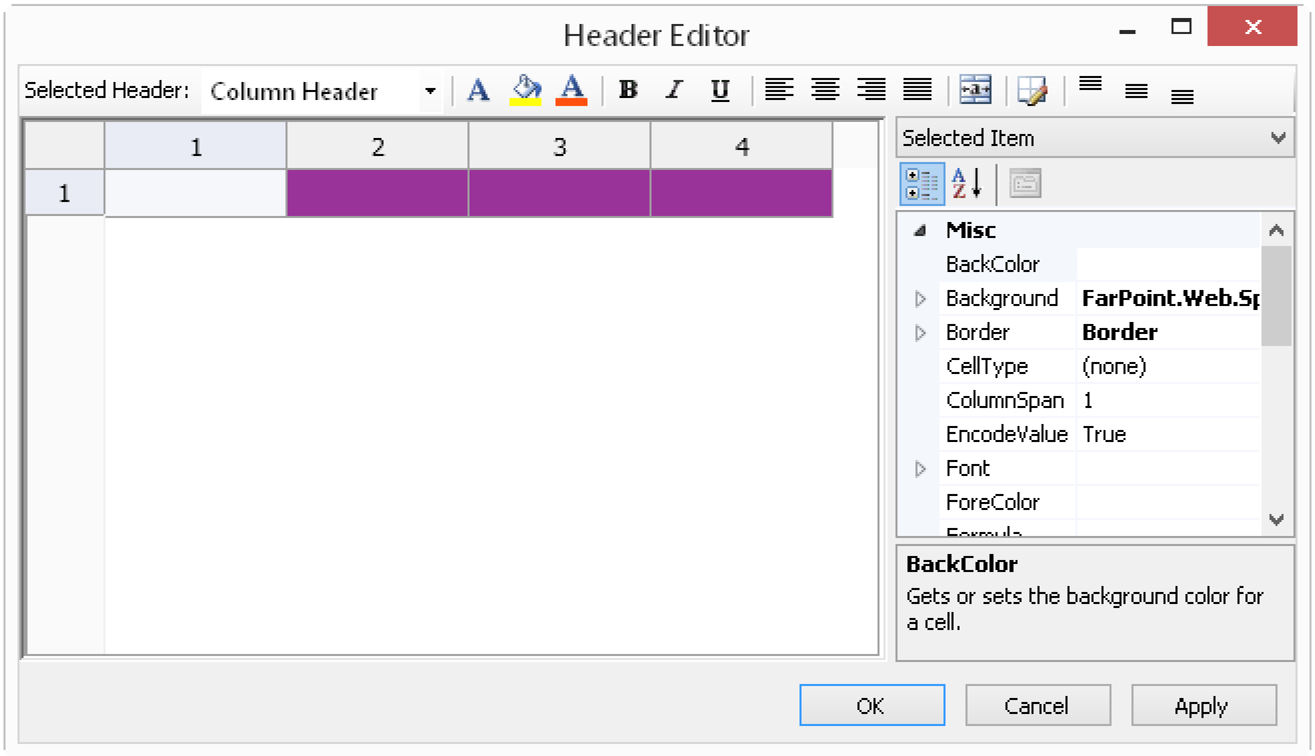
BackColor	White
Background	FarPoint.Web.Spreac
Border	Border
CssClass	
Font	
ForeColor	SteelBlue
HorizontalAlign	Left
TabStop	False
VerticalAlign	Middle

BackColor
Ges or sets the BackColor of the grouping information.

For more information on grouping and grouping headers, refer to **Customizing Grouping of Rows of User Data**.

Header Editor

You can customize column and row headers by selecting which headers display and by customizing the properties of the headers. In the Spread Designer, from the **Settings** menu, select the **Header Editor** icon. An example of a header dialog with customized header appearances is shown here.

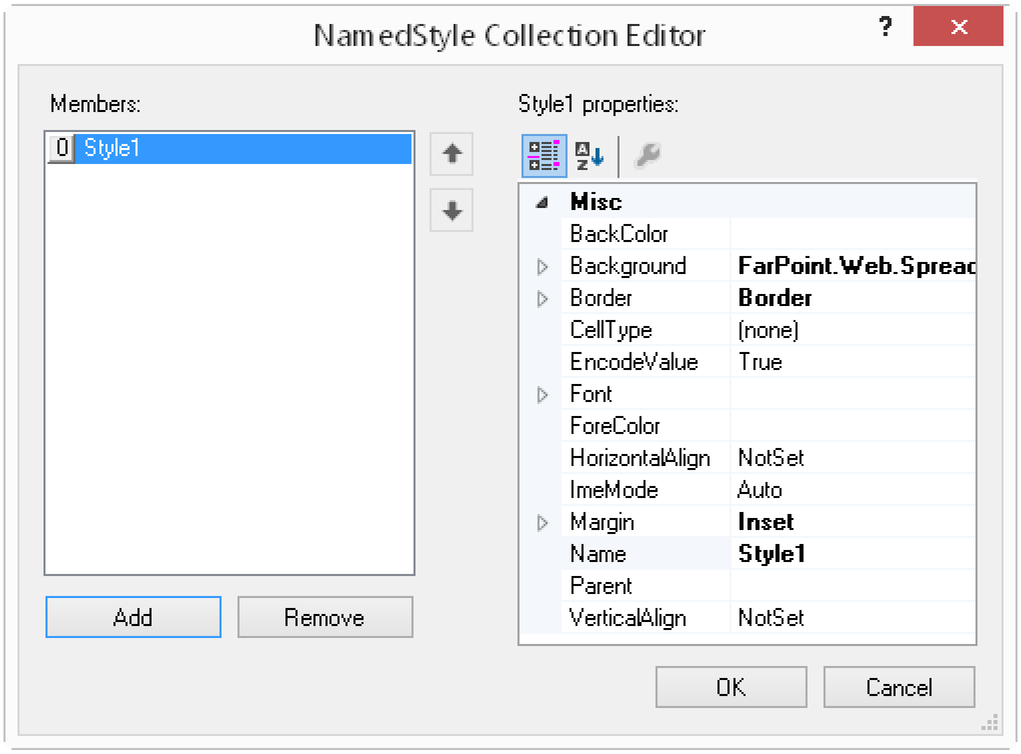


First, select which headers these customizations apply to by choosing from the drop-down list at the top of the dialog. Then select the formatting from the format bar at the top or the various properties listed in the **Property** window to the right. The preview pane on the left displays how those customizations appear. When done, click **Apply** or **OK**.

For more information on customizing headers, refer to **Customizing the Appearance of Headers**.

NamedStyle Collection Editor

You can customize the appearance of cells by defining a named style. You can do this within the Spread Designer using the **Named Style Editor**. This editor is launched from the **Properties** window by first selecting the Spread and then clicking on the button in the **NamedStyle** property.

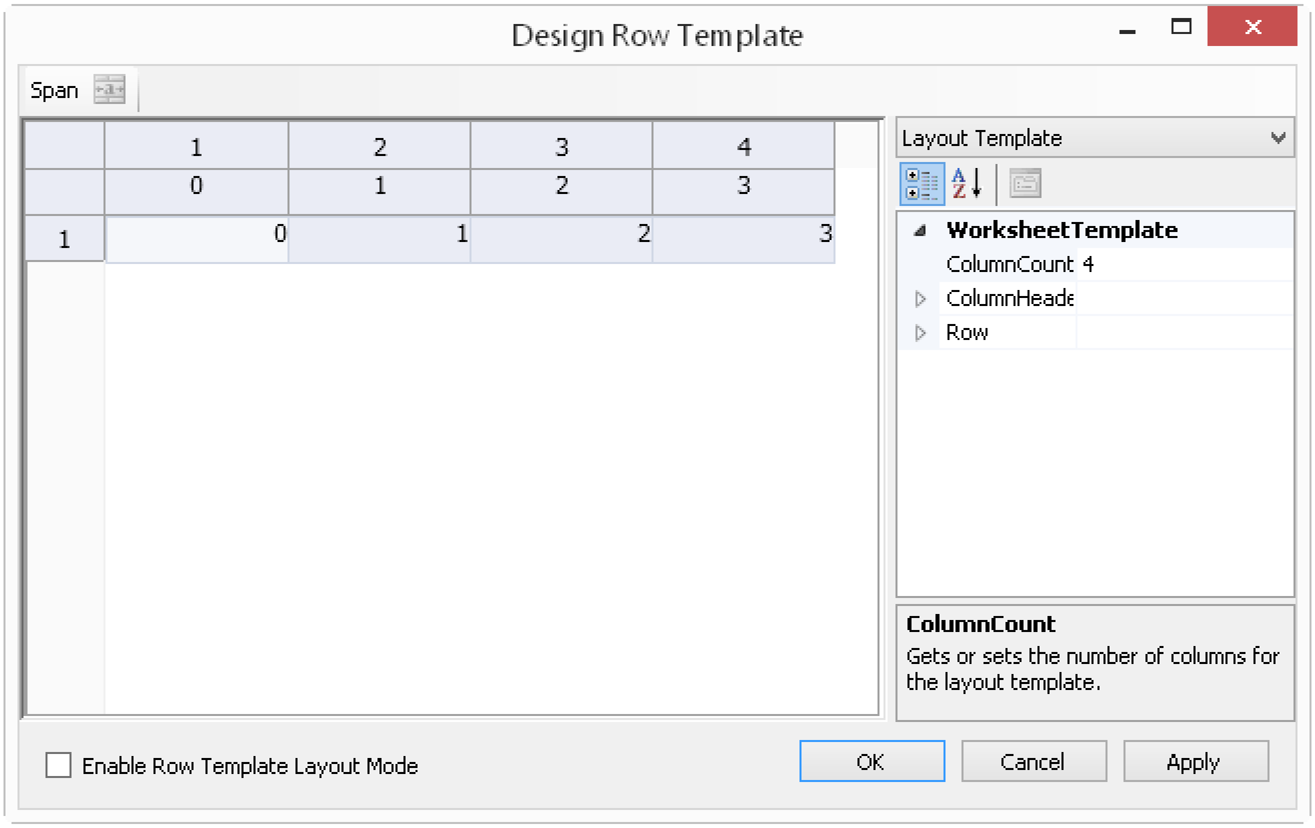


Misc	
BackColor	
Background	FarPoint.Web.Spreac
Border	Border
CellType	(none)
EncodeValue	True
Font	
ForeColor	
HorizontalAlign	NotSet
ImeMode	Auto
Margin	Inset
Name	Style1
Parent	
VerticalAlign	NotSet

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

Row Template Editor

The row template editor allows you to design the layout of the column headers and the data rows. From the **Settings** menu, select the **RowTemplate** icon (**Other Settings** section).



	1	2	3	4
	0	1	2	3
1	0	1	2	3

WorksheetTemplate

ColumnCount	4
ColumnHeade	
Row	

ColumnCount
Gets or sets the number of columns for the layout template.

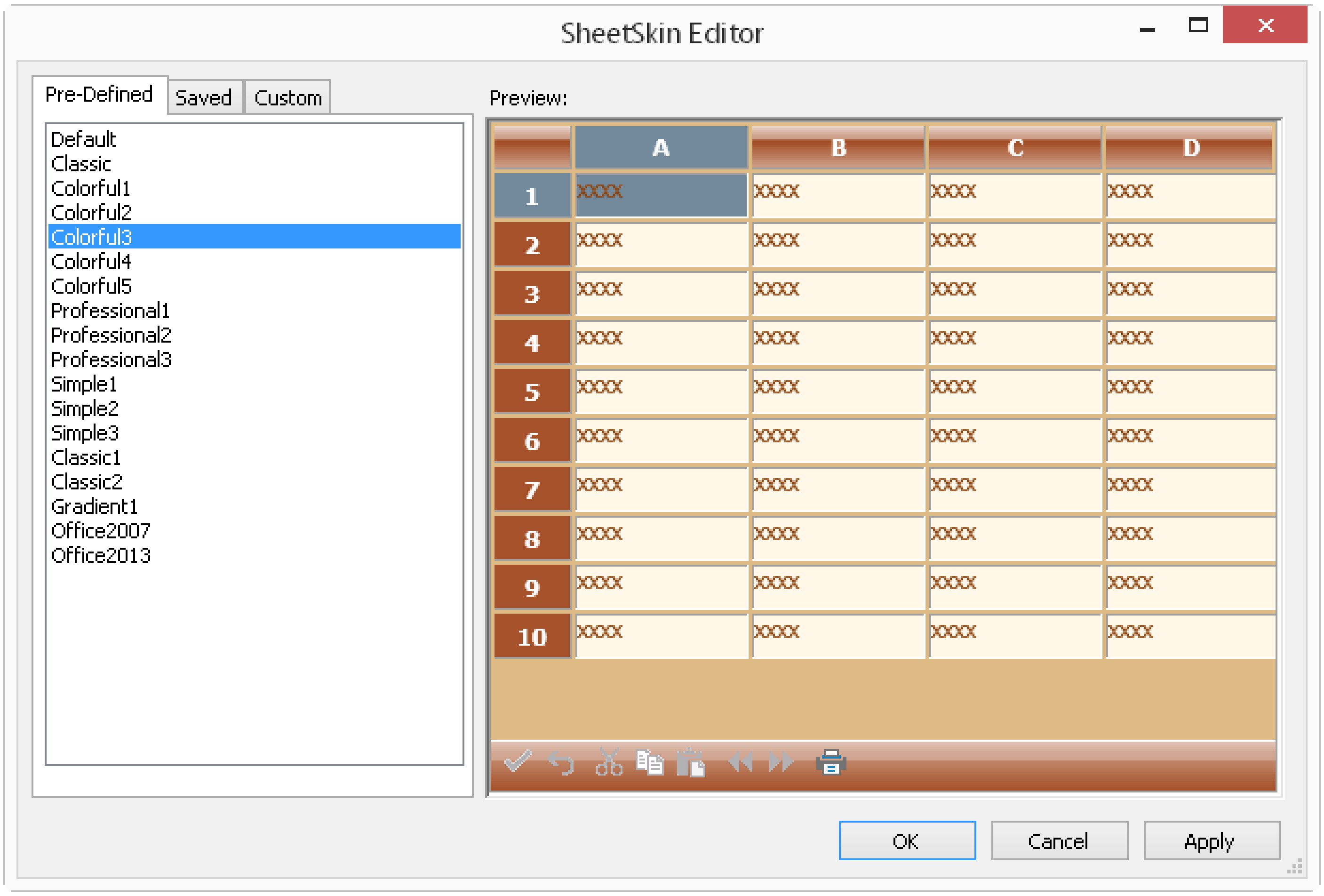
□ Enable Row Template Layout Mode

You can use the **Span** icon to span cells in the data row or column header. Select the cells to span and then select the **Span** icon.

SheetSkin Editor

You can customize various sheet properties and save them as a set called a skin. That skin can be saved and used in other projects. You can also use pre-defined built-in skins and apply a set of appearance settings at once.

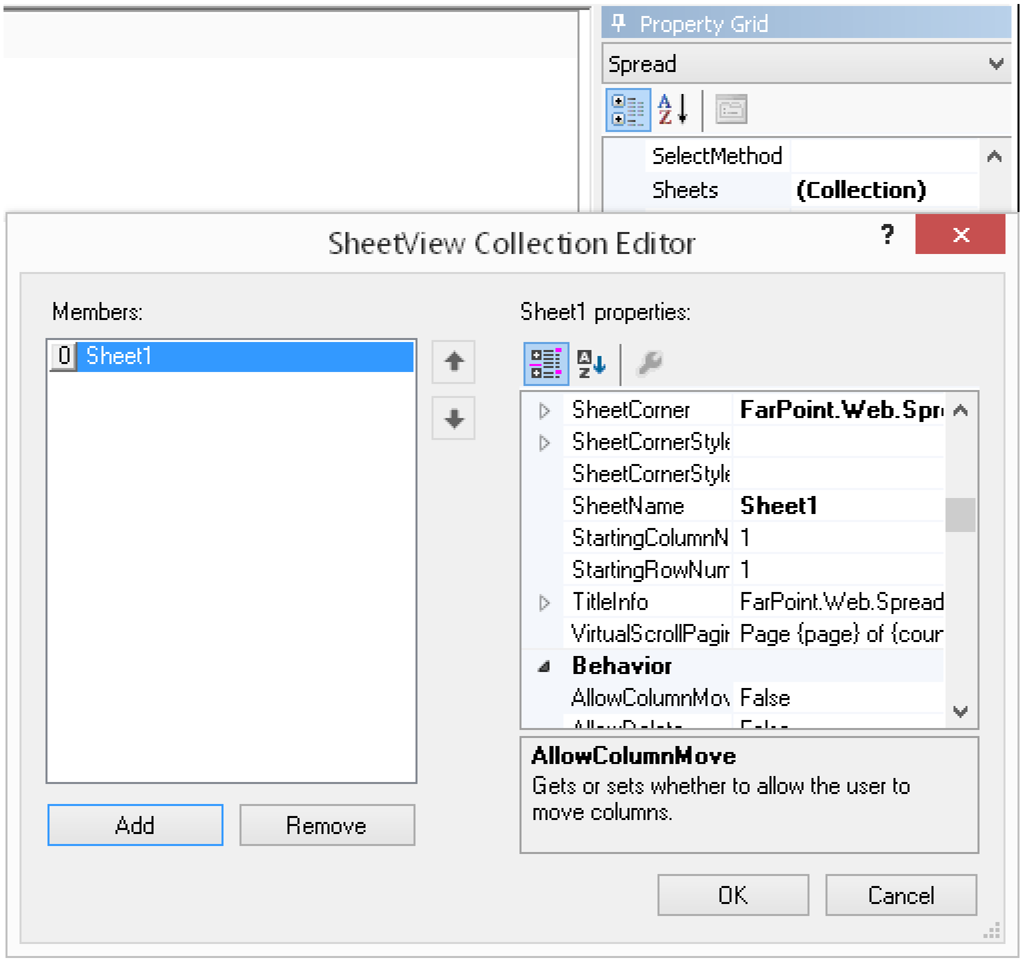
Select the **Settings** menu in the Spread Designer, then select the **SheetSkin** menu under the **Appearance Settings** section. The **SheetSkin** editor is shown here.



For information about using skins, refer to **Applying a Skin to a Sheet** and **Creating a Skin for Sheets**.

SheetView Collection Editor

You can customize the appearance of sheets with the **Sheet View Collection Editor**. Select Spread in the drop-down box on the right side of the designer. Then click on the **Sheets** collection under the **Data** section.



SelectMethod
Sheets (Collection)

0 Sheet1

Sheet1 properties:

SheetCorner	FarPoint.Web.Spr
SheetCornerStyle	
SheetCornerStyle	
SheetName	Sheet1
StartingColumnN	1
StartingRowNum	1
TitleInfo	FarPoint.Web.Spread
VirtualScrollPagi	Page {page} of {cour
Behavior	
AllowColumnMov	False

AllowColumnMove
Gets or sets whether to allow the user to move columns.

For more information on sheet appearance settings, refer to **Customizing the Appearance of the Sheet**.

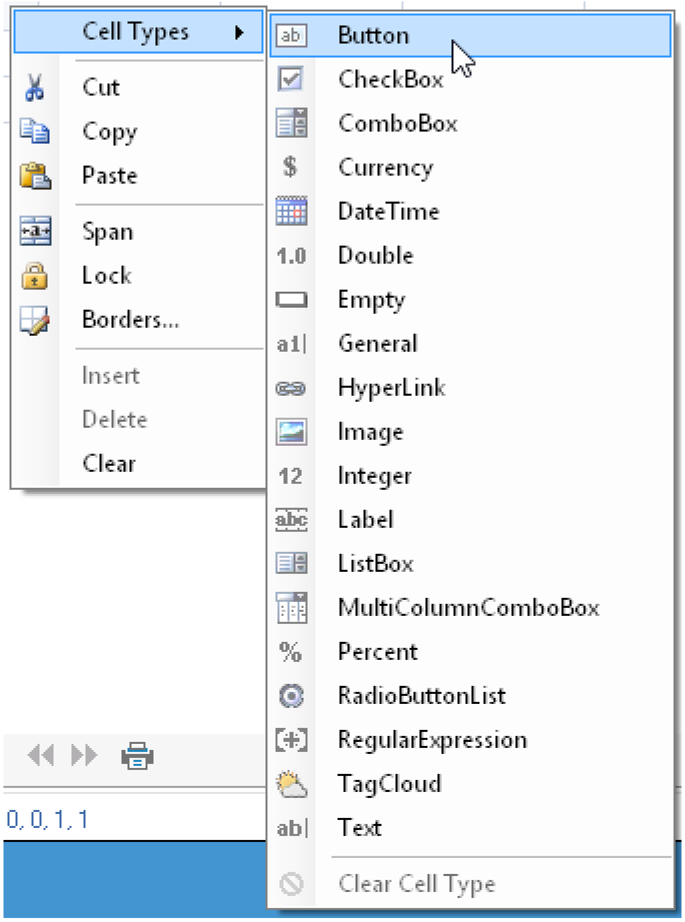
Spread Designer Context Menus

In the spreadsheet preview area of the Spread Designer you right-click and bring up a context menu depending on whether you are clicking on an individual cell or the entire sheet or component.

For more tasks within Spread Designer, return to **Understanding the Spread Designer Interface**.

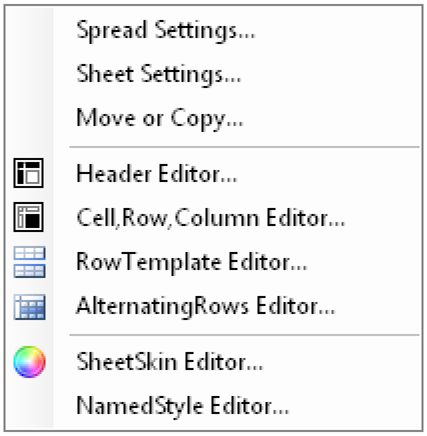
Cell Context Menu

With the cell selected, you can display the cell context menu.



Sheet Context Menu

You can select the sheet or component and right-click to display the sheet context menu. The **Move** or **Copy** menu option allows you to make a copy of a sheet.



Using the Spread Designer

Spread Designer helps you design your FpSpread component by letting you see most of the settings you make at the time you make them, and by letting you access settings that you cannot access in Visual Studio at design time. You can use Spread Designer for many aspects of design, including customizing the appearance of your component. You can also load data into the sheets in your component, if you want to do so.

The tasks you can do using Spread Designer are described throughout this guide in the "how-to" instructions provided. The following topics describe some specific tasks you need to do in Spread Designer to work with the component and the Designer. Refer to these topics if you have questions while working through tasks described in other sections in this guide.

The following sections describe working inside the Spread Designer:

- **Customizing Sheets, Rows, and Columns in Spread Designer**
- **Customizing Cells in Spread Designer**
- **Adding Formulas to Cells**
- **Saving and Opening Design Files**
- **Applying Changes and Closing Spread Designer**

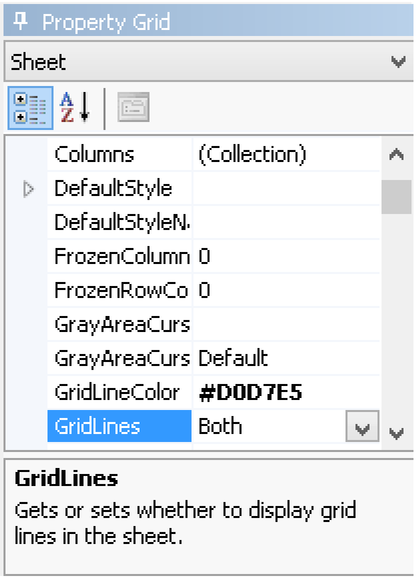
For more information about Spread Designer, return to **Working with the Spread Designer**.

Customizing Sheets, Rows, and Columns in Spread Designer

Using the Spread Designer, you can set several types of properties to customize the sheet appearance. This includes the following sheet properties:

- Colors
- Grid lines
- Sheet names
- Alternating rows
- Group bar colors
- Header starting numbers
- Automatic recalculation

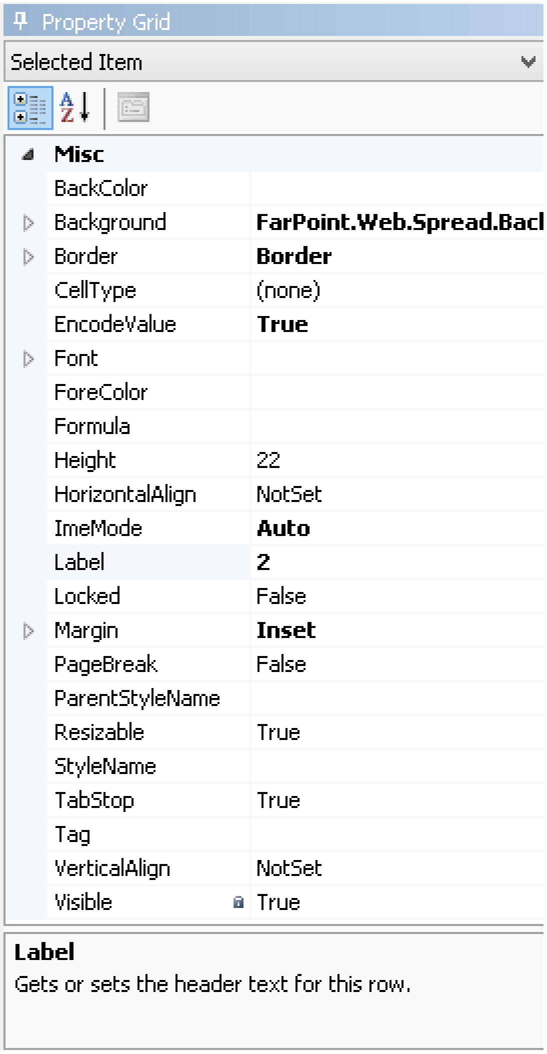
For example, the grid lines properties are shown in the following figure.



Columns	(Collection)
DefaultStyle	
DefaultStyleN.	
FrozenColumn	0
FrozenRowCo	0
GrayAreaCurs	
GrayAreaCurs	Default
GridLineColor	**#D0D7E5**
GridLines	Both

GridLines
Gets or sets whether to display grid lines in the sheet.

Besides the Sheet properties, you can also set several types of properties to customize the appearance of rows and columns and headers using the Spread Designer. Use the **Row** menu for rows and row headers. Use the **Column** menu for columns and column headers. The figure below shows the types of properties that are available for rows.



Misc	
BackColor	
Background	FarPoint.Web.Spread.Bacl
Border	Border
CellType	(none)
EncodeValue	True
Font	
ForeColor	
Formula	
Height	22
HorizontalAlign	NotSet
ImeMode	Auto
Label	2
Locked	False
Margin	Inset
PageBreak	False
ParentStyleName	
Resizable	True
StyleName	
TabStop	True
Tag	
VerticalAlign	NotSet
Visible	True

Label
Gets or sets the header text for this row.

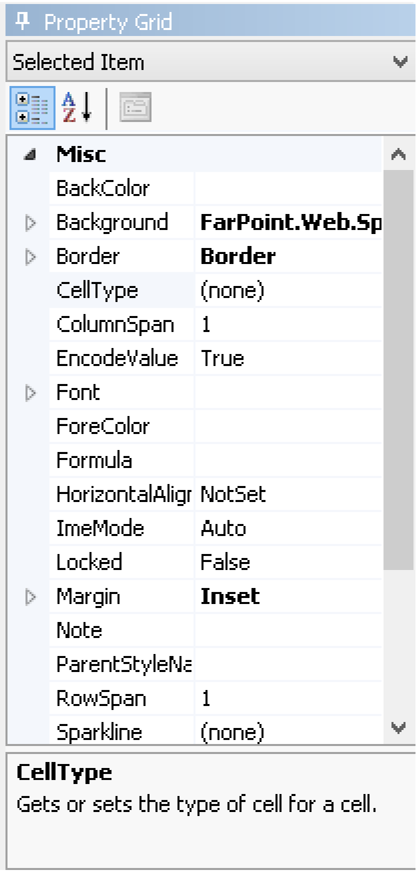
For more information on setting cell types for a cell or range of cells, refer to **Customizing Cells in Spread Designer**.

For more tasks within Spread Designer, return to **Using the Spread Designer**.

Customizing Cells in Spread Designer

Using the Spread Designer, you can set the cell types of cells in the data area of the spreadsheet.

Select the cells in the window of the Designer for which you want to set the cell type.



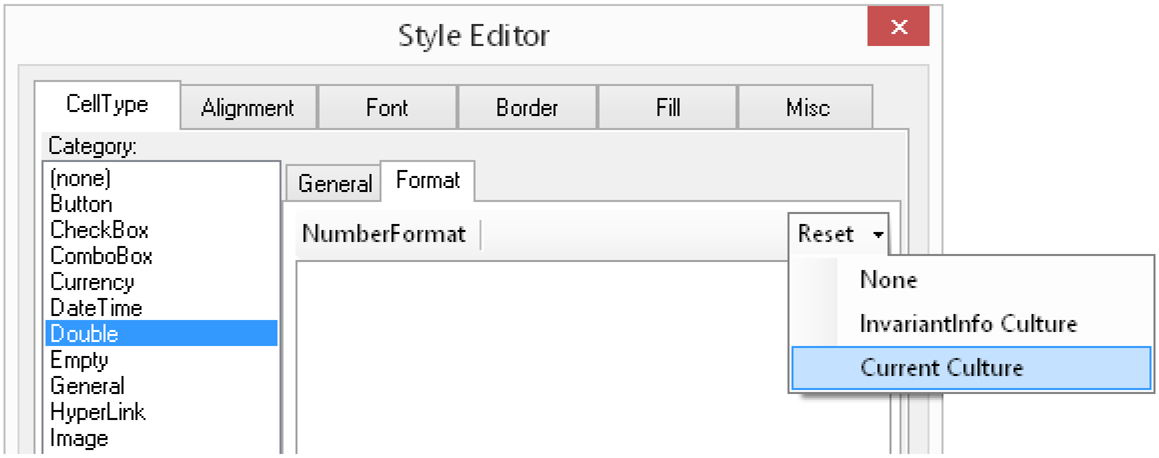
Misc

BackColor	
Background	FarPoint.Web.Sp
Border	Border
CellType	(none)
ColumnSpan	1
EncodeValue	True
Font	
ForeColor	
Formula	
HorizontalAligr	NotSet
ImeMode	Auto
Locked	False
Margin	Inset
Note	
ParentStyleNa	
RowSpan	1
Sparkline	(none)

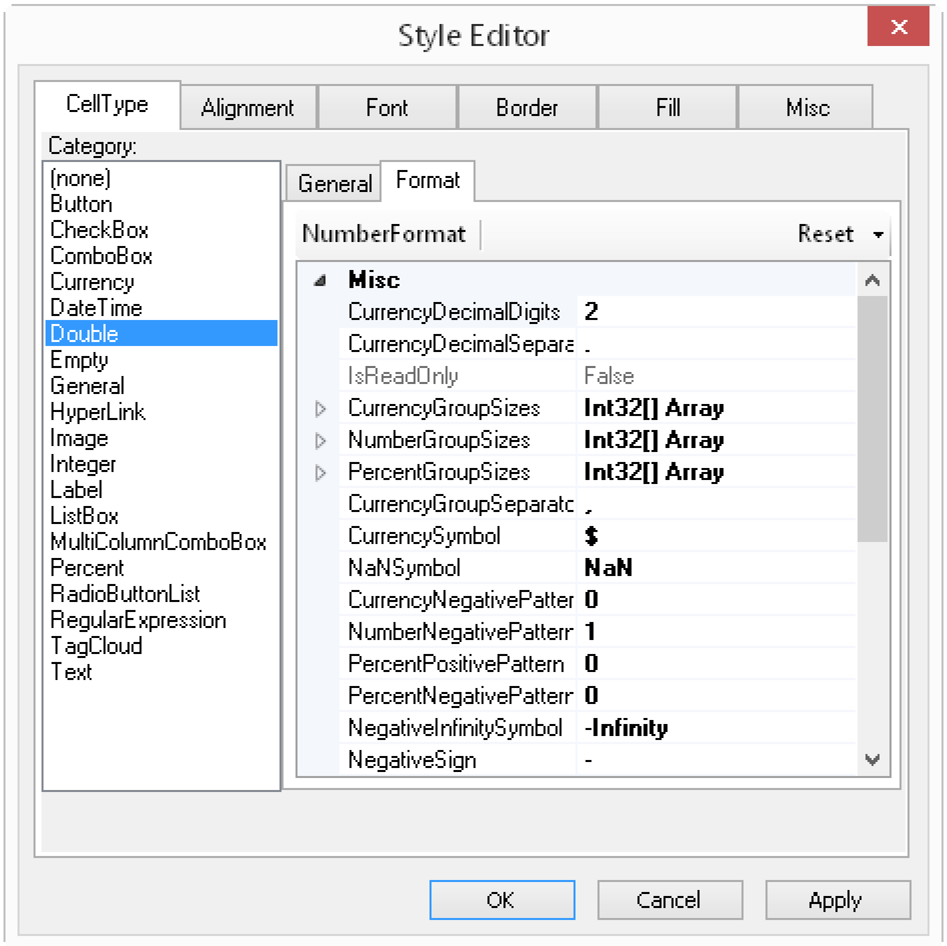
CellType
Gets or sets the type of cell for a cell.

If you are setting cell types for a given cell or range of cells, use the cell types right-click menu to quickly apply the cell type. After making one or more settings, click **Apply** to apply the changes to Spread.

Several editable celltypes have a **Format** tab that can be used to set the **NumberFormat** property. The **NumberFormat** property uses the **NumberFormatInfo** class. The **NumberFormatInfo** class may have properties that the cell type does not support. Unsupported properties are ignored.



Select the culture type and then set any of the number format properties.



Misc

CurrencyDecimalDigits	2
CurrencyDecimalSepara	.
IsReadOnly	False
CurrencyGroupSizes	Int32[] Array
NumberGroupSizes	Int32[] Array
PercentGroupSizes	Int32[] Array
CurrencyGroupSeparatc	,
CurrencySymbol	$
NaNSymbol	NaN
CurrencyNegativePatter	0
NumberNegativePatterr	1
PercentPositivePattern	0
PercentNegativePatterr	0
NegativeInfinitySymbol	-Infinity
NegativeSign	-

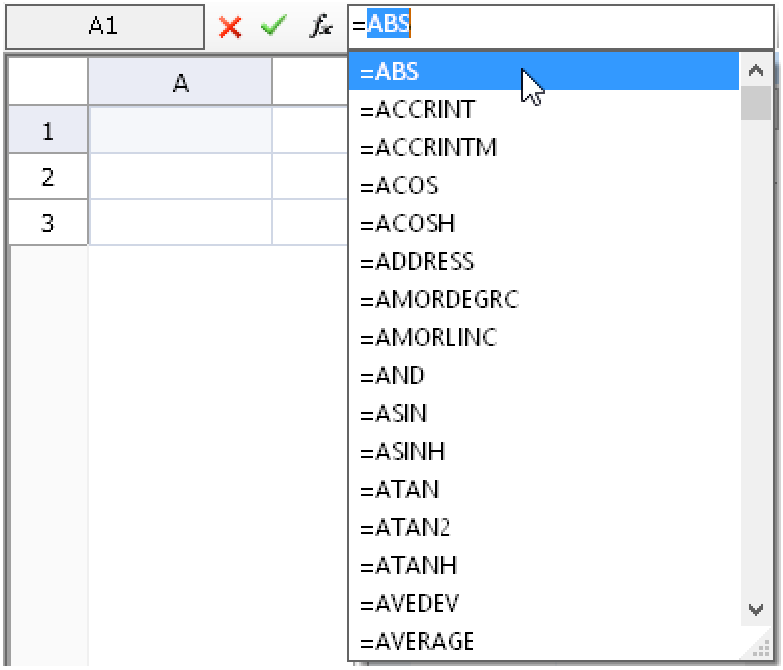
You can enter a formula into a cell or range of cells using the Formula property from the **Formula** tab of the **Cell** page in the Spread Designer. For more information, refer to **Adding Formulas to Cells**. For more information on formulas and functions, refer to the Formula Reference.

For more information on setting other properties for a cell or range of cells, refer to **Customizing Sheets, Rows, and Columns in Spread Designer**.

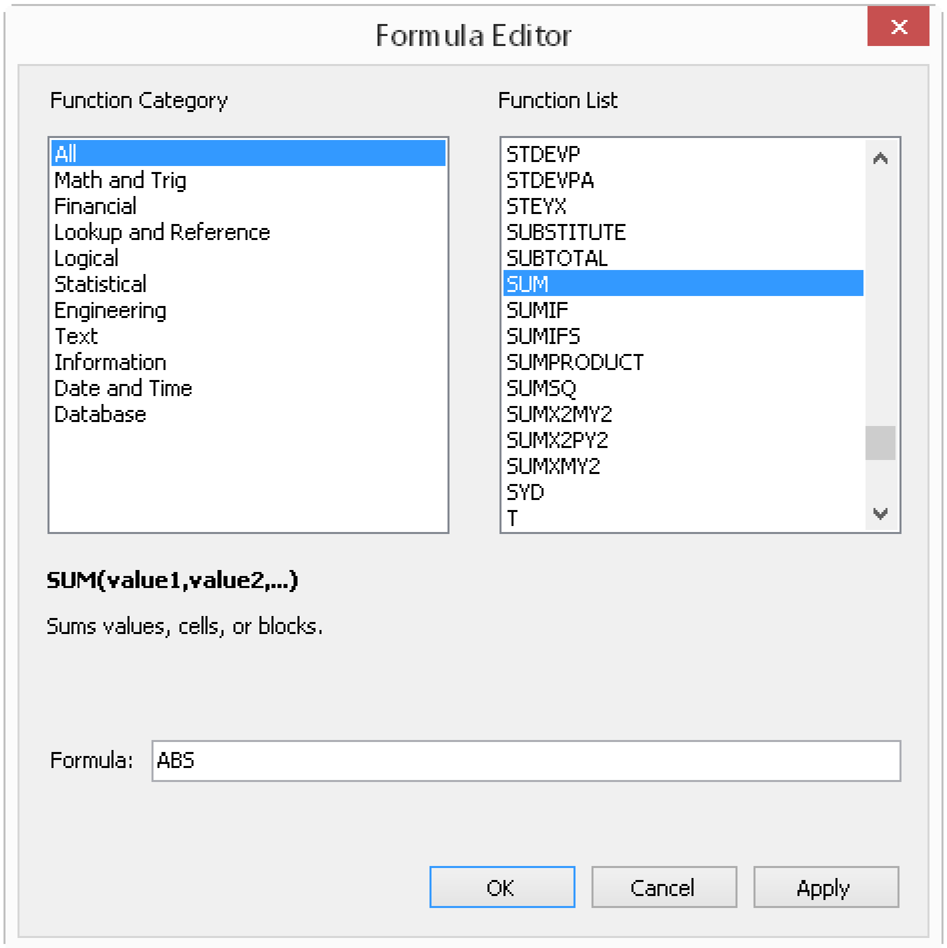
For more tasks within Spread Designer, return to **Using the Spread Designer**.

Adding Formulas to Cells

You can enter a formula into a cell or range of cells using the **Formula** bar and the **Formula Editor** in the Spread Designer.



In the **Formula** bar, when you type an equals sign, ("=") then the drop-down list displays all the built-in functions for you to choose from. Or if you click on the **Choose Formula** button on the **Formula** bar, the **Formula Editor** is displayed. The **Formula Editor** allows you to select any of the built-in functions.



SUM(value1,value2,...)

Sums values, cells, or blocks.

Formula: ABS

The **Formula Editor** gives you a list of the built-in functions that you can use and displays a brief description and syntax of the selected function. Functions are organized by category; you can select a category to show only functions of a given category. To choose a function, double-click on the function name and it appears in the Formula field. You may also type operators and constants to construct your formula.

You can enter the formula in the Formula field in the **Formula Editor** or in the formula box in the **Formula** bar. When you are done entering the formula with the **Formula Editor**, click **Apply** or **OK**. When you are done typing the formula in the formula box, click Enter (the check mark button). This applies the formula to the selected cell or range of cells. When you click **OK** or **Apply**, the **Formula Editor** evaluates the formula to see if it is a valid formula. For more information on formulas and functions, refer to the Formula Reference.

To display or hide the formula bar, from the **View** menu, select **Formula Bar**.

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

For more tasks within Spread Designer, return to **Using the Spread Designer**.

Saving and Opening Design Files

When you have finished working on a design, you can save the design to a file as any of several file types:

- Spread XML
- Excel (BIFF) XLS
- Excel 2007 XLSX
- Text file

From the **File** menu, select **Save** or **Save As New** and specify a file name, the file type, and the location of the file.

To open an existing file, from the **File** menu, select **Open** and select the file from the **File** dialog, or **Open Recent** and select the name from the recent files list.

For saving Spread Designer to an XML file, the Spread element contains these elements

- Data
- Presentation
- Settings
- Style

For more information on the save and open options from the **File** menu, refer to **File Menu**.

For details of what is exported to the BIFF-compatible file, refer to the **Import and Export Reference (on-line documentation)**.

For more tasks within Spread Designer, return to **Using the Spread Designer**.

Applying Changes and Closing Spread Designer

When you have finished setting the properties you want to set in Spread Designer, you can apply your changes to the component, and then either continue to work in Spread Designer or close Spread Designer.

To apply your changes, do one of the following. From the Designer **File** menu,

- choose **Apply**
- choose **Apply and Exit**

For more tasks within Spread Designer, return to **Using the Spread Designer**.

Customizing User Interaction

You can customize how the user interacts with the spreadsheet. The tasks that relate to customizing the way the user interacts with the spreadsheet include:

- **Customizing Interaction with the Overall Component**
- **Working with AJAX**
- **Customizing the Tool Bars**
- **Customizing Interaction with Rows and Columns**
- **Managing Filtering of Rows of User Data**
- **Customizing Grouping of Rows of User Data**
- **Customizing Sorting of Rows of User Data**
- **Customizing Interaction with Cells**
- **Customizing Selections of Cells**
- **Managing Printing**

Customizing Interaction with the Overall Component

You can customize some aspects of the user interaction with the overall component. To customize user interaction, you may perform the following tasks:

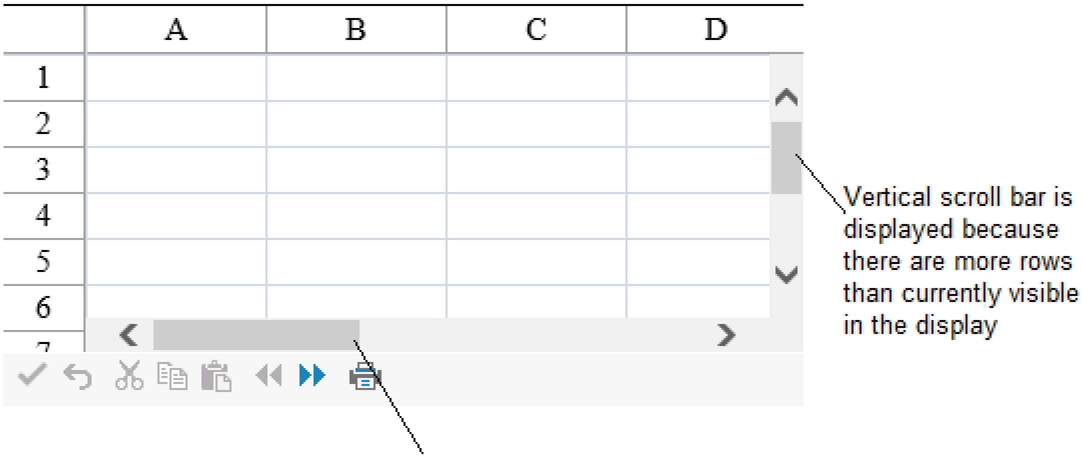
- **Displaying Scroll Bars**
- **Displaying Scroll Bar Text Tips**
- **Customizing the Scroll Bar Colors**
- **Allowing Load on Demand**
- **Customizing Interaction Based on Events**
- **Handling the Tab Key**
- **Customizing the Graphical Interface**
- **Searching for Data with Code**
- **Adding a Context Menu**
- **Using the Formula Extender Control (on-line documentation)**

For information about the appearance of the component, refer to **Customizing the Appearance of the Overall Component**.

For information about displaying the sheet names, refer to **Displaying the Sheet Names**.

Displaying Scroll Bars

You can customize how and if to display the scroll bars in the component. You can display the individual scroll bars (horizontal or vertical) only when needed, as shown in the figure.



Vertical scroll bar is displayed because there are more rows than currently visible in the display

Horizontal scroll bar is displayed because there are more columns than currently visible in the display

Using the Properties Window

1. Select the FpSpread component.
2. With the **Properties** window open, select the **HorizontalScrollBarPolicy** property and **VerticalScrollBarPolicy** (under the **Behavior** category) and from the drop-down list, select a value for each.
3. The scroll bar policy is now set.

Using Code

Determine when to display the scroll bars by setting the **HorizontalScrollBarPolicy ('HorizontalScrollBarPolicy Property' in the on-line documentation)** property and **VerticalScrollBarPolicy ('VerticalScrollBarPolicy Property' in the on-line documentation)** property for the **FpSpread ('FpSpread Class' in the on-line documentation)** component and the settings of the **ScrollBarPolicy ('ScrollBarPolicy Enumeration' in the on-line documentation)** enumeration.

Example

The following example sets the horizontal and vertical scroll bar policies.

C#

```
FpSpread1.HorizontalScrollBarPolicy = FarPoint.Web.Spread.ScrollBarPolicy.Always;
FpSpread1.VerticalScrollBarPolicy = FarPoint.Web.Spread.ScrollBarPolicy.AsNeeded;
```

VB

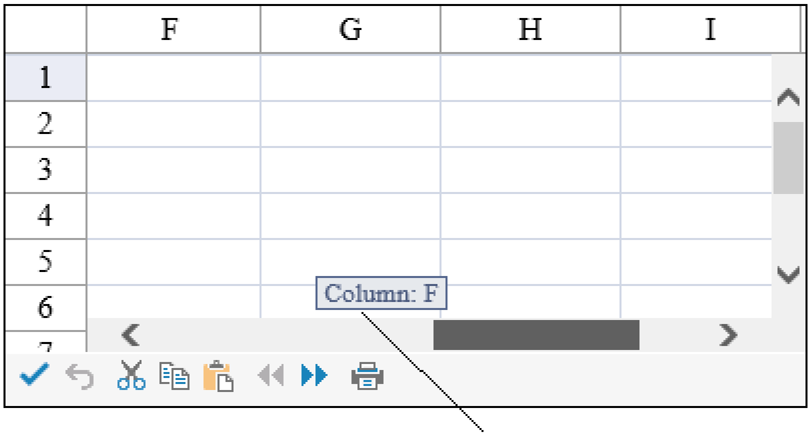
```
FpSpread1.HorizontalScrollBarPolicy = FarPoint.Web.Spread.ScrollBarPolicy.Always
FpSpread1.VerticalScrollBarPolicy = FarPoint.Web.Spread.ScrollBarPolicy.AsNeeded
```

Using the Spread Designer

1. Select the **Settings** menu.
2. Select the **Scrollbar** icon under the **Spread Settings** section.
3. Set the policy options.
4. Click **OK** to apply the changes.
5. Click **Apply and Exit** to close the Spread Designer.

Displaying Scroll Bar Text Tips

You can display scroll bar text tips for the sheet when the user scrolls. The text tip displays information for the leftmost column or the topmost row in the viewing area.



Scroll bar text tip

You can also customize the scrolling text tip when using virtual paging with the **VirtualScrollPagingFormatString ('VirtualScrollPagingFormatString Property' in the on-line documentation)** property.

Using the Properties Window

1. Select the FpSpread component.
2. Select the **Sheets** collection in the properties window.
3. Select the **ScrollingContentVisible** property in the **Sheets** collection editor.

Using Code

Use the **ScrollingContentVisible ('ScrollingContentVisible Property' in the on-line documentation)** property to enable tips for the scroll bar.

Example

The following example enables the tips for the scroll bar.

C#

```
FpSpread1.Sheets[0].ScrollingContentVisible = true;
```

VB

```
FpSpread1.Sheets(0).ScrollingContentVisible = True
```

Using the Spread Designer

1. Select the **Settings** menu.
2. Select the **Scrollbar** icon under the **Spread Settings** section.
3. Select the **Scrolling Content** option.
4. Click **OK** to apply the changes.
5. Click **Apply and Exit** to close the Spread Designer.

Customizing the Scroll Bar Colors

```
FpSpread1.ScrollBarFaceColor = Color.Orange;
FpSpread1.ScrollBarHighlightColor = Color.White;
FpSpread1.ScrollBarShadowColor = Color.Blue;
FpSpread1.ScrollBarTrackColor = Color.Pink;
```

VB

```
FpSpread1.VerticalScrollBarPolicy = FarPoint.Web.Spread.ScrollBarPolicy.AsNeeded
FpSpread1.VerticalScrollBarPolicy = FarPoint.Web.Spread.ScrollBarPolicy.AsNeeded
FpSpread1.Sheets(0).ColumnCount = 10
FpSpread1.Sheets(0).RowCount = 10
FpSpread1.ScrollBar3DLightColor = Color.Yellow
FpSpread1.ScrollBarArrowColor = Color.Green
FpSpread1.ScrollBarBaseColor = Color.Brown
FpSpread1.ScrollBarDarkShadowColor = Color.Purple
FpSpread1.ScrollBarFaceColor = Color.Orange
FpSpread1.ScrollBarHighlightColor = Color.White
FpSpread1.ScrollBarShadowColor = Color.Blue
FpSpread1.ScrollBarTrackColor = Color.Pink
```

Allowing Load on Demand

You can allow the Web page to load on demand -- as the user scrolls further down the spreadsheet the Spread component on the client loads another set of rows from the server as needed. The height of the component should be smaller than the height needed for the initial number of rows to load (**LoadInitRowCount ('LoadInitRowCount Property' in the on-line documentation)** property); otherwise, the scroll bar will not be visible and you will need to use the next page icon instead of the scroll bar. The load on demand feature scrolls up to the maximum number of rows you have set with the page size. If the row count is greater than the page size, you will need to use the next page icon to display the rows beyond the page size setting.

The following properties are used to set the allow load on demand feature. They can be set at the control level or the sheet level.

FpSpread class:

- FpSpread.**AllowLoadOnDemand ('AllowLoadOnDemand Property' in the on-line documentation)**
- FpSpread.**LoadInitRowCount ('LoadInitRowCount Property' in the on-line documentation)**
- FpSpread.**LoadRowIncrement ('LoadRowIncrement Property' in the on-line documentation)**

SheetView class:

- SheetView.**AllowLoadOnDemand ('AllowLoadOnDemand Property' in the on-line documentation)**
- SheetView.**LoadInitRowCount ('LoadInitRowCount Property' in the on-line documentation)**
- SheetView.**LoadRowIncrement ('LoadRowIncrement Property' in the on-line documentation)**

You can specify whether to use the standard or background load on demand options with the **LoadOnDemandMode ('LoadOnDemandMode Property' in the on-line documentation)** property.

The load on demand feature is not intended to work with a hierarchical display (parent sheet expanding into child sheets) so it is disabled in a hierarchical Spread.

Standard Load on Demand

The standard mode only loads the next set of rows if there are rows that are hidden from the view. The default value for the **LoadOnDemandMode ('LoadOnDemandMode Property' in the on-line documentation)** property is standard.

Background Load on Demand

You can load new rows in the background before the last row is displayed. For example, if 20 rows are loaded and the user scrolls to row 15, the next set of rows is loaded. Set the **LoadOnDemandMode ('LoadOnDemandMode Property' in the on-line documentation)** property to Background to load the new rows before the last row is displayed. You can also specify whether to load the new rows using a time interval or when the scrolling is a specified number of rows from the bottom of the view. Use the **LoadOnDemandTriggerMode ('LoadOnDemandTriggerMode Property' in the on-line documentation)** property to specify whether to use a time interval or an offset from the bottom of the view.

The background load on demand allows other pending AJAX requests while rows are being loaded (rows are loaded without locking the Spread). The actions are put in a queue of pending requests to be processed later. When load on demand is finished, pending requests in the queue are processed until there are no more requests in the queue.

Virtual Paging

Another option for loading pages as the user scrolls vertically is the **AllowVirtualScrollPaging ('AllowVirtualScrollPaging Property' in the on-line documentation)** property. This can be used instead of the allow load on demand properties. The virtual paging feature will not work with load on demand. The **EnableClientScript ('EnableClientScript Property' in the on-line documentation)** property must be true for the virtual paging. For the best performance, you may also wish to set **EnableAjaxCall ('EnableAjaxCall Property' in the on-line documentation)** to true since the virtual paging uses Ajax calls.

The virtual scrolling option scrolls from the first row to the last row of the page size. If the row count is greater than the page size, then when you scroll past the maximum page size row, the next set of rows is loaded (a wait icon is displayed while the next page is loading in this case).

You can also display rows from the previous page with the **VirtualScrollPagingPrevRowCount ('VirtualScrollPagingPrevRowCount Property' in the on-line documentation)** property.

The scroll bar button size reflects the total number of rows with virtual scrolling (rather than the number of currently loaded rows).

Using the Properties Window

You can set several of the properties at design time using the Properties window of Visual Studio .NET or the Property grid in the designer.

1. Select the sheet.
2. Set the **AllowLoadOnDemand** property to true to allow loading on demand.
3. Set the **LoadInitRowCount** property to specify the initial number of rows to load.
4. Set the **LoadRowIncrement** property to specify how many rows to load after the initial set of rows is loaded.

Using Code

Use the **AllowLoadOnDemand ('AllowLoadOnDemand Property' in the on-line documentation)** property to allow load on demand. Set the **LoadInitRowCount ('LoadInitRowCount Property' in the on-line documentation)** property to load the initial set of rows. Set the **LoadRowIncrement ('LoadRowIncrement Property' in the on-line documentation)** property to specify the number of rows to load after the initial page is loaded.

Example

The height of the component should be smaller than the height of ten rows (for this example).

C#

```
FpSpread1.Sheets[0].RowCount = 40;
```

```
FpSpread1.Sheets[0].AllowLoadOnDemand = True;
FpSpread1.Sheets(0).PageSize = 40;
FpSpread1.Sheets[0].LoadInitRowCount = 10;
FpSpread1.Sheets[0].LoadRowIncrement = 10;
long i;
for (i = 1; i <= 20; i++)
{
FpSpread1.Sheets[0].Cells[i, 0].Value = i;
}
```

VB

```
FpSpread1.Sheets(0).RowCount = 40
FpSpread1.Sheets(0).AllowLoadOnDemand = True
FpSpread1.Sheets(0).PageSize = 40
FpSpread1.Sheets(0).LoadInitRowCount = 10
FpSpread1.Sheets(0).LoadRowIncrement = 10
Dim i As Long
For i = 1 To 20
FpSpread1.Sheets(0).Cells(i, 0).Value = i
Next
```

Customizing Interaction Based on Events

You can customize user interaction based on events in the FpSpread component. In the **FpSpread ('FpSpread Class' in the on-line documentation)** class there are several events, from **ButtonCommand ('ButtonCommand Event' in the on-line documentation)** to **InsertCommand ('InsertCommand Event' in the on-line documentation)** to **SaveOrLoadSheetState ('SaveOrLoadSheetState Event' in the on-line documentation)**. Use these events to initiate actions.

For a list of events with code samples, refer to the **FpSpread ('FpSpread Class' in the on-line documentation)** class in the **Assembly Reference (on-line documentation)**.

Handling the Tab Key

You can customize the use of the Tab key. By default the user can press the Tab key to advance the focus to the next active cell. You can turn off this behavior so that the component does not pay attention to the Tab key being pressed. To control this behavior, set the **ProcessTab ('ProcessTab Property' in the on-line documentation)** property of the **FpSpread ('FpSpread Class' in the on-line documentation)** class.

Using Code

Use the **ProcessTab ('ProcessTab Property' in the on-line documentation)** property.

Example

Set the **ProcessTab ('ProcessTab Property' in the on-line documentation)** property.

C#

```
FpSpread1.ProcessTab = false;
```

VB

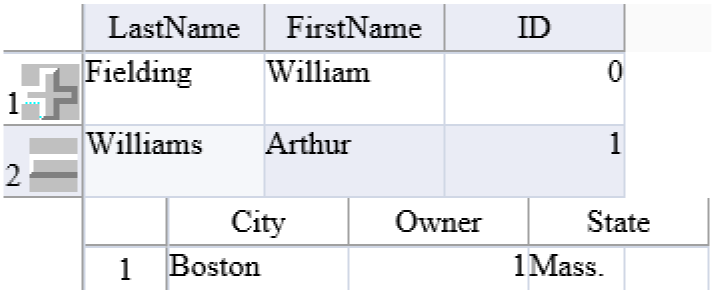
```
FpSpread1.ProcessTab = False
```

Customizing the Graphical Interface

You can customize the graphical user interface of the component by using your own graphics for certain parts of the interface. You can customize these aspects of the component:

- Sort indicator that is displayed in the column header
- Expand and collapse icons in the hierarchical display
- Icons in the filter bar

The following image displays custom icons in the hierarchical display.



	LastName	FirstName	ID
1	Fielding	William	0
2	Williams	Arthur	1

	City	Owner	State
1	Boston	1	Mass.

To display or hide the sort indicator, use the **SortIndicator ('SortIndicator Enumeration' in the on-line documentation)** enumeration settings and the **SortIndicator ('SortIndicator Property' in the on-line documentation)** property of the **Column ('Column Class' in the on-line documentation)** class.

Use the **GetImage ('GetImage Method' in the on-line documentation)** method and **SetImage ('SetImage Method' in the on-line documentation)** method in the **FpSpread ('FpSpread Class' in the on-line documentation)** component to work with the image. Use the **SpreadImages ('SpreadImages Enumeration' in the on-line documentation)** enumeration to specify the images to customize.

You can also manage whether users can expand rows to see child views. For more information, refer to **Handling Row Expansion**.

For more information on sorting, refer to **Customizing Sorting of Rows of User Data**.

Using Code

You can use the **SetImage ('SetImage Method' in the on-line documentation)** method to add your own image to the control. Specify which image to replace and the URL of the image with this method.

Example

The following example uses the **SetImage ('SetImage Method' in the on-line documentation)** method with a control that has been bound to a hierarchical data set.

C#

```
System.Data.DataSet ds = new System.Data.DataSet();
DataTable name;
DataTable city;
name = ds.Tables.Add("Customers");
name.Columns.AddRange(new DataColumn[] {new DataColumn("LastName", typeof(string)), new
DataColumn("FirstName", typeof(string)), new DataColumn("ID", typeof(Int32))});
name.Rows.Add(new object[] {"Fielding", "William", 0});
name.Rows.Add(new object[] {"Williams", "Arthur", 1});
name.Rows.Add(new object[] {"Zuchini", "Theodore", 2});
city = ds.Tables.Add("City/State");
city.Columns.AddRange(new DataColumn[] {new DataColumn("City", typeof(string)), new
DataColumn("Owner", typeof(Int32)), new DataColumn("State", typeof(string))});
```

```
city.Rows.Add(new object[] {"Atlanta", 0, "Georgia"});
city.Rows.Add(new object[] {"Boston", 1, "Mass."});
city.Rows.Add(new object[] {"Tampa", 2, "Fla."});
ds.Relations.Add("City/State", name.Columns["ID"], city.Columns["Owner"]);
FpSpread1.DataSource = ds;
FpSpread1.SetImage(FarPoint.Web.Spread.SpreadImages.Expand, "icon1.ico");
FpSpread1.SetImage(FarPoint.Web.Spread.SpreadImages.Collapse, "icon2.ico");
```

VB

```
Dim ds As New System.Data.DataSet
Dim name As DataTable
Dim city As DataTable
name = ds.Tables.Add("Customers")
name.Columns.AddRange(New DataColumn() {New DataColumn("LastName",
Type.GetType("System.String")), New DataColumn("FirstName",
Type.GetType("System.String")), New DataColumn("ID", Type.GetType("System.Int32"))})
name.Rows.Add(New Object() {"Fielding", "William", 0})
name.Rows.Add(New Object() {"Williams", "Arthur", 1})
name.Rows.Add(New Object() {"Zuchini", "Theodore", 2})
city = ds.Tables.Add("City/State")
city.Columns.AddRange(New DataColumn() {New DataColumn("City",
Type.GetType("System.String")), New DataColumn("Owner", Type.GetType("System.Int32")),
New DataColumn("State", Type.GetType("System.String"))})
city.Rows.Add(New Object() {"Atlanta", 0, "Georgia"})
city.Rows.Add(New Object() {"Boston", 1, "Mass."})
city.Rows.Add(New Object() {"Tampa", 2, "Fla."})
ds.Relations.Add("City/State", name.Columns("ID"), city.Columns("Owner"))
FpSpread1.DataSource = ds
FpSpread1.SetImage(FarPoint.Web.Spread.SpreadImages.Expand, "icon1.ico")
FpSpread1.SetImage(FarPoint.Web.Spread.SpreadImages.Collapse, "icon2.ico")
```

Searching for Data with Code

To search for data in any of the cells of a sheet, use either of these sets of methods in the FpSpread class:

- **Search ('Search Method' in the on-line documentation)** methods
- **SearchHeaders ('SearchHeaders 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.

Using Code

Use the **Search ('Search Method' in the on-line documentation)** method for the FpSpread component to perform a search.

Example

Use the Search method to perform an exact-match search on the third sheet (Sheet 2) for the word "Total" and return the values of the row index and column index of the found cell.

C#

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

VB

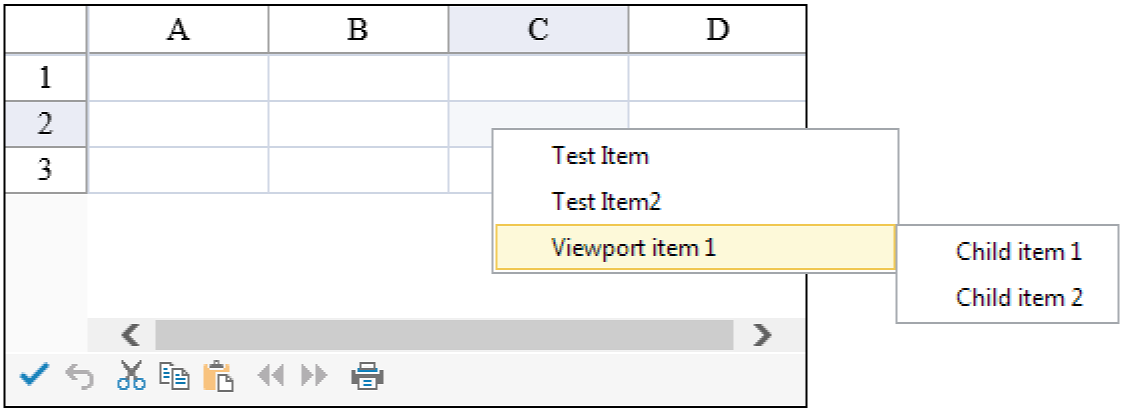
```
FpSpread1.Search(2,"Total",True,True,False,False,1,1,56,56,ref rowindx,ref colindx))
```

Adding a Context Menu

You can display a Spread context menu when right-clicking on the Spread control. The context menu can be displayed when the user right-clicks on the column header, row header, or viewport area (data area and empty area).

You can add menu items to the menu and set the height or other properties. Specify the type of menu with the **ContextMenuType ('ContextMenuType Enumeration' in the on-line documentation)** enumeration. You can create a menu using markup code, the **ContextMenus ('ContextMenus Property' in the on-line documentation)** property in the **Properties** window, or server code.

The **CommandArgument ('CommandArgument Property' in the on-line documentation)** and **CommandName ('CommandName Property' in the on-line documentation)** properties are used to separate which menu item is clicked in code. So in the **MenuItemClicked ('MenuItemClicked Event' in the on-line documentation)** event on the server side you could add code such as switch(eventArgs.SelectedItem.CommandName) or switch(eventArgs.SelectedItem.CommandArgument).



Using the Properties Window

1. In the **Properties** windows select **Spread**.
2. Under the **Behavior** section select the **ContextMenus** property.
3. Use the **ContextMenu Collection** editor to add menus, menu items, and set any menu properties.
4. Click **OK** when finished.

Using Code

1. Create a viewport menu using markup or the **ContextMenus ('ContextMenus Property' in the on-line documentation)** property in the property window at design time.
2. Set the **EnableContextMenu ('EnableContextMenu Property' in the on-line documentation)** property to true.
3. Create a row header menu with code.

Example

This example code creates one menu at design time and one menu at run time.

Code

```
//Markup code
<ContextMenus>
        <FarPoint:ContextMenu Type="Viewport">
          <Items>
            <FarPoint:MenuItem Enabled="True" ImageUrl="http://linktoimagehere/abc.jpc"
Text="Menu item 1">
                <ItemTemplate>
                <asp:TextBox ID="bac" runat="server" />
                </ItemTemplate>
            </FarPoint:MenuItem>
            <FarPoint:MenuItem Text="Sort" ImageUrl="http://linktoimagehere/abc.jpc">
                <ChildItems >
                  <FarPoint:MenuItem  Text="Child Item1"
ImageUrl="http://avc/abc.jpc"></FarPoint:MenuItem>
                  <FarPoint:MenuItem Text="Child Item2"></FarPoint:MenuItem>
                </ChildItems>
            </FarPoint:MenuItem>
            <FarPoint:MenuItem Enabled="True"
ImageUrl="http://linktoimagehere/abc.jpc">Menu item 3</FarPoint:MenuItem>
          </Items>
        </FarPoint:ContextMenu>
      </ContextMenus>
```

C#

```
protected void Page_Load(object sender, System.EventArgs e)
{
if (this.IsPostBack) return;
FpSpread1.EnableContextMenu = true;
//Create this viewport menu using markup or the ContextMenus property in the property
window
FarPoint.Web.Spread.ContextMenu viewportMenu =
FpSpread1.ContextMenus[FarPoint.Web.Spread.ContextMenuType.Viewport];
FarPoint.Web.Spread.MenuItem customViewportItem = new
FarPoint.Web.Spread.MenuItem("Viewport item 1");
customViewportItem.ChildItems.Add(new FarPoint.Web.Spread.MenuItem("Child item 1"));
customViewportItem.ChildItems.Add(new FarPoint.Web.Spread.MenuItem("Child item 2"));
viewportMenu.Items.Add(customViewportItem);

//This row header menu is created here (no markup or design properties)
FarPoint.Web.Spread.ContextMenu rowHeaderContextMenu = new
FarPoint.Web.Spread.ContextMenu();
rowHeaderContextMenu.Type = FarPoint.Web.Spread.ContextMenuType.RowHeader;
FarPoint.Web.Spread.MenuItem rowHeaderItem = new
FarPoint.Web.Spread.MenuItem("RowHeader item 1");
rowHeaderItem.ChildItems.Add(new FarPoint.Web.Spread.MenuItem("Child item 1"));
rowHeaderItem.ChildItems.Add(new FarPoint.Web.Spread.MenuItem("Child item 2"));
rowHeaderContextMenu.Items.Add(rowHeaderItem);
FpSpread1.ContextMenus.Add(rowHeaderContextMenu);
}
```

VB

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
If (IsPostBack) Then
```

```
    Return
End If
FpSpread1.EnableContextMenu = True
'Create this viewport menu using markup or the ContextMenus property in the property
window
Dim viewportMenu As FarPoint.Web.Spread.ContextMenu =
FpSpread1.ContextMenus(FarPoint.Web.Spread.ContextMenuType.Viewport)
Dim customViewportItem As New FarPoint.Web.Spread.MenuItem("Viewport item 1")
customViewportItem.ChildItems.Add(New FarPoint.Web.Spread.MenuItem("Child item 1"))
customViewportItem.ChildItems.Add(New FarPoint.Web.Spread.MenuItem("Child item 2"))
viewportMenu.Items.Add(customViewportItem)

'This row header menu is created here (no markup or design properties)
Dim rowHeaderContextMenu As New FarPoint.Web.Spread.ContextMenu()
rowHeaderContextMenu.Type = FarPoint.Web.Spread.ContextMenuType.RowHeader
Dim rowHeaderItem As New FarPoint.Web.Spread.MenuItem("RowHeader item 1")
rowHeaderItem.ChildItems.Add(New FarPoint.Web.Spread.MenuItem("Child item 1"))
rowHeaderItem.ChildItems.Add(New FarPoint.Web.Spread.MenuItem("Child item 2"))
rowHeaderContextMenu.Items.Add(rowHeaderItem)
FpSpread1.ContextMenus.Add(rowHeaderContextMenu)
End Sub
```

Working with AJAX

You can use AJAX and ASP.NET AJAX to extend the capability of Spread by providing additional cell types and page refresh options. You may perform the following tasks:

- **Enabling AJAX Support**
- **Using ASP.NET AJAX Extenders**

Enabling AJAX Support

AJAX allows the component to refresh without refreshing the entire page. You can add AJAX support to the FpSpread component by setting the **EnableAjaxCall ('EnableAjaxCall Property' in the on-line documentation)** property. Setting this property to True prevents Spread from doing a full page postback when implementing the following features - expanding and collapsing child sheets in a hierarchical display, column sorting, inserting rows, or paging.

If the **ClientAutoCalculation ('ClientAutoCalculation Property' in the on-line documentation)** property is true, then after a cell value is changed, an AJAX call is made to the FpSpread component. Then the component calculates the formulas and sends the values to the client side. The component then updates the values at the client side.

Use the **EnableAjaxCall ('EnableAjaxCall Property' in the on-line documentation)** property to enable AJAX support or use EnableAjaxCall and the **ClientAutoCalculation ('ClientAutoCalculation Property' in the on-line documentation)** properties to enable AJAX support of formulas.

Using Code

You can set the **EnableAjaxCall ('EnableAjaxCall Property' in the on-line documentation)** property and the **ClientAutoCalculation ('ClientAutoCalculation Property' in the on-line documentation)** property in code.

Example

The following code allows AJAX support and AJAX support of formulas.

C#

```
FpSpread1.EnableAjaxCall = true;
FpSpread1.ClientAutoCalculation = true;
```

VB

```
FpSpread1.EnableAjaxCall = True
FpSpread1.ClientAutoCalculation = True<
```

Using the Spread Designer

1. Select the **Settings** menu.
2. Select the **General** icon under the **Spread Settings** section.
3. Select the **General** tab and check the **Enable AJAX Call** box.
4. Select the **Edit** tab in order to set the **Client Auto Calculation** check box.
5. Click **OK**.
6. Click **Apply and Exit** to close the Spread Designer.

Using ASP.NET AJAX Extenders

You can use the many cell types in the FarPoint.Web.Spread.Extender assembly to provide controls that are available as ASP.NET AJAX extender controls. The extender controls enhance the client capabilities of other controls.

For more information about extender cell types, refer to **Working with ASP.NET AJAX Extender Cell Types**.

The various cell types that use ASP.NET AJAX extender controls include:

- **Setting an Automatic-Completion Cell**
- **Setting a Calendar Cell**
- **Setting a Filtered Text Cell**
- **Setting a Masked Edit Cell**
- **Setting a Mutually Exclusive Check Box Cell**
- **Setting a Numeric Spin Cell**
- **Setting a Rating Cell**
- **Setting a Slider Cell**
- **Setting a Slide Show Cell**
- **Setting a Text Box with Watermark Cell**

Using the AJAX Extenders

1. The current AJAX toolkit and setup is available at https://ajaxcontroltoolkit.codeplex.com/.
2. After installing the setup, add references to the new AjaxControlToolkit.dll files.

> If you are using Ajax Control Toolkit 15.1.2, check http://www.nuget.org/packages/AjaxControlToolkit/ for information about adding the references to the project.

> Check the https://ajaxcontroltoolkit.codeplex.com/ web site for the current location of the dll if you are using Ajax Control Toolkit 15.1 or older.

> Earlier versions of the AJAX toolkit are available at the Microsoft ASP.NET AJAX Controls page (http://ajax.asp.net/ajaxtoolkit/) and require downloading the zip file with the AJAX Control extenders and adding the references.

3. To use the AJAX Extender CellType, add a Script Manager to the page. From the Toolbox, under the AJAX Extenders category select Script Manager and drag it to the ASPX page (Web Form) where you have the Spread component.

> The AJAX Control Toolkit 15.1 stopped supporting ToolkitScriptManager and now supports the standard ScriptManager. For more information, refer to https://ajaxcontroltoolkit.codeplex.com.

Spread supports multiple versions of AJAX so the oldest version is used in the development environment. If you use AJAX Control Toolkit 15.1 when deploying to a server, the following configuration information about assembly binding must be added so that the web server loads the correct version.

Web.config

```
<configuration>
    <system.web>
    </system.web>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="AjaxControlToolkit" publicKeyToken="28f01b0e84b6d53e"
culture="neutral"/>
        <bindingRedirect oldVersion="3.0.30930.28736" newVersion="15.1.2.0" />
      </dependentAssembly>
    </assemblyBinding>
  </runtime>
</configuration>
```

Customizing the Tool Bars

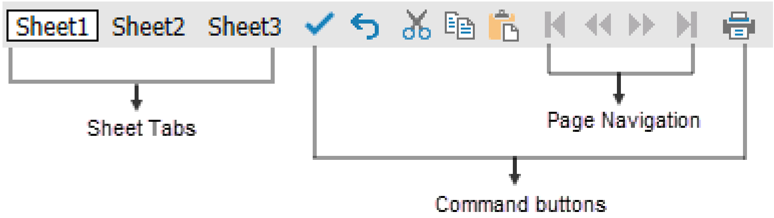
There are a set of tasks that allow you to customize how the user can interact with the spreadsheet, namely the parts of the component that involve navigation and buttons in the tool bars that FpSpread displays above and below any sheet of data. Not all of the tool bars are displayed automatically; some are optional. The tasks involved in customizing the various tool bars include:

- **Customizing the Command Bar on the Component**
- **Customizing the Command Buttons**
- **Changing the Command Button Images**
- **Hiding a Specific Command Button**
- **Working with the SaveExcel button on the CommandBar (on-line documentation)**
- **Displaying the Sheet Names**
- **Customizing Page Navigation**
- **Customizing Page Navigation Buttons on the Client**
- **Customizing the Hierarchy Bar**

For information about the scroll bars, refer to **Displaying Scroll Bars**. For information about the graphical user interface, refer to **Customizing the Graphical Interface**.

Customizing the Command Bar on the Component

The command bar is a tool bar that is displayed at the top or bottom of the component. This bar includes the sheet name tabs (if there is more than one sheet) and the command buttons. By default, page navigation aids are also displayed on the command bar but can be repositioned or not displayed on the command bar.



Customizations

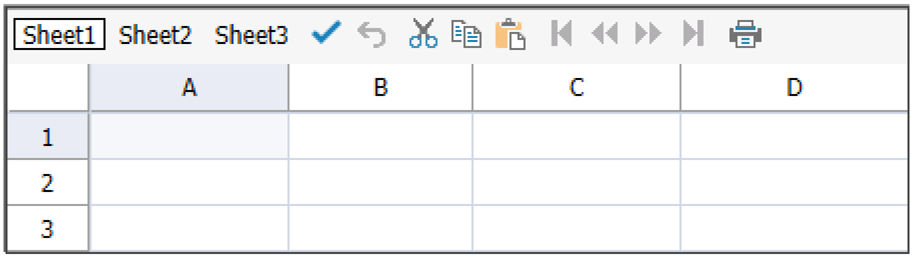
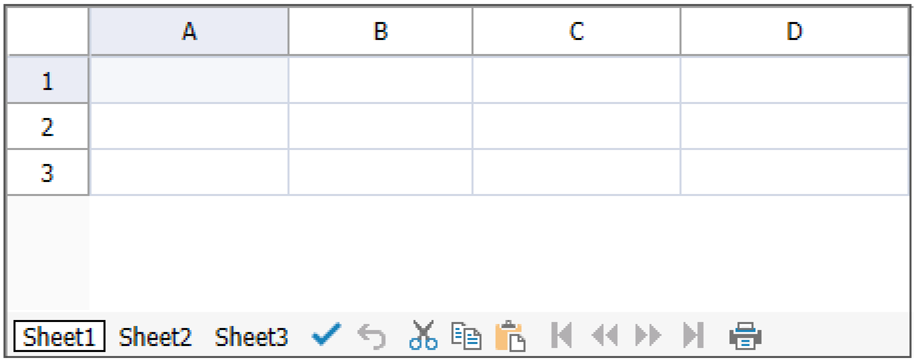
The customizations are made possible with the **CommandBarInfo ('CommandBarInfo Class' in the on-line documentation)** object. You can do the following customizations:

- Customize the color of the command bar
- Customize the font style of the text in the sheet name tabs and buttons
- Hide the command bar if there is only one sheet
- Set the position of the command bar to be either at the top or bottom of the component

To set the color of the command bar, use the **BackColor ('BackColor Property' in the on-line documentation)** property. To change the font of the text that appears in the buttons, use the **Font ('Font Property' in the on-line documentation)** property.

To hide the command bar when there is only one sheet (and thus no sheet name tabs to display), use the **Visible ('Visible Property' in the on-line documentation)** property.

By default, the command bar is displayed at the bottom of the component. You can display it at the top by setting the **CommandBarOnBottom ('CommandBarOnBottom Property' in the on-line documentation)** property in the **FpSpread ('FpSpread Class' in the on-line documentation)** class.

Command Bar Position	Example Spread
On Top	
On Bottom	

Postbacks

Several, but not all, of the buttons in the command bar trigger a postback to the server. These include:

- Delete button (AJAX postback)
- Print button
- Pager buttons (AJAX postback)
- Sheet buttons (AJAX postback)
- Update button (AJAX postback)

For information on other aspects of the appearance of the command bar buttons, refer to **Customizing the Command Buttons**.

Using the Properties Window

1. Select the FpSpread component.
2. With the **Properties** window open, select the **CommandBar** property drop-down list, and set any of the command bar properties. In order to set the BackColor setting for the command bar, set the **Enable** property to **False** in the **Background** section.

Using Code

Use the properties of the **FpSpread ('FpSpread Class' in the on-line documentation)** class to define the position of the command bar and use the command bar properties to customize the look of the command bar.

Example

In this example, set the command bar to display at the top of the component and set the color to yellow.

C#

```
FpSpread1.CommandBarOnBottom = false;
FpSpread1.CommandBar.Background = null;
FpSpread1.CommandBar.BackColor = Color.Yellow;
```

VB

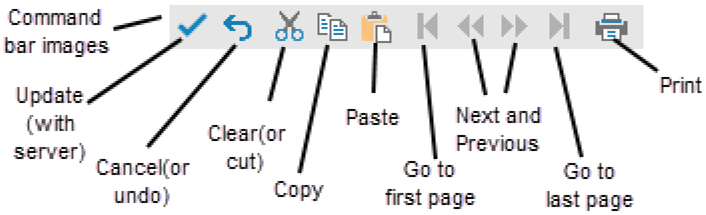
```
FpSpread1.CommandBarOnBottom = False
FpSpread1.CommandBar.Background = Nothing
FpSpread1.CommandBar.BackColor = Color.Yellow
```

Using the Spread Designer

1. Select the **Settings** menu.
2. Select the **Command Bar** icon under the **Spread Settings** section (the property grid has additional settings that are not available in the designer).
3. Select the various options.
4. Click **Apply** and **OK**.
5. Click **Apply and Exit** to close the Spread Designer.

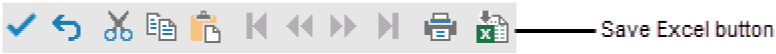
Customizing the Command Buttons

You can customize how (and if) the command buttons are shown in the command bar. The figure below shows the default display of the command buttons using the images (or icon) type of display.



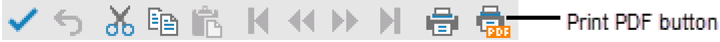
The various settings of the command bar are handled using the **CommandBarInfo ('CommandBarInfo Class' in the on-line documentation)** class.

- You can display the command buttons as images (also called icons), push buttons (with text), or links (with text). The default is to display them as images. This is the type of buttons displayed. You can also specify themes for the button images.
- You can change the appearance of the text of the command buttons (if you are using the button type) using the **Font ('Font Property' in the on-line documentation)** property.
- You can change the appearance of the command buttons using various button properties.
- You can display or not display the buttons on the command bar using the **Visible ('Visible Property' in the on-line documentation)** property.
- You can change the images (as described in **Changing the Command Button Images**)
- You can hide or show the Save Excel button on the CommandBar to quickly export your spreadsheets to Excel (using the **ShowExcelButton ('ShowExcelButton Property' in the on-line documentation)** property in the **CommandBarInfo ('CommandBarInfo Class' in the on-line documentation)** class).



For more information, please refer to **Working with the SaveExcel button on the CommandBar (on-line documentation)**.

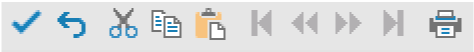
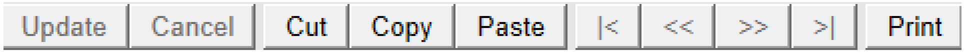
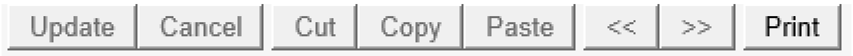
- You can hide or show a PDF button for printing to PDF (using the **ShowPDFButton ('ShowPDFButton Property' in the on-line documentation)** property in the **CommandBarInfo ('CommandBarInfo Class' in the on-line documentation)** class).



Which command buttons appear in the command bar change if you set the **EnableClientScript ('EnableClientScript Property' in the on-line documentation)** property to false for the component. Fewer buttons are displayed, due to the limitations of not providing scripting on the client.

Command Bar Button Types

The type of buttons displayed in the command bar can be any of these types:

Button Type	Enumeration Setting	Typical Default Display
Images (or icons)	ImageButton	
Text links	LinkButton	
Push-buttons	PushButton	

Refer to the **ButtonType ('ButtonType Enumeration' in the on-line documentation)** enumeration and the

ButtonType ('ButtonType Property' in the on-line documentation) property. Note that the buttons appear grayed out or inactive until they can be used. The figures shown above show all the buttons active. The link type command button option requires that the client-side scripting be disabled (**FpSpread ('FpSpread Class' in the on-line documentation)** class, **EnableClientScript ('EnableClientScript Property' in the on-line documentation)** property set to false). Client-side operations such as cut, copy, and paste are not available when scripting is disabled so buttons for those operations do not appear on the command bar.

The properties that affect the appearance of the buttons include these:

CommandBarInfo Property	Appearance Description
ButtonFaceColor ('ButtonFaceColor Property' in the on-line documentation)	the background color of the buttons
ButtonHighlightColor ('ButtonHighlightColor Property' in the on-line documentation)	the color of the top and left outline of the buttons
ButtonShadowColor ('ButtonShadowColor Property' in the on-line documentation)	the color of the bottom and right outline of the buttons
Font ('Font Property' in the on-line documentation)	the color of text in the buttons

Remember that to update the data on the server, changes must be saved from the client. Changes from the client can be saved either by using the **SaveChanges ('SaveChanges Method' in the on-line documentation)** method in code or by having the user click the Update button on the command bar.

For other ways to customize parts of the command bar, refer to **Customizing the Command Bar on the Component**.

Using the Properties Window

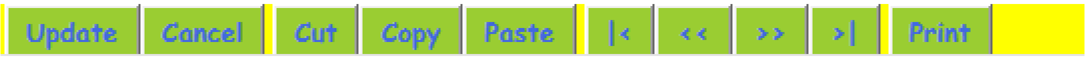
1. Select the FpSpread component.
2. With the **Properties** window open, select the **CommandBar** property drop-down list, and set any of the button properties.

Using Code

Use the properties of the **CommandBarInfo ('CommandBarInfo Class' in the on-line documentation)** class to define the appearance of the buttons.

Example

In this example, some of the button properties of the command bar are set. The result is shown in this figure.



C#

```
FpSpread1.CommandBar.Background = null;
FpSpread1.CommandBar.BackColor = Color.Yellow;
FpSpread1.CommandBar.ButtonFaceColor = Color.YellowGreen;
FpSpread1.CommandBar.ButtonTextColor = Color.RoyalBlue;
FpSpread1.CommandBar.ButtonType = FarPoint.Web.Spread.ButtonType.PushButton;
FpSpread1.CommandBar.Font.Bold = true;
FpSpread1.CommandBar.Font.Name = "Comic Sans MS";
FpSpread1.CommandBar.Visible = true;
```

VB

```
FpSpread1.CommandBar.Background = Nothing
```

```
FpSpread1.CommandBar.BackColor = Color.Yellow
FpSpread1.CommandBar.ButtonFaceColor = Color.YellowGreen
FpSpread1.CommandBar.ButtonTextColor = Color.RoyalBlue
FpSpread1.CommandBar.ButtonType = FarPoint.Web.Spread.ButtonType.PushButton
FpSpread1.CommandBar.Font.Bold = True
FpSpread1.CommandBar.Font.Name = "Comic Sans MS"
FpSpread1.CommandBar.Visible = True
```

Using the Spread Designer

1. Select **Spread** in the **Property** grid.
2. Select the CommandBar property drop-down list, and set any of the button properties.
3. Click **Apply** and **OK**.
4. Click **Apply and Exit** to close the Spread Designer.

Note: While you set any of the button properties above, make sure that the **UseSheetSkin ('UseSheetSkin Property' in the on-line documentation)** property of the **CommandBarInfo Class (on-line documentation)** is set to false.

Changing the Command Button Images

You can change the images used for the buttons in the command bar. By default, the command buttons are displayed as images (or icons) since the **ButtonType ('ButtonType Property' in the on-line documentation)** property in the **CommandBarInfo ('CommandBarInfo Class' in the on-line documentation)** class is set to ImageButton by default. You can change the images by providing replacement images or by adding your own buttons in code. In addition, you can change the buttons by setting the **Theme ('Theme Property' in the on-line documentation)** property.

You can put images of any size in the command bar; the only limit to the size is the size of the command bar.

You can change the existing images by replacing them in the images subdirectory of the fp_client folder.

For information on other aspects of the appearance of the command bar buttons, refer to **Customizing the Command Buttons**.

Using Code

Use the properties of the **CommandBarInfo ('CommandBarInfo Class' in the on-line documentation)** class to define the appearance of the buttons.

Example

In this example, the default images are changed to XP theme images.

C#

```
FpSpread1.Sheets[0].RowCount = 20;
FpSpread1.CommandBar.ButtonType = FarPoint.Web.Spread.ButtonType.ImageButton;
FpSpread1.CommandBar.Theme = FarPoint.Web.Spread.ImageButtonTheme.Xp;
```

VB

```
FpSpread1.Sheets(0).RowCount = 20
FpSpread1.CommandBar.ButtonType = FarPoint.Web.Spread.ButtonType.ImageButton
FpSpread1.CommandBar.Theme = FarPoint.Web.Spread.ImageButtonTheme.Xp
```

Using Code

Change the image for the **Print** button using the **CreateButton ('CreateButton Event' in the on-line documentation)** event.

Example

In this example, the print button image is changed.

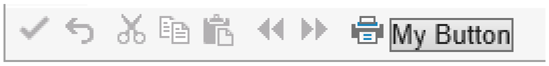
C#

```
private void FpSpread1_CreateButton(object sender,
FarPoint.Web.Spread.CreateButtonEventArgs e)
{
if (e.Command == "Print")
{
e.EnabledImgUrl = "happy.bmp";
}
}
```

VB

```
Protected Sub FpSpread1_CreateButton(ByVal sender As Object, ByVal e As
FarPoint.Web.Spread.CreateButtonEventArgs) Handles FpSpread1.CreateButton
If e.Command = "Print" Then
e.EnabledImgUrl = "happy.bmp"
End If
End Sub
```

Using Code



1. You can also create your own buttons with code as displayed by the above image.
2. Override the **Render** event.
3. Create a new table cell.
4. Create a button control and set the button properties.
5. Add the button to the table cell.

Example

In this example, add the My Button button.

C#

```
protected override void Render(System.Web.UI.HtmlTextWriter writer)
{
Control updateBtn = FpSpread1.FindControl("Update");
if ((updateBtn != null))
{
TableCell tc = (TableCell)updateBtn.Parent;
TableRow tr = (TableRow)tc.Parent;
TableCell tc1 = new TableCell();
tr.Cells.Add(tc1);
Button btn = new Button();
btn.CausesValidation = false;
btn.Text = "My Button";
btn.Attributes.Add("onclick", "javascript:" +
this.Page.GetPostBackEventReference(FpSpread1, "my command") + "; return false;");
tc1.Controls.Add(btn);
}
base.Render(writer);
}
```

VB

```
Protected Overrides Sub Render(ByVal writer As System.Web.UI.HtmlTextWriter)
       Dim updateBtn As Control = FpSpread1.FindControl("Update")
       If Not updateBtn Is Nothing Then

            Dim tc As TableCell = updateBtn.Parent
            Dim tr As TableRow = tc.Parent

            Dim tc1 As New TableCell()
            tr.Cells.Add(tc1)

            Dim btn As New Button()
            btn.CausesValidation = False
            btn.Text = "My Button"
            btn.Attributes.Add("onclick", "javascript:" +
Me.Page.GetPostBackEventReference(FpSpread1, "my command") + "; return false;")
            tc1.Controls.Add(btn)
       End If

       MyBase.Render(writer)
End Sub
```

You can process the button command by adding an event handler to the **ButtonCommand ('ButtonCommand Event' in the on-line documentation)** event.

C#

```
private void FpSpread1ButtonCommand(object sender,
FarPoint.Web.Spread.SpreadCommandEventArgs e)
{
}
```

VB

```
Private Sub FpSpread1_ButtonCommand(ByVal sender As Object, ByVal e As
  FarPoint.Web.Spread.SpreadCommandEventArgs) HandlesFpSpread1.ButtonCommand
End Sub
```

Hiding a Specific Command Button

You can customize the display of the command buttons in the command bar by hiding any or all of the command buttons.

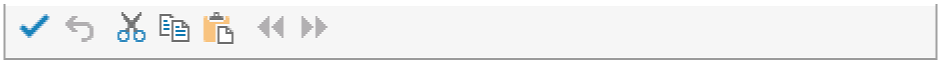
To hide a command button, set the **Visible ('Visible Property' in the on-line documentation)** property to false in the event that creates the button.

Using Code

Use the **CreateButton ('CreateButton Event' in the on-line documentation)** event to hide certain buttons.

Example

This example uses code to hide the print icon by adding the **Visible ('Visible Property' in the on-line documentation)** property to the **CreateButton ('CreateButton Event' in the on-line documentation)** event. The result is shown in this figure.



C#

```
private void FpSpread1_CreateButton(object sender,
FarPoint.Web.Spread.CreateButtonEventArgs e)
{
    if (e.Command == "Print")
    {
        e.Visible = false;
    }
}
```

VB

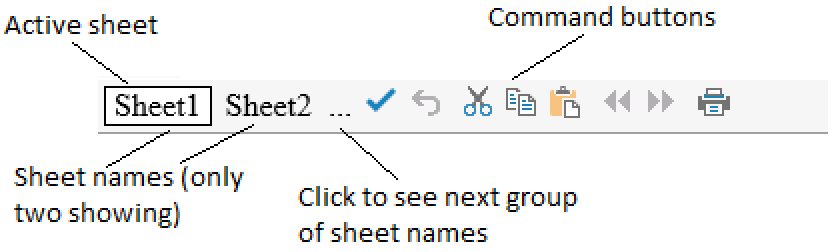
```
Private Sub FpSpread1CreateButton(ByVal sender As Object, ByVal e As
FarPoint.Web.Spread.CreateButtonEventArgs) Handles FpSpread1.CreateButton
If e.Command = "Print" Then
e.Visible = False
End If
End Sub
```

Displaying the Sheet Names

You can customize how and if to display the sheet names in the bar at the bottom of the component. Since a component may have more than one sheet, the tabs (or buttons) in the command bar contain the sheet names and provide a way to navigate to different sheets. These are called sheet name tabs. The default sheet names are Sheet0, Sheet1, etc. You can specify other names for the sheets and these appear in the sheet name tabs. By default, the component has only one sheet and so no sheet name tabs are displayed. When you add a sheet, as described in **Adding a Sheet**, the sheet name tabs are added to the command bar for display.

You can set how many sheet name tabs are displayed. If the number of tabs exceeds the value specified, an ellipses is displayed. Click the ellipses to display the next (or previous) set of sheet names. You can also set the increment for advancing the sheet names. Be sure not to set the increment bigger than the number displayed if you want to be able to see all the sheet name tabs. You can set which sheet number displays first. These are all properties of the **TabInfo ('TabInfo Class' in the on-line documentation)** class.

> **Note:** The sheet changes when you click a different sheet name tab or when you click on the ellipses. When you click on the ellipses, the lowest number sheet in the set of sheet names is displayed.


Active sheet
Command buttons
Sheet1
Sheet2
...
Sheet names (only two showing)
Click to see next group of sheet names

Using Code

1. To define when the sheet name tabs are displayed, use the **TabControlPolicy ('TabControlPolicy Property' in the on-line documentation)** property of the **TabInfo ('TabInfo Class' in the on-line documentation)** class and the settings of the **TabControlPolicy ('TabControlPolicy Enumeration' in the on-line documentation)** enumeration.
2. Determine the various settings using the **Tab ('Tab Property' in the on-line documentation)** property of the **FpSpread ('FpSpread Class' in the on-line documentation)** component. Determine how many sheet name tabs to display using the **VisibleCount ('VisibleCount Property' in the on-line documentation)** property, how many to increment with the **ScrollIncrement ('ScrollIncrement Property' in the on-line documentation)** property, and which is the first visible sheet name with the **FirstVisibleTab ('FirstVisibleTab Property' in the on-line documentation)** property of the **TabInfo ('TabInfo Class' in the on-line documentation)** class.
3. Determine the appearance of the sheet name tabs, such as the background color, the text color, and the text of the sheet name using the properties of the **TabInfo ('TabInfo Class' in the on-line documentation)** class for the **Tab ('Tab Property' in the on-line documentation)** property of the component.

Example

In this example, set the sheet name tabs to always appear below the sheet at the bottom of the component and show only two sheet names (two tabs) and set the background color of the active tab to green.

C#

```
FpSpread1.Sheets.Count = 3;
FpSpread1.Tab.TabControlPolicy = FarPoint.Web.Spread.TabControlPolicy.Always;
FpSpread1.Tab.VisibleCount = 2;
FpSpread1.Tab.ScrollIncrement = 2;
FpSpread1.Tab.FirstVisibleTab = 1;
FpSpread1.Tab.TextColor = Color.Yellow;
FpSpread1.Tab.ActiveTabBackColor = Color.Green;
FpSpread1.Tab[0] = "First";
FpSpread1.Tab[1] = "Second";
FpSpread1.Tab[2] = "Third";
```

VB

```
FpSpread1.Tab.TabControlPolicy = FarPoint.Web.Spread.TabControlPolicy.Always
FpSpread1.Sheets.Count = 3
FpSpread1.Tab.VisibleCount = 2
FpSpread1.Tab.ScrollIncrement = 2
FpSpread1.Tab.FirstVisibleTab = 1
FpSpread1.Tab.ActiveTabBackColor = Color.Green
FpSpread1.Tab.TextColor = Color.Yellow
FpSpread1.Tab(0) = "First"
FpSpread1.Tab(1) = "Second"
FpSpread1.Tab(2) = "Third"
```

Using the Spread Designer

1. Select the **Settings** menu.
2. Select the **General** icon under the **Spread Settings** section.
3. Select **Tab** and specify the policy settings.
4. Click **OK**.
5. Click **Apply and Exit** to close the Spread Designer.

Customizing Page Navigation

A page is the amount of data in a sheet that can be displayed at one time. This is not the same as an HTML page. When the sheet contains more rows than can be displayed in the component, Spread automatically creates pages that group the rows and allows you to navigate between the pages of the sheet. For sheets that have more rows than fit in the display area, the sheet has multiple pages. For example, for a sheet that has 50 rows, you may want to display only 10 rows at a time, so each page would be a set of 10 rows.

You can advance through these pages using the page navigation buttons that are available at the edge of the component. These include next (right arrow) and previous (left arrow) buttons as well as page numbers. You can determine which of these buttons are displayed by the component and where on the component they are displayed. Customizing these page navigation aids is done with the properties of the **PagerInfo ('PagerInfo Class' in the on-line documentation)** class and the **Pager ('Pager Property' in the on-line documentation)** property of the **FpSpread ('FpSpread Class' in the on-line documentation)** class.

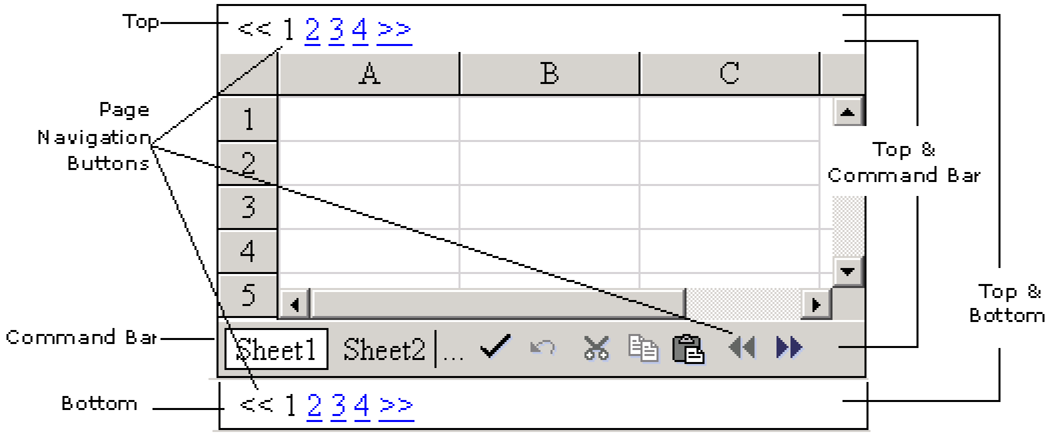
You can customize the page navigation in the following ways:

- Appearance (mode) of the navigation aids
- Type of buttons (image, link, or push) and image button themes
- Position of the navigation aids on the component; in which tool bars they appear
- Count of how many page numbers are displayed as part of the navigation aids
- Alignment of the navigation aids on the top or bottom bars
- Color of the background and color of the text in the bars with the navigation aids
- Font of the navigation aids in the bars
- Labels (text) of the navigation aids in the command bar

Mode: You can display either next (>>) and previous (<<) arrows, page numbers, or both as page navigation aids. Clicking on the next and previous arrows has the same effect as clicking on the corresponding page number: advancing through the pages to see the set of rows for that page.

Position: You can display these page navigation aids at the top of the sheet, the bottom of the sheet, on the command bar, or some combination of these. The page numbers do not appear in the command bar, only the next and previous arrows.

This figure illustrates the various optional placements of page navigation buttons with the default font and alignment:



Note that you cannot display the page navigation in all these positions at once, but you can display them at the top and bottom or top and command bar at the same time. The illustration serves to show the possible placements in one diagram.

Numbers: For sheets that consist of many pages, you can set how many page numbers are displayed. If the number of pages for a sheet exceeds the value specified by the PageCount property, an ellipses (...) is displayed. The user clicks the ellipses to display the next (or previous) set of page numbers.

Alignment: You can set the alignment of the navigation aids when they appear in the top or bottom bar or both. When right aligned, for example, the navigation aids appear on the right side of the bar(s). This does not affect the display in the command bar.

Colors: You can set both the background color of the page navigation aids and the text color (foreground color) for display in the top or bottom bar or both. This does not affect the display in the command bar.

Font: You can set the font, when the navigation aids are displayed in the top or bottom bar or both. This does not affect the display in the command bar.

Labels: You can customize the labels of these buttons on the client using the CreateButton event. For more information, refer to **Customizing Page Navigation Buttons on the Client**.

For more information on setting the size of the page, refer to **Customizing the Page Size (Rows to Display)**.

Using the Properties Window

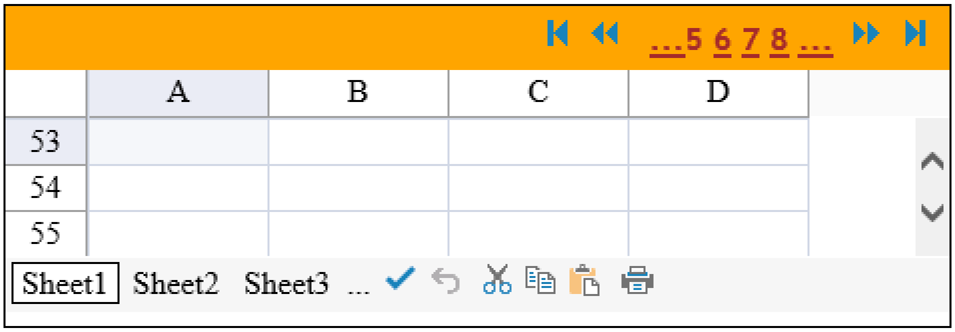
1. Select the FpSpread component.
2. Select the **Pager** object and set properties as needed.
3. Select the **Sheets** collection and set properties such as **PageSize** and **RowCount**.

Using Code

1. Use the **PagerInfo ('PagerInfo Class' in the on-line documentation)** class to set the properties of the **Pager ('Pager Property' in the on-line documentation)** property for the **FpSpread ('FpSpread Class' in the on-line documentation)** component.
2. Specify what part of the page navigation to display by setting the **Mode ('Mode Property' in the on-line documentation)** property of the **PagerInfo ('PagerInfo Class' in the on-line documentation)** class with the settings of the **PagerMode ('PagerMode Enumeration' in the on-line documentation)** enumeration.
3. Specify where to display the page navigation aids by setting the **Position ('Position Property' in the on-line documentation)** property with the settings of the **PagerPosition ('PagerPosition Enumeration' in the on-line documentation)** enumeration and how many page numbers to display by setting the **PageCount ('PageCount Property' in the on-line documentation)** property.
4. Specify the appearance of the page navigation aids by setting the font and color.

Example

In this example, set the page navigation buttons to appear at the top of the component on a separate tool bar above the sheet (and not on the tool bars below the sheet). Display both the page numbers and the page arrows with the specified font and colors and display them on the right side of that top bar, as shown in the figure.



C#

```
// Set the number of sheets.
FpSpread1.Sheets.Count = 5;
// Set the number of rows on the first sheet
FpSpread1.Sheets[0].RowCount = 136;// Set the number of rows per page in this sheet.
FpSpread1.Sheets[0].PageSize = 13;

// Display the pager only at the top of the component.
FpSpread1.Pager.Position = FarPoint.Web.Spread.PagerPosition.Top;
// Display both numbers and arrows by setting mode.
// Set the mode after the position, otherwise an error.
FpSpread1.Pager.Mode = FarPoint.Web.Spread.PagerMode.Both;
// Format the text in the pager at the top.
FpSpread1.Pager.Align = HorizontalAlign.Right;
FpSpread1.Pager.Font.Bold = true;
FpSpread1.Pager.Font.Name = "Trebuchet MS";
FpSpread1.Pager.ForeColor = Color.Brown;
FpSpread1.Pager.BackColor = Color.Orange;
// Display at most four page numbers at a time.
FpSpread1.Pager.PageCount = 4;
```

VB

```
' Set the number of sheets.
FpSpread1.Sheets.Count = 5
' Set the number of rows on the first sheet.
FpSpread1.Sheets(0).RowCount = 136' Set the number of rows per page in this sheet.
FpSpread1.Sheets(0).PageSize = 13

' Display the pager only at the top of the component.
FpSpread1.Pager.Position = FarPoint.Web.Spread.PagerPosition.Top
' Display both numbers and arrows by setting mode.
' Set the mode after the position, otherwise an error.
FpSpread1.Pager.Mode = FarPoint.Web.Spread.PagerMode.Both
' Format the text in the pager at the top.
FpSpread1.Pager.Align = HorizontalAlign.Right
FpSpread1.Pager.Font.Bold = True
FpSpread1.Pager.Font.Name = "Trebuchet MS"
FpSpread1.Pager.ForeColor = Color.Brown
```

```
FpSpread1.Pager.BackColor = Color.Orange
' Display at most four page numbers at a time.
FpSpread1.Pager.PageCount = 4
```

Using the Spread Designer

1. Select the **Settings** menu.
2. Select the **Paging** icon under the **Spread Settings** section.
3. Select the various options.
4. Click **Apply** and **OK**.
5. Click **Apply and Exit** to close the Spread Designer.

Customizing Page Navigation Buttons on the Client

If the command bar is displayed, you can customize the labels of the page navigation buttons, as well as other aspects of the button appearance by using the **CreateButton ('CreateButton Event' in the on-line documentation)** event. Refer to the **CreateButtonEventArgs ('CreateButtonEventArgs Class' in the on-line documentation)** class members for more details.

If the command bar is not displayed, there are no buttons being created so you cannot use the **CreateButton ('CreateButton Event' in the on-line documentation)** event. But you can still customize the appearance using client code to achieve the same results. On the client side, the page navigation links are drawn in a table cell on the resulting HTML page. You can get access to that table cell and draw your own label instead.

Using Code

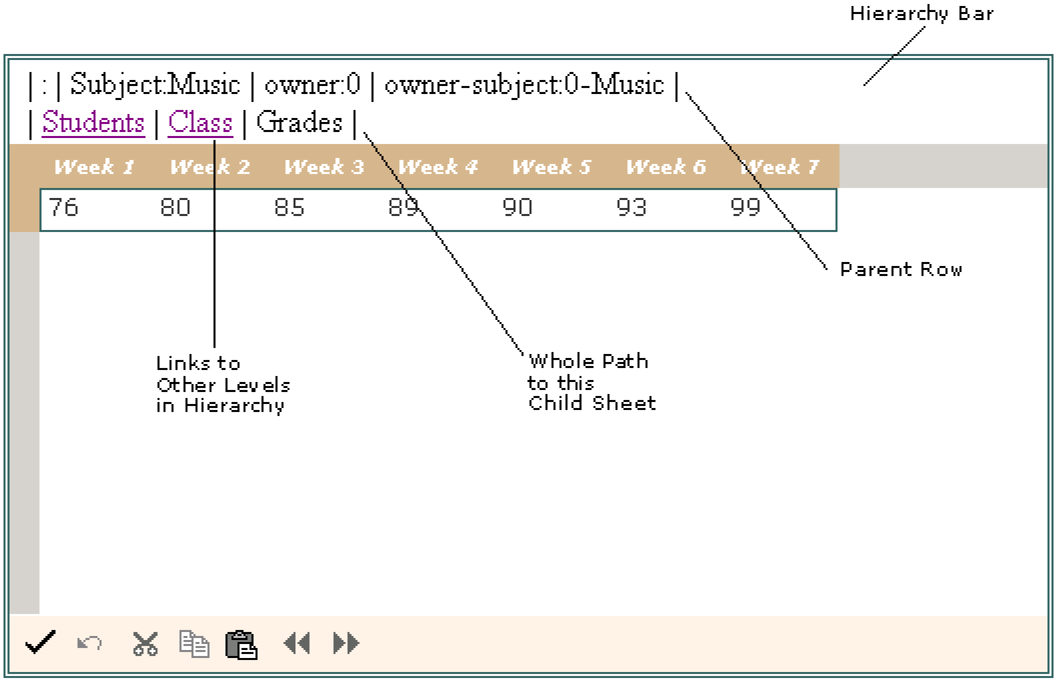
Here is the client code to display a version of the links to the next and previous pages by manipulating the table cells and creating buttons for these.

Client Code

```
function window.onload()
   {
     var pager=document.all("FpSpread1_Pager1");
     for(var i=0;i&lt;pager.childNodes.length;i++) {
       switch (pager.childNodes(i).nodeType) {
        case 1:
         switch (pager.childNodes(i).innerText) {
          case "&lt;&lt;":
            pager.childNodes(i).innerText = "&lt;&lt;Prev ";
            break;
          case "&gt;&gt;":
            pager.childNodes(i).innerText = " Next&gt;&gt;";
            break;
        }
      case 3:
        switch (pager.childNodes(i).data) {
          case "&lt;&lt;":
            pager.childNodes(i).data = "&lt;&lt; Prev";
            break;
          case "&gt;&gt;":
            pager.childNodes(i).data = " Next &gt;&gt;";
            break;
         }
    }
    }
}
```

Customizing the Hierarchy Bar

When you nest an entire sheet in a cell, you have a hierarchy. As an alternative to displaying the entire hierarchy of sheets, you can display only one sheet at a time with its hierarchy information displayed in the tool bars above the sheet. This hierarchy information displays the names of the different sheet levels (the whole path) on one line and lets you click on any of those levels, and it displays the information about the parent row (the row above the displayed sheet) on another line. You can decide whether to display one or both of these with the **HierBar ('HierBar Property' in the on-line documentation)** property.


Hierarchy Bar
| : | Subject:Music | owner:0 | owner-subject:0-Music |
| Students | Class | Grades |
Week 1
Week 2
Week 3
Week 4
Week 5
Week 6
Week 7
76
80
85
89
90
93
99
Parent Row
Links to Other Levels in Hierarchy
Whole Path to this Child Sheet

You can customize how the hierarchy information is displayed for cells that have sheets within them. You can display the parent row information, the whole path information, or both.

For more information on the hierarchical display of data, typically with a component bound to a data set, refer to **Displaying Data as a Hierarchy**.

For more information on Outlook-style grouping for hierarchical display of data, refer to **Customizing Grouping of Rows of User Data**.

For information about how to customize the expand and collapse icons, refer to the **Customizing the Graphical Interface**.

Using Code

1. Bind the Spread control to a hierarchical dataset (refer to the **HierBar ('HierBar Property' in the on-line documentation)** property to see an example of how to bind the control to a hierarchical dataset).
2. Add code that sets the specific property for the hierarchy bar using either the **HierBarInfo ('HierBarInfo Class' in the on-line documentation)** class or the **HierBar ('HierBar Property' in the on-line documentation)** property of the **FpSpread ('FpSpread Class' in the on-line documentation)** component.

Example

In this example, turn on the hierarchy bar and display it above the sheet (and thereby not display the entire hierarchy of sheets) and show both pieces of information: the parent row information and the whole path to this child sheet.

C#

```
FpSpread1.HierarchicalView = false;
FpSpread1.HierBar.ShowParentRow = true;
FpSpread1.HierBar.ShowWholePath = true;
```

VB

```
FpSpread1.HierarchicalView = False
FpSpread1.HierBar.ShowParentRow = True
FpSpread1.HierBar.ShowWholePath = True
```

Customizing Interaction with Rows and Columns

You can customize these aspects of user interaction with rows and columns:

- **Allowing the User to Move Columns**
- **Allowing the User to Move Rows (on-line documentation)**
- **Allowing the User to Resize Rows or Columns**
- **Freezing Rows and Columns**
- **Setting up Row Edit Templates**
- **Setting up Preview Rows**

For more information related to rows and columns, refer to these topics:

- **Customizing Simple Filtering of Rows of User Data**
- **Customizing Grouping of Rows of User Data**
- **Customizing Sorting of Rows of User Data**

For more information related to ranges of cells, refer to these topics:

- **Customizing Selections of Cells**
- **Customizing the Appearance of a Cell**
- **Customizing with Cell Types**

Allowing the User to Move Columns

You can allow the user to drag and move columns. Set the **AllowColumnMove ('AllowColumnMove Property' in the on-line documentation)** property in the **SheetView ('SheetView Class' in the on-line documentation)** class to allow the user to move columns.

For the user to move columns, they simply left click on the header of the 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.

Moving columns is not supported in child sheets in hierarchical displays.

Using the Properties Window

1. At design time, in the **Properties** window, select the FpSpread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.

4. In the **Behavior** section, set the **AllowColumnMove** property.
5. Click **OK** to close the **SheetView Collection Editor**.

Using a Shortcut

Set the **AllowColumnMove ('AllowColumnMove Property' in the on-line documentation)** property.

Example

This example code sets the **AllowColumnMove ('AllowColumnMove Property' in the on-line documentation)** property.

C#

```
FpSpread1.ActiveSheetView.AllowColumnMove = true;
```

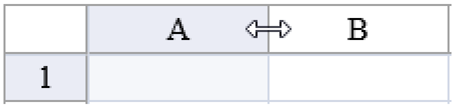
VB

```
FpSpread1.ActiveSheetView.AllowColumnMove = True
```

Allowing the User to Resize Rows or Columns

You can allow the user to readjust 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.

For users to resize rows or columns, they simply 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 figure below. Be sure to click on the right edge of the column and bottom edge of the row.



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** property and prevent the user from resizing.

You can determine if a row or column can be resized by the user with these methods in the **SheetView ('SheetView Class' in the on-line documentation)** 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)**

Using Code

Set the **Resizable ('Resizable Property' in the on-line documentation)** property.

Example

This example code sets the **Resizable ('Resizable Property' in the on-line documentation)** property.

C#

```
FpSpread1.Sheets[0].Columns[0].Resizable = true;
FpSpread1.Sheets[0].Rows[0].Resizable = true;
FpSpread1.Sheets[0].Columns[1].Resizable = false;
FpSpread1.Sheets[0].Rows[1].Resizable = false;
```

VB

```
FpSpread1.Sheets(0).Columns(0).Resizable = True
FpSpread1.Sheets(0).Rows(0).Resizable = True
FpSpread1.Sheets(0).Columns(1).Resizable = False
FpSpread1.Sheets(0).Rows(1).Resizable = False
```

Using the Spread Designer

1. Select a column or row.
2. Set the **Resizable** property in the **Property Grid**.
3. Click **Apply and Exit** to close the Spread Designer.

Freezing Rows and Columns

Frozen rows and frozen columns do not scroll when the user uses the scroll bar or navigation keys in the component. This is useful if you need information in non-header rows or columns to stay visible regardless of where in the sheet the user navigates. Frozen rows and frozen columns are supported with Microsoft Internet Explorer (IE) and Mozilla Firefox when the FpSpread **EnableClientScript ('EnableClientScript Property' in the on-line documentation)** property is true.

Using the Properties Window

1. At design time, in the **Properties** window, select the FpSpread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. In the **Appearance** section, set the number of frozen rows or columns using **FrozenRowCount** or **FrozenColumnCount**.
5. Click **OK** to close the **SheetView Collection Editor**.

Using a Shortcut

Set the **FrozenColumnCount ('FrozenColumnCount Property' in the on-line documentation)** or **FrozenRowCount ('FrozenRowCount Property' in the on-line documentation)** property in the sheet of the component.

Example

This example code sets **FrozenColumnCount ('FrozenColumnCount Property' in the on-line documentation)** and **FrozenRowCount ('FrozenRowCount Property' in the on-line documentation)**.

C#

```
FpSpread1.Sheets[0].FrozenColumnCount = 2;
FpSpread1.Sheets[0].FrozenRowCount = 1;
```

VB

```
FpSpread1.Sheets(0).FrozenColumnCount = 2
```

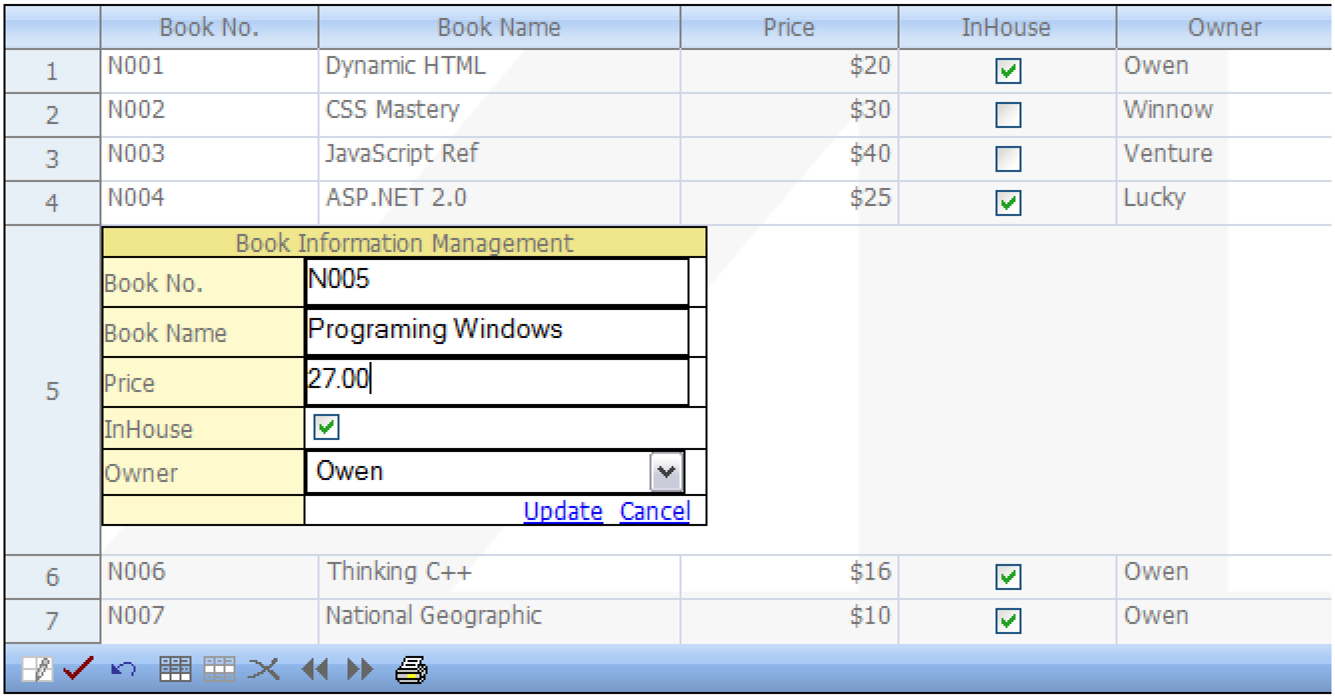
```
FpSpread1.Sheets(0).FrozenRowCount = 1
```

Using the Spread Designer

1. Select Sheet from the drop-down combo box to the right of the designer.
2. In the **Appearance** section, set the number for **FrozenColumnCount** or **FrozenRowCount**.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting up Row Edit Templates

You can allow a row edit template to be displayed so the user can edit the contents of an entire row using a specified template or form. In the example shown here, the entire row of data is presented to the user as a vertical form with an Update and **Cancel** button at the bottom of the form.



	Book No.	Book Name	Price	InHouse	Owner
1	N001	Dynamic HTML	$20	☑	Owen
2	N002	CSS Mastery	$30	☐	Winnow
3	N003	JavaScript Ref	$40	☐	Venture
4	N004	ASP.NET 2.0	$25	☑	Lucky
5					
6	N006	Thinking C++	$16	☑	Owen
7	N007	National Geographic	$10	☑	Owen

Book Information Management	
Book No.	N005
Book Name	Programing Windows
Price	27.00
InHouse	☑
Owner	Owen
	Update Cancel

The API members that are involved with this feature include:

- FpSpread.**RowEditTemplate ('RowEditTemplate Property' in the on-line documentation)** property
- SheetView.**RowEditTemplate ('RowEditTemplate Property' in the on-line documentation)** property
- **RowEditTemplateContainer ('RowEditTemplateContainer Class' in the on-line documentation)** class and its members

In most cases, you can use **RowEditTemplateContainer ('RowEditTemplateContainer Class' in the on-line documentation)** for the template control. But the editing template can contain other kinds of controls. In these cases, you must develop code to update the data of these controls.

When users double click a row, the row edit template appears. Then users can edit the data of the row, and press the **Update** link button to commit changes or press the **Cancel** link button to discard the changes.

The row edit template supports XML serialization so custom editing template can be saved and loaded along with Spread.

You can create the row edit template at design time or run time.

Using Code

Set the **EnableRowEditTemplate ('EnableRowEditTemplate Property' in the on-line documentation)**, **EnableClientScript ('EnableClientScript Property' in the on-line documentation)**, and **OperationMode ('OperationMode Property' in the on-line documentation)** properties.

Example

This example code sets the **EnableRowEditTemplate ('EnableRowEditTemplate Property' in the on-line documentation)** property.

C#

```
FpSpread1.ActiveSheetView.EnableRowEditTemplate = true;
FpSpread1.EnableClientScript = true;
FpSpread1.ActiveSheetView.OperationMode = FarPoint.Web.Spread.OperationMode.RowMode;
```

VB

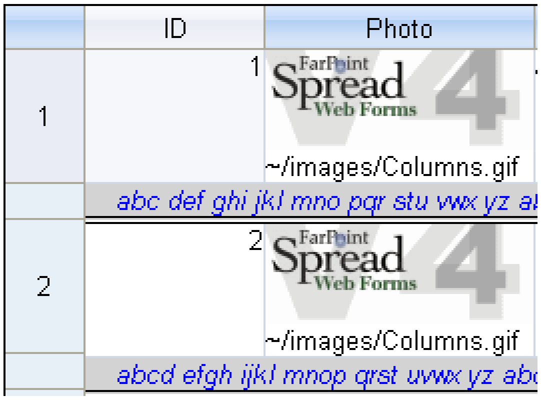
```
FpSpread1.ActiveSheetView.EnableRowEditTemplate = True
FpSpread1.EnableClientScript = True
FpSpread1.ActiveSheetView.OperationMode = FarPoint.Web.Spread.OperationMode.RowMode
```

Using Design Settings

1. Select **Edit Templates** from the Spread smart tag.
2. Select **Editing Template** from the drop-down.
3. Drag the FpSpreadTemplateReplacement control from the Visual Studio toolbox to the edit template area.
4. Select a task (such as CellType) and specify the column index.
5. Add code to set **EnableRowEditTemplate** to true and **OperationMode** to row mode (see above code).
6. Run the project and double-click a cell in column 0 to edit it.

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 it provides information for. You can specify colors and other formatting for the preview row as well. The gray rows in the following picture are preview rows.



	ID	Photo
1	1	~/images/Columns.gif
	abc def ghi jkl mno pqr stu vwx yz a	
2	2	~/images/Columns.gif
	abcd efgh ijkl mnop qrst uvwx yz abc	

Use the **Edit Template** verb at design time to create a preview template.

Set the **PreviewRowVisible ('PreviewRowVisible Property' in the on-line documentation)** property to true in order to see the preview row. Use the **PreviewRowColumnIndex ('PreviewRowColumnIndex Property' in the on-line documentation)** property to specify which column's text you wish to see in the preview row. You can use the **PreviewRowStyle ('PreviewRowStyle Property' in the on-line documentation)** property to provide additional formatting using a style element. Or you can set various properties for the **PreviewRowStyle ('PreviewRowStyle Property' in the on-line documentation)** such as BackColor, Border, Font, and so on.

The API members involved in this feature include:

- **PreviewRowInfo ('PreviewRowInfo Class' in the on-line documentation)** class
- **PreviewRowTemplateContainer ('PreviewRowTemplateContainer Class' in the on-line documentation)** class
- SheetView.**PreviewRowColumnIndex ('PreviewRowColumnIndex Property' in the on-line documentation)** property
- SheetView.**PreviewRowStyle ('PreviewRowStyle Property' in the on-line documentation)** property
- SheetView.**PreviewRowTemplate ('PreviewRowTemplate Property' in the on-line documentation)** property
- SheetView.**PreviewRowVisible ('PreviewRowVisible Property' in the on-line documentation)** property

Using Design Settings

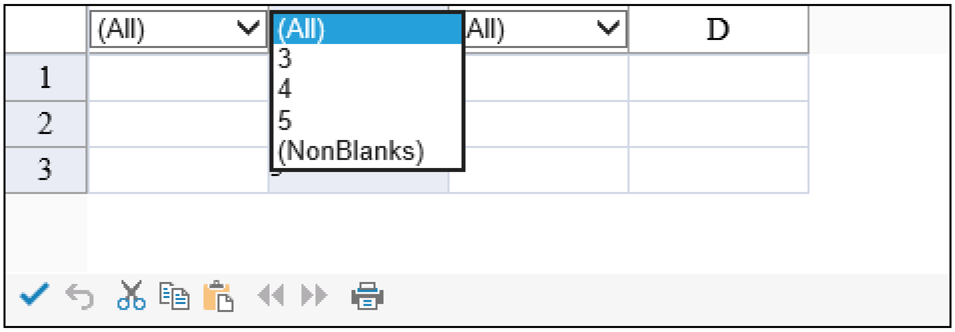
1. Click on **Edit Templates** from the Spread smart tag.
2. Select **Preview Row Template** from the drop-down.
3. Drag the a control from the Visual Studio toolbox to the edit template area or use a default template.
4. Set colors or other properties.
5. Select **End Template Editing** when you are done editing the template.

Managing Filtering of Rows of User Data

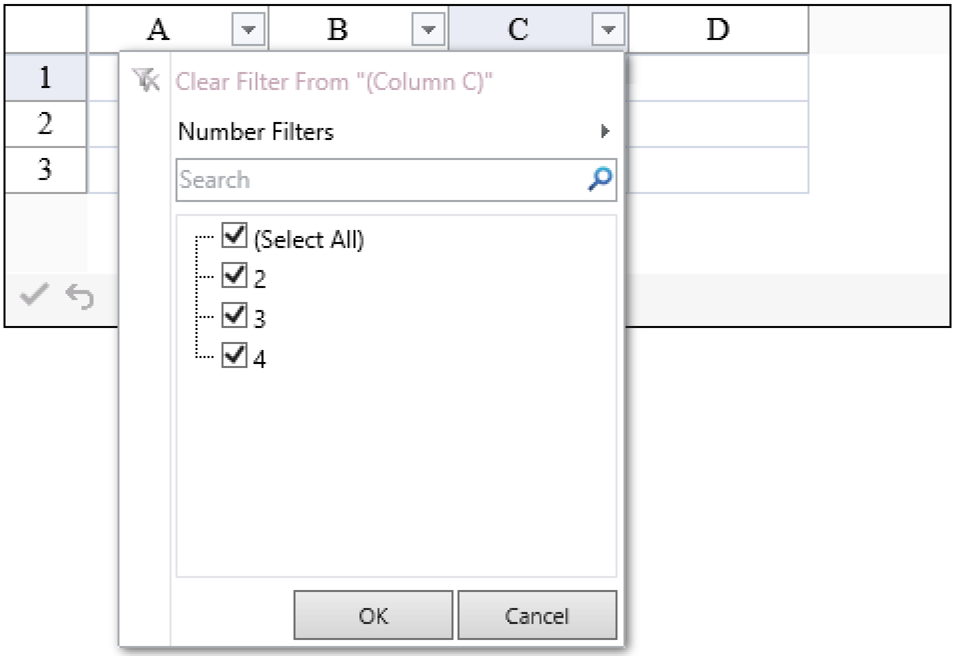
You can create filters for data. The basic steps are to create a filter and then assign the filter to a column. You can specify certain details for the filter such as creating a background color for filtered and non-filtered rows with a style filter or creating a hide row filter that hides the rows.

Spread provides several 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. Spread also provides a filter bar that uses the enhanced filtering options.

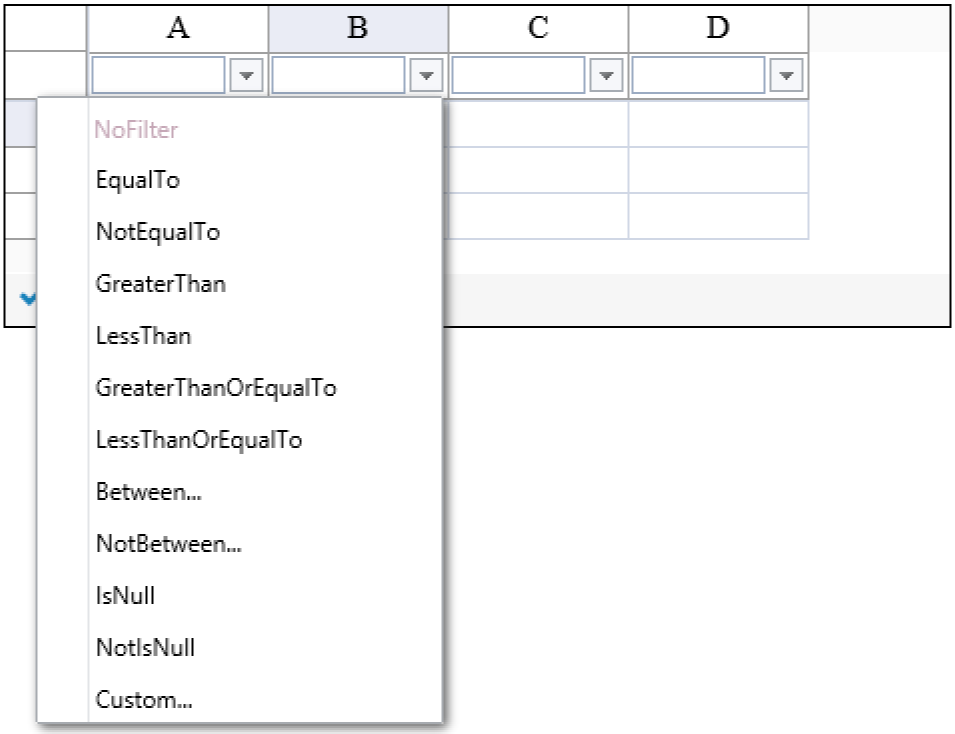
Each type of filtering provides a way for users to change data's appearance or temporarily hide data based on conditions that they specify, as shown in the following figures. This figure illustrates the simple filter.



The following figure illustrates the enhanced filter.



The filter bar provides a text box, a list of enhanced filter choices, and a filter icon. This figure illustrates a filter bar.



You can customize many features for each type of filtering, as well as the display of filtered rows, as described in the following sections.

- **Creating Filtered Rows and Setting the Appearance**
- **Customizing Simple Filtering of Rows of User Data**
- **Using Enhanced Filtering**
- **Using the Filter Bar**

Creating Filtered Rows and Setting the Appearance

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 such as the background color 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.

Hidden rows are not displayed even if they match the filter criteria.

A row filter uses a style row filter or a hide row filter. The style row filter changes the appearance of the filtered row. The hide row filter hides the rows that do not meet the filter criteria.

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. For more information about the row filter that uses styles, refer to the **StyleRowFilter ('StyleRowFilter Class' in the on-line documentation)** class.

You can create a hide or style row filter using the Spread Designer. Select the Spread control in the property grid drop-down of the designer, then select the **Sheets Collection** (under **Data**), and then select the **Row Filter** option in the **SheetView Collection** editor.

In addition to creating row filters for the user to select the item; you can also programmatically filter a row.

You can specify simple filtering, enhanced filtering, or the filter bar with the **AutoFilterMode ('AutoFilterMode Property' in the on-line documentation)** property.

For detailed information on the objects involved, refer to these classes.

- **BaseFilterItem ('BaseFilterItem Class' in the on-line documentation)** Class
- **DefaultFilterItem ('DefaultFilterItem Class' in the on-line documentation)** Class
- **DefaultRowFilter ('DefaultRowFilter Class' in the on-line documentation)** Class
- **FilterColumnDefinition ('FilterColumnDefinition Class' in the on-line documentation)** Class
- **FilterColumnDefinitionCollection ('FilterColumnDefinitionCollection Class' in the on-line documentation)** Class
- **FilterItemCollection ('FilterItemCollection Class' in the on-line documentation)** Class
- **FilterListBehavior ('FilterListBehavior Enumeration' in the on-line documentation)** Enumeration
- **HideRowFilter ('HideRowFilter Class' in the on-line documentation)** Class
- **StyleRowFilter ('StyleRowFilter Class' in the on-line documentation)** Class

Using Code

Create a named style and then set the style row filter.

Example

This example code sets a style row filter.

C#

```
FarPoint.Web.Spread.NamedStyle instyle = new FarPoint.Web.Spread.NamedStyle();
FarPoint.Web.Spread.NamedStyle outstyle = new FarPoint.Web.Spread.NamedStyle();
instyle.BackColor = Color.Yellow;
outstyle.BackColor = Color.Aquamarine;
FarPoint.Web.Spread.FilterColumnDefinition fcd = new
FarPoint.Web.Spread.FilterColumnDefinition(1,
FarPoint.Web.Spread.FilterListBehavior.SortByMostOccurrences |
FarPoint.Web.Spread.FilterListBehavior.Default);
FarPoint.Web.Spread.FilterColumnDefinition fcd1 = new
FarPoint.Web.Spread.FilterColumnDefinition(2);
FarPoint.Web.Spread.FilterColumnDefinition fcd2 = new
FarPoint.Web.Spread.FilterColumnDefinition();
FarPoint.Web.Spread.StyleRowFilter sf = new
FarPoint.Web.Spread.StyleRowFilter(FpSpread1.Sheets[0], instyle, outstyle);
sf.AddColumn(fcd);
sf.AddColumn(fcd1);
sf.AddColumn(fcd2);
FpSpread1.Sheets[0].RowFilter = sf;
```

VB

```
Dim instyle As New FarPoint.Web.Spread.NamedStyle()
Dim outstyle As New FarPoint.Web.Spread.NamedStyle()
instyle.BackColor = Color.Yellow
outstyle.BackColor = Color.Aquamarine
Dim fcd As New FarPoint.Web.Spread.FilterColumnDefinition(1,
FarPoint.Web.Spread.FilterListBehavior.SortByMostOccurrences Or
FarPoint.Web.Spread.FilterListBehavior.Default)
Dim fcd1 As New FarPoint.Web.Spread.FilterColumnDefinition(2)
Dim fcd2 As New FarPoint.Web.Spread.FilterColumnDefinition()
Dim sf As New FarPoint.Web.Spread.StyleRowFilter(FpSpread1.Sheets(0), instyle,
outstyle)
sf.AddColumn(fcd)
sf.AddColumn(fcd1)
sf.AddColumn(fcd2)
FpSpread1.Sheets(0).RowFilter = sf
```

Using Code

Create a column filter definition and set a hide row filter.

Example

This example code uses the hide row filter.

C#

```
FarPoint.Web.Spread.FilterColumnDefinition fcd = new
FarPoint.Web.Spread.FilterColumnDefinition(1,
FarPoint.Web.Spread.FilterListBehavior.SortByMostOccurrences |
FarPoint.Web.Spread.FilterListBehavior.Default);
FarPoint.Web.Spread.FilterColumnDefinition fcd1 = new
FarPoint.Web.Spread.FilterColumnDefinition(2,
FarPoint.Web.Spread.FilterListBehavior.Default);
FarPoint.Web.Spread.FilterColumnDefinition fcd2 = new
FarPoint.Web.Spread.FilterColumnDefinition(3);
FarPoint.Web.Spread.HideRowFilter hf = new
FarPoint.Web.Spread.HideRowFilter(FpSpread1.Sheets[0]);
hf.AddColumn(fcd);
hf.AddColumn(fcd1);
hf.AddColumn(fcd2);
FpSpread1.Sheets[0].RowFilter = hf;
```

VB

```
Dim fcd As New FarPoint.Web.Spread.FilterColumnDefinition(1,
FarPoint.Web.Spread.FilterListBehavior.SortByMostOccurrences Or
FarPoint.Web.Spread.FilterListBehavior.Default)
Dim fcd1 As New FarPoint.Web.Spread.FilterColumnDefinition(2)
Dim fcd2 As New FarPoint.Web.Spread.FilterColumnDefinition(3)
Dim hf As New FarPoint.Web.Spread.HideRowFilter(FpSpread1.Sheets(0))
hf.AddColumn(fcd)
hf.AddColumn(fcd1)
hf.AddColumn(fcd2)
FpSpread1.Sheets(0).RowFilter = hf
```

Using Code

Create a column filter definition and set a row filter.

Example

This example applies a filter programmatically with the **AutoFilterColumn ('AutoFilterColumn Method' in the on-line documentation)** method.

C#

```
FpSpread1.Sheets[0].Cells[0, 2].Text = "test";
FarPoint.Web.Spread.NamedStyle instyle = new FarPoint.Web.Spread.NamedStyle();
FarPoint.Web.Spread.NamedStyle outstyle = new FarPoint.Web.Spread.NamedStyle();
instyle.BackColor = Color.Yellow;
outstyle.BackColor = Color.Aquamarine;
FarPoint.Web.Spread.FilterColumnDefinition fcd = new
FarPoint.Web.Spread.FilterColumnDefinition(1,
FarPoint.Web.Spread.FilterListBehavior.SortByMostOccurrences |
FarPoint.Web.Spread.FilterListBehavior.Default);
FarPoint.Web.Spread.FilterColumnDefinition fcd1 = new
FarPoint.Web.Spread.FilterColumnDefinition(2);
FarPoint.Web.Spread.FilterColumnDefinition fcd2 = new
FarPoint.Web.Spread.FilterColumnDefinition();
FarPoint.Web.Spread.StyleRowFilter sf = new
FarPoint.Web.Spread.StyleRowFilter(FpSpread1.Sheets[0], instyle, outstyle);
sf.AddColumn(fcd);
sf.AddColumn(fcd1);
sf.AddColumn(fcd2);
FpSpread1.Sheets[0].RowFilter = sf;
FpSpread1.Sheets[0].AutoFilterColumn(2, "test");
```

VB

```
FpSpread1.Sheets(0).Cells(0, 2).Text = "test"
Dim instyle As New FarPoint.Web.Spread.NamedStyle()
Dim outstyle As New FarPoint.Web.Spread.NamedStyle()
instyle.BackColor = Color.Yellow
outstyle.BackColor = Color.Aquamarine
Dim fcd As New FarPoint.Web.Spread.FilterColumnDefinition(1,
FarPoint.Web.Spread.FilterListBehavior.SortByMostOccurrences Or
FarPoint.Web.Spread.FilterListBehavior.Default)
Dim fcd1 As New FarPoint.Web.Spread.FilterColumnDefinition(2)
Dim fcd2 As New FarPoint.Web.Spread.FilterColumnDefinition()
Dim sf As New FarPoint.Web.Spread.StyleRowFilter(FpSpread1.Sheets(0), instyle,
outstyle)
```

```
sf.AddColumn(fcd)
sf.AddColumn(fcd1)
sf.AddColumn(fcd2)
FpSpread1.Sheets(0).RowFilter = sf
FpSpread1.Sheets(0).AutoFilterColumn(2, "test")
```

Customizing Simple Filtering of Rows of User Data

You can customize certain details for simple filtering as well as create a custom filter. The following topics provide more information about using and customizing simple filters:

- **Using Row Filtering**
- **Customizing the List of Filter Items**
- **Creating a Completely Custom Filter**

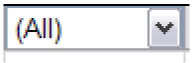
For information on selections of cells, refer to **Customizing Selections of Cells**.

For information on how to change the appearance of cells, refer to **Customizing the Appearance of a Cell**. For information on setting the cell type, refer to **Customizing with Cell Types**.

Using Row Filtering

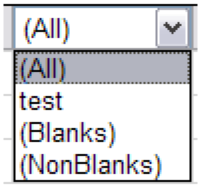
This topic summarizes how the end user can interact with the simple row filtering feature.

Once you have row filtering applied to a column, an indicator appears in the column header as in the following figure:



If you want to display a label for the header, you must add another row of column headers and put the label in that row.

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 filter list. For more information, see **Customizing the List of Filter Items**.

Using Code

Create a named style and then set the style row filter.

Example

This example code sets a style row filter.

C#

```
FpSpread1.ActiveSheetView.AutoFilterMode =
FarPoint.Web.Spread.AutoFilterMode.FilterGadget;

FarPoint.Web.Spread.NamedStyle instyle = new FarPoint.Web.Spread.NamedStyle();
FarPoint.Web.Spread.NamedStyle outstyle = new FarPoint.Web.Spread.NamedStyle();
instyle.BackColor = Color.Yellow;
outstyle.BackColor = Color.Aquamarine;
FarPoint.Web.Spread.FilterColumnDefinition fcd = new
FarPoint.Web.Spread.FilterColumnDefinition(1,
FarPoint.Web.Spread.FilterListBehavior.SortByMostOccurrences |
FarPoint.Web.Spread.FilterListBehavior.Default);
FarPoint.Web.Spread.FilterColumnDefinition fcd1 = new
FarPoint.Web.Spread.FilterColumnDefinition(2);
FarPoint.Web.Spread.FilterColumnDefinition fcd2 = new
FarPoint.Web.Spread.FilterColumnDefinition();
FarPoint.Web.Spread.StyleRowFilter sf = new
FarPoint.Web.Spread.StyleRowFilter(FpSpread1.Sheets[0], instyle, outstyle);
sf.AddColumn(fcd);
sf.AddColumn(fcd1);
sf.AddColumn(fcd2);
FpSpread1.Sheets[0].RowFilter = sf;
```

VB

```
FpSpread1.ActiveSheetView.AutoFilterMode =
FarPoint.Web.Spread.AutoFilterMode.FilterGadget

Dim instyle As New FarPoint.Web.Spread.NamedStyle()
Dim outstyle As New FarPoint.Web.Spread.NamedStyle()
instyle.BackColor = Color.Yellow
outstyle.BackColor = Color.Aquamarine
Dim fcd As New FarPoint.Web.Spread.FilterColumnDefinition(1,
FarPoint.Web.Spread.FilterListBehavior.SortByMostOccurrences Or
FarPoint.Web.Spread.FilterListBehavior.Default)
Dim fcd1 As New FarPoint.Web.Spread.FilterColumnDefinition(2)
Dim fcd2 As New FarPoint.Web.Spread.FilterColumnDefinition()
Dim sf As New FarPoint.Web.Spread.StyleRowFilter(FpSpread1.Sheets(0), instyle,
outstyle)
sf.AddColumn(fcd)
sf.AddColumn(fcd1)
sf.AddColumn(fcd2)
FpSpread1.Sheets(0).RowFilter = sf
```

Customizing the List of Filter Items

You can customize the text of the default filter items in the drop-down filter list. The items that are displayed in the drop-down filter list are column filter definitions. You can customize the text of those by changing the value of the

AllString ('AllString Property' in the on-line documentation), **BlanksString ('BlanksString Property' in the on-line documentation)**, and **NonBlanksString ('NonBlanksString Property' in the on-line documentation)** properties.

Using Code

Create a row filter and set the filter strings.

Example

This example creates custom filter items for the drop-down list.

C#

```
FarPoint.Web.Spread.NamedStyle instyle = new FarPoint.Web.Spread.NamedStyle();
FarPoint.Web.Spread.NamedStyle outstyle = new FarPoint.Web.Spread.NamedStyle();
instyle.BackColor = Color.Yellow;
outstyle.BackColor = Color.Gray;
FarPoint.Web.Spread.StyleRowFilter rf = new
FarPoint.Web.Spread.StyleRowFilter(FpSpread1.Sheets[0], instyle, outstyle);
FpSpread1.Sheets[0].RowFilter = rf;
// Assign filter and customize filter options.
FpSpread1.Sheets[0].RowFilter.AddColumn(1);
FpSpread1.Sheets[0].RowFilter.ShowFilterIndicator = true;
FpSpread1.Sheets[0].RowFilter.AllString = "Show All";
FpSpread1.Sheets[0].RowFilter.BlanksString = "Show The Blanks";
FpSpread1.Sheets[0].RowFilter.NonBlanksString = "Show The Non-Blanks";
```

VB

```
Dim instyle As New FarPoint.Web.Spread.NamedStyle()
Dim outstyle As New FarPoint.Web.Spread.NamedStyle()
instyle.BackColor = Color.Yellow
outstyle.BackColor = Color.Gray
Dim rf As New FarPoint.Web.Spread.StyleRowFilter(FpSpread1.Sheets(0), instyle,
outstyle)
FpSpread1.Sheets(0).RowFilter = rf
' Assign filter and customize filter options.
FpSpread1.Sheets(0).RowFilter.AddColumn(1)
FpSpread1.Sheets(0).RowFilter.ShowFilterIndicator = True
FpSpread1.Sheets(0).RowFilter.AllString = "Show All"
FpSpread1.Sheets(0).RowFilter.BlanksString = "Show The Blanks"
FpSpread1.Sheets(0).RowFilter.NonBlanksString = "Show The Non-Blanks"
```

Creating a Completely Custom Filter

You can create a custom filter that you can then include in the filter column definition collection. In order to create a custom filter, follow these steps:

1. Create a class that inherits from FarPoint.Web.Spread.BaseFilterItem or FarPoint.Web.Spread.DefaultFilterItem.
2. Override **DisplayName** property to return the name to be displayed in the drop-down list of filter items.
3. Override the **ShowInDropDown** method to specify if this filter item should be displayed in the drop-down list given the current filtered in rows. If the custom filter does not contain this item then the filter is not applied.
4. Override the **Filter** method to perform the filter action on the specified column.
5. Override the **Serialize** and **Deserialize** methods. Make calls to the base.Serialize and base.Deserialize methods unless your methods handle persisting the default properties.

6. Create a HideRowFilter or StyleRowFilter 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.

If you are creating a custom filter item and you display a modal dialog inside the **Filter** method of the filter item, then after the dialog goes away, call the IRowFilter interface **ResetCachedIntersectedFilteredInRowIndexes** method.

C#

```
FpSpread1.Sheets(0).RowFilter.ResetCachedIntersectedFilteredInRowIndexes
```

More information about creating custom filters is available on our online technical support forum (see the Read Me for more information).

Using Enhanced Filtering

When the control has enhanced filtering turned on, the user can drop-down a list of available filters to apply to the data.

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 **Creating Filtered Rows and Setting the Appearance**.

Using Code

Set the **AutoFilterMode ('AutoFilterMode Property' in the on-line documentation)** property to Enhanced, create a filter style, and then apply the filter to the sheet.

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].AutoFilterMode = FarPoint.Web.Spread.AutoFilterMode.Enhanced;
FarPoint.Web.Spread.NamedStyle instyle = new FarPoint.Web.Spread.NamedStyle();
FarPoint.Web.Spread.NamedStyle outstyle = new FarPoint.Web.Spread.NamedStyle();
instyle.BackColor = Color.Yellow;
outstyle.BackColor = Color.Aquamarine;
FarPoint.Web.Spread.FilterColumnDefinition fcd = new
FarPoint.Web.Spread.FilterColumnDefinition(1,
FarPoint.Web.Spread.FilterListBehavior.SortByMostOccurrences |
FarPoint.Web.Spread.FilterListBehavior.Default);
FarPoint.Web.Spread.FilterColumnDefinition fcd1 = new
FarPoint.Web.Spread.FilterColumnDefinition(2);
FarPoint.Web.Spread.FilterColumnDefinition fcd2 = new
FarPoint.Web.Spread.FilterColumnDefinition();
FarPoint.Web.Spread.StyleRowFilter sf = new
FarPoint.Web.Spread.StyleRowFilter(FpSpread1.Sheets[0], instyle, outstyle);
sf.AddColumn(fcd);
sf.AddColumn(fcd1);
sf.AddColumn(fcd2);
FpSpread1.Sheets[0].RowFilter = sf;
```

VB

```
FpSpread1.Sheets(0).AutoFilterMode = FarPoint.Web.Spread.AutoFilterMode.Enhanced
Dim instyle As New FarPoint.Web.Spread.NamedStyle()
Dim outstyle As New FarPoint.Web.Spread.NamedStyle()
instyle.BackColor = Drawing.Color.Yellow
outstyle.BackColor = Drawing.Color.Aquamarine
Dim fcd As New FarPoint.Web.Spread.FilterColumnDefinition(1,
FarPoint.Web.Spread.FilterListBehavior.SortByMostOccurrences Or
FarPoint.Web.Spread.FilterListBehavior.Default)
Dim fcd1 As New FarPoint.Web.Spread.FilterColumnDefinition(2)
Dim fcd2 As New FarPoint.Web.Spread.FilterColumnDefinition()
Dim sf As New FarPoint.Web.Spread.StyleRowFilter(FpSpread1.Sheets(0), instyle,
outstyle)
sf.AddColumn(fcd)
sf.AddColumn(fcd1)
sf.AddColumn(fcd2)
FpSpread1.Sheets(0).RowFilter = sf
```

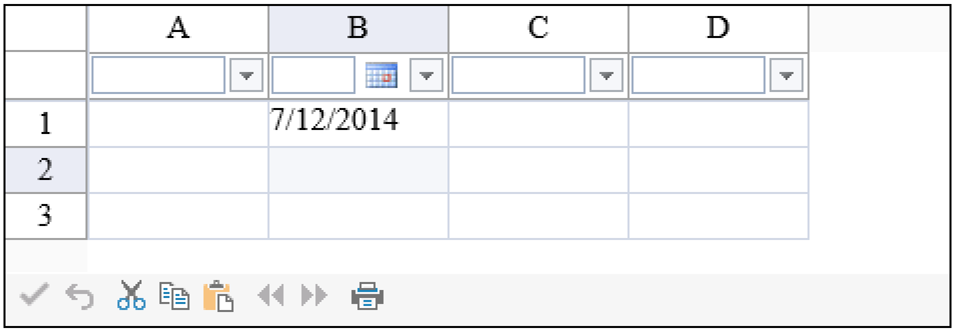
Using the Filter Bar

The filter bar allows filtering in all columns by displaying a filter bar below the column header area. The filter icon appears in the filter bar instead of the column header. The filter bar displays filter information and allows users to edit the condition criteria for each filter column.

The filter bar contains a text box, a menu of choices, and the filter icon. The user can enter a filter item in the text box, select a menu option, and then click on the filter icon to apply the filter.

The **MenuType ('MenuType Property' in the on-line documentation)** property specifies the type of menu options that are displayed for the filter (number, date time, enhanced, or text). If the menu type is date, a date time picker is also available in the filter bar. The automatic option displays the menu options based on the type of data in the column.

The following image displays a date picker in the filter bar for column B.



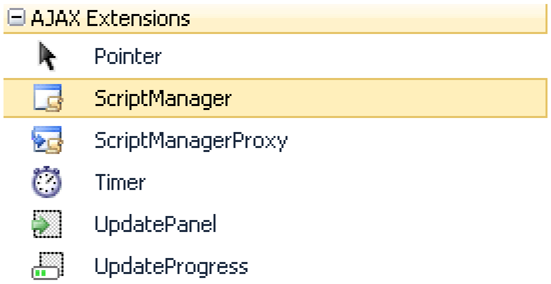
The following table lists the menu options for text, number, and date:

Filter	Menu Options
Text	Contains
	DoesNotContain
	StartsWith
	EndsWin
	EqualTo
	NotEqualTo
	Between
	NotBetween
	IsEmpty
	NotIsEmpty
	IsNull
	NotIsNull
Number and Date	EqualTo
	NotEqualTo
	GreaterThan
	LessThan
	GreaterThanOrEqualTo
	LessThanOrEqualTo
	Between
	NotBetween
	IsNull
	NotIsNull

The **FilterBarMode ('FilterBarMode Property' in the on-line documentation)** property specifies whether the filter context menu data is requested from the server after the page is loaded or loaded in the server before the page is rendered.

Set the **DateTimeFormat ('DateTimeFormat Property' in the on-line documentation)** or **FormatString ('FormatString Property' in the on-line documentation)** property to format the value from the date picker in the filter bar. Set these properties if the format of the data in the cell is different from the format in the filter. The EqualTo menu option requires that the cell format and the filter bar format be the same. A ScriptManager is required for the

DateTimeFormat ('DateTimeFormat Property' in the on-line documentation) and **FormatString ('FormatString Property' in the on-line documentation)** properties.



Using Code

1. Create a filter bar cell if you wish to customize the default options in the filter bar.
2. Set the **AutoFilterMode ('AutoFilterMode Property' in the on-line documentation)** property to FilterBar.

Example

This example code customizes the filter bar and using a filter bar for filtering.

C#

```
protected void Page_Load(object sender, EventArgs e)
  {
    if (IsPostBack) return;
    for (int i = 0; i < FpSpread1.ActiveSheetView.RowCount; i++)
      for (int j = 0; j < FpSpread1.ActiveSheetView.ColumnCount; j++)
      {
        FpSpread1.ActiveSheetView.Cells[i, j].Value = i + j;
      }

    FarPoint.Web.Spread.FilterBarCellType fbc = new
FarPoint.Web.Spread.FilterBarCellType();
    fbc.MenuType = FarPoint.Web.Spread.FilterMenuType.Auto;
    FpSpread1.ActiveSheetView.FilterBar.DefaultStyle.CellType = fbc;
    FpSpread1.ActiveSheetView.AutoFilterMode =
FarPoint.Web.Spread.AutoFilterMode.FilterBar;
  }
```

VB

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
If (IsPostBack) Then
    Return
End If
For i As Integer = 0 To FpSpread1.ActiveSheetView.RowCount - 1
    For j As Integer = 0 To FpSpread1.ActiveSheetView.ColumnCount - 1
        FpSpread1.ActiveSheetView.Cells(i, j).Value = i + j
    Next
Next

Dim fbc As New FarPoint.Web.Spread.FilterBarCellType()
fbc.MenuType = FarPoint.Web.Spread.FilterMenuType.Auto
```

```
FpSpread1.ActiveSheetView.FilterBar.DefaultStyle.CellType = fbc
FpSpread1.ActiveSheetView.AutoFilterMode = FarPoint.Web.Spread.AutoFilterMode.FilterBar
End Sub
```

Customizing Grouping of Rows of User Data

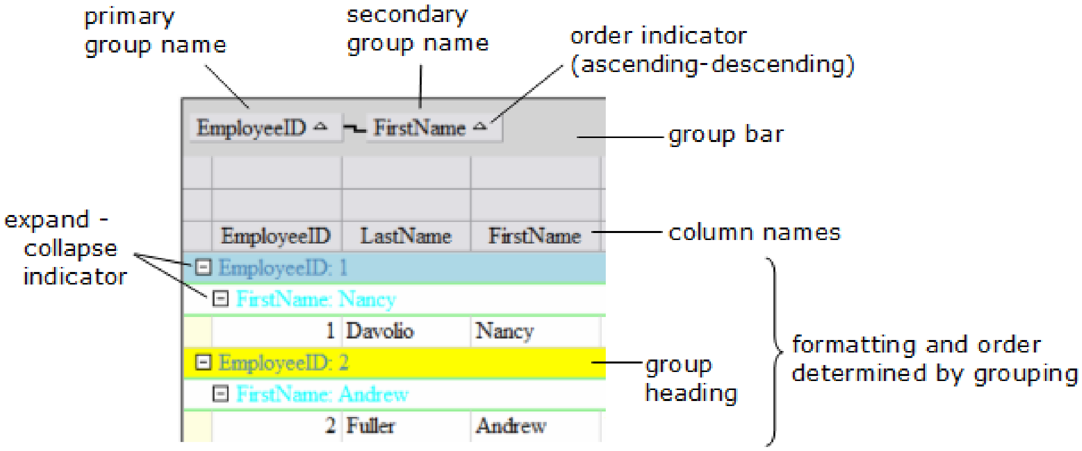
You can set the display of the spreadsheet component to allow rows to be grouped according to the column headers. You can customize the user experience for grouping data on a sheet. With grouping, you can allow the user to group rows of data according to the column headers that are dragged into the group bar. Special group headings are displayed above the grouped rows. Grouping of rows includes the following tasks.

- **Using Grouping**
- **Allowing the User to Group Rows**
- **Setting the Appearance of Grouped Rows**
- **Customizing the Group Bar**
- **Creating a Custom Group**
- **Compatibility with Other Features**

For information about the **GroupInfo** editor in the Spread Designer, refer to **GroupInfo Collection Editor**.

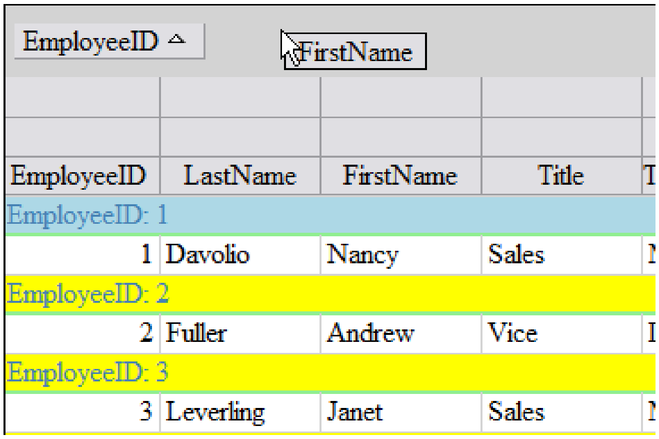
Using Grouping

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.



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.

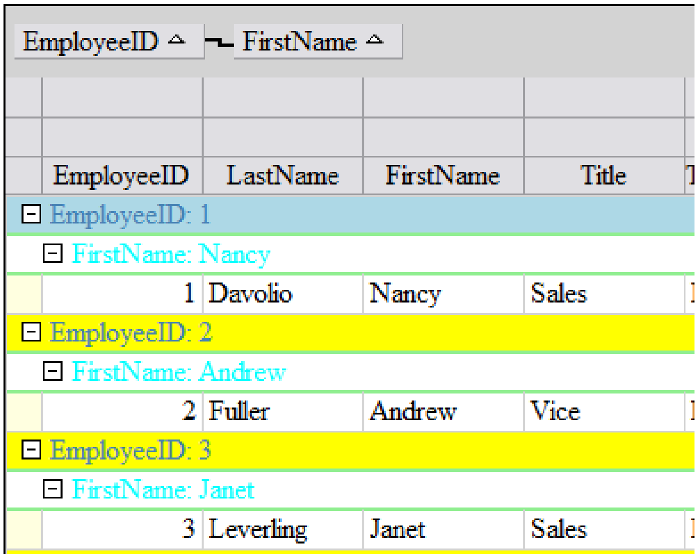
You can expand or collapse groups by clicking the expand (+) or collapse (-) indicators.



EmployeeID ▲ FirstName

EmployeeID	LastName	FirstName	Title
EmployeeID: 1			
1	Davolio	Nancy	Sales
EmployeeID: 2			
2	Fuller	Andrew	Vice
EmployeeID: 3			
3	Leverling	Janet	Sales

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



EmployeeID ▲ FirstName ▲

EmployeeID	LastName	FirstName	Title
⊟ EmployeeID: 1			
⊟ FirstName: Nancy			
1	Davolio	Nancy	Sales
⊟ EmployeeID: 2			
⊟ FirstName: Andrew			
2	Fuller	Andrew	Vice
⊟ EmployeeID: 3			
⊟ FirstName: Janet			
3	Leverling	Janet	Sales

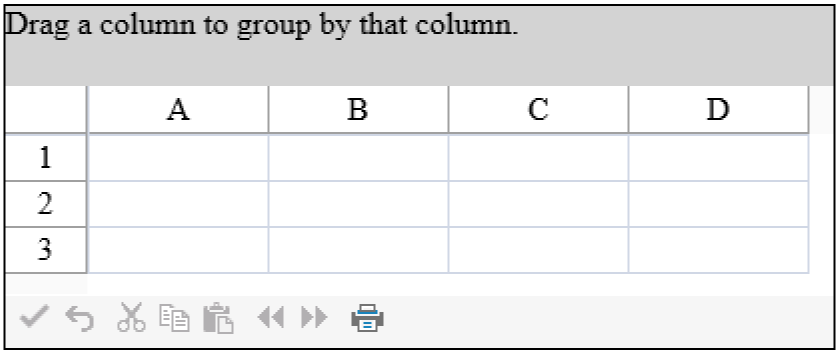
After secondary grouping: now a second level of hierarchy is shown.

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 picture above with secondary grouping, the Employee ID is the parent group and the First Name is the child group.

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.

Use the **AllowGroup ('AllowGroup Property' in the on-line documentation)** property of the sheet to turn on grouping. Use the **GroupBarVisible ('GroupBarVisible Property' in the on-line documentation)** property of the sheet 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 sheet 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. The following image shows the control before the user drags the column header:



Drag a column to group by that column.

	A	B	C	D
1				
2				
3				

You can turn on or off the row headers; these have no effect on the display of 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 the Properties Window

1. At design time, in the **Properties** window, select the FpSpread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. Set **AllowColumnMove**, **GroupBarVisible**, and **AllowGroup**.
5. Click **OK** to close the **SheetView Collection Editor**.

Using Code

Set the **AllowGroup ('AllowGroup Property' in the on-line documentation)**, **AllowColumnMove ('AllowColumnMove Property' in the on-line documentation)**, and the **GroupBarVisible ('GroupBarVisible Property' in the on-line documentation)** properties to allow user grouping.

Example

This example allows the user to group rows.

C#

```
FpSpread1.ActiveSheetView.AllowColumnMove = true;
FpSpread1.ActiveSheetView.GroupBarVisible = true;
FpSpread1.ActiveSheetView.AllowGroup = true;
```

VB

```
FpSpread1.ActiveSheetView.AllowColumnMove = True
FpSpread1.ActiveSheetView.GroupBarVisible = True
FpSpread1.ActiveSheetView.AllowGroup = True
```

Using the Spread Designer

1. Select the **Settings** menu.
2. Select the **Group** icon in the **Other Settings** section.
3. Set the various properties.
4. Select **Sheet** in the **Property Grid** and set **AllowColumnMove**.
5. Click **OK** to close the dialog.
6. Use the **View** menu to show or hide the group bar.
7. Click **Apply and Exit** to close the Spread Designer.

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 with the **GroupingPolicy ('GroupingPolicy Property' in the on-line documentation)** property.

You can set the colors and other formatting of both the hierarchy names and the data in the rows when grouping is performed with the **GroupInfo ('GroupInfo Class' in the on-line documentation)** class.

For more information on other hierarchical displays of data, refer to **Displaying Data as a Hierarchy**.

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 ('GroupInfo Class' in the on-line documentation)** objects is in the GroupInfoCollection. To set the appearance settings in a **GroupInfo ('GroupInfo Class' in the on-line documentation)** to a particular sheet, set the **GroupInfos ('GroupInfos Property' in the on-line documentation)** property on that sheet. Appearance settings for grouping include:

- Background color
- Border
- CSS class
- Font
- Foreground (text) color
- HorizontalAlignment
- VerticalAlignment

Only column and sheet appearance settings remain when grouping is turned on. For more information about the group data model and the effect on the sheet data model, refer to **Creating a Custom Group**. Since rows and cells are moved when the grouping feature is turned on, any style or span settings are ignored. 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 information about the **GroupInfo** editor in the Spread Designer, for customizing the appearance settings of the group headers, refer to **GroupInfo Collection Editor**.

Using Code

1. Set the **AllowColumnMove ('AllowColumnMove Property' in the on-line documentation)**, **GroupBarVisible ('GroupBarVisible Property' in the on-line documentation)**, and **AllowGroup ('AllowGroup Property' in the on-line documentation)** properties.
2. Specify the main group bar color with the **GroupBarBackColor ('GroupBarBackColor Property' in the on-line documentation)** property.
3. Specify colors for subgroups with the **GroupInfo ('GroupInfo Class' in the on-line documentation)** class.

Example

This example sets the appearance for the group bar and the grouped rows.

C#

```
FpSpread1.ActiveSheetView.AllowColumnMove = true;
FpSpread1.ActiveSheetView.GroupBarVisible = true;
FpSpread1.ActiveSheetView.GroupBarBackColor = Color.Salmon;
FpSpread1.ActiveSheetView.GroupBarHeight = 50;
FpSpread1.ActiveSheetView.GroupMaximumLevel = 5;
FpSpread1.ActiveSheetView.AllowGroup = true;
FarPoint.Web.Spread.GroupInfo gi = new FarPoint.Web.Spread.GroupInfo();
gi.BackColor = Color.Yellow;
FarPoint.Web.Spread.GroupInfo gi2 = new FarPoint.Web.Spread.GroupInfo();
gi2.BackColor = Color.Green;
FarPoint.Web.Spread.GroupInfoCollection gic = new
FarPoint.Web.Spread.GroupInfoCollection();
gic.AddRange(new FarPoint.Web.Spread.GroupInfo[] {gi, gi2});
FpSpread1.ActiveSheetView.GroupInfos.Add(gic[0]);
```

VB

```
FpSpread1.ActiveSheetView.AllowColumnMove = True
FpSpread1.ActiveSheetView.GroupBarVisible = True
```

```
FpSpread1.ActiveSheetView.GroupBarBackColor = Color.Salmon
FpSpread1.ActiveSheetView.GroupBarHeight = 50
FpSpread1.ActiveSheetView.GroupMaximumLevel = 5
FpSpread1.ActiveSheetView.AllowGroup = True
Dim gi As New FarPoint.Web.Spread.GroupInfo
gi.BackColor = Color.Yellow
Dim gi2 As New FarPoint.Web.Spread.GroupInfo
gi2.BackColor = Color.Green
Dim gic As New FarPoint.Web.Spread.GroupInfoCollection()
gic.Add(gi)
FpSpread1.ActiveSheetView.GroupInfos.Add(gic(0))
```

Using the Spread Designer

1. Select the **Settings** menu.
2. Select the **Group** icon in the **Other Settings** section.
3. Set the various properties.
4. Click **OK** to close the dialog.
5. Use the **View** menu to show or hide the group bar.
6. Click **Apply and Exit** to close the Spread Designer.

Customizing the Group Bar

You can customize the appearance of the group bar at the top of the grouping display and you can hide or display the grouping bar at the top of the sheet. The properties on the sheet (**SheetView ('SheetView Class' in the on-line documentation)** object) include:

SheetView Property	Description
GroupBarBackColor ('GroupBarBackColor Property' in the on-line documentation)	Set the background color of the grouping bar
GroupBarHeight ('GroupBarHeight Property' in the on-line documentation)	Set the height of the grouping bar
GroupBarVisible ('GroupBarVisible Property' in the on-line documentation)	Set whether to display the grouping bar
GroupMaximumLevel ('GroupMaximumLevel Property' in the on-line documentation)	Set the maximum number of levels of grouping allowed

Using Code

1. Set the **AllowColumnMove ('AllowColumnMove Property' in the on-line documentation)**, **GroupBarVisible ('GroupBarVisible Property' in the on-line documentation)**, and **AllowGroup ('AllowGroup Property' in the on-line documentation)** properties.
2. Specify the main group bar color with the **GroupBarBackColor ('GroupBarBackColor Property' in the on-line documentation)** property.
3. Specify colors for subgroups with the **GroupInfo ('GroupInfo Class' in the on-line documentation)** class.

Example

This example sets the appearance for the group bar and the grouped rows.

C#

```
FpSpread1.ActiveSheetView.AllowColumnMove = true;
FpSpread1.ActiveSheetView.GroupBarVisible = true;
FpSpread1.ActiveSheetView.GroupBarBackColor = Color.Salmon;
FpSpread1.ActiveSheetView.GroupBarHeight = 50;
FpSpread1.ActiveSheetView.GroupMaximumLevel = 5;
FpSpread1.ActiveSheetView.AllowGroup = true;
FarPoint.Web.Spread.GroupInfo gi = new FarPoint.Web.Spread.GroupInfo();
gi.BackColor = Color.Yellow;
FarPoint.Web.Spread.GroupInfo gi2 = new FarPoint.Web.Spread.GroupInfo();
gi2.BackColor = Color.Green;
FarPoint.Web.Spread.GroupInfoCollection gic = new
FarPoint.Web.Spread.GroupInfoCollection();
gic.AddRange(new FarPoint.Web.Spread.GroupInfo[] {gi, gi2});
FpSpread1.ActiveSheetView.GroupInfos.Add(gic[0]);
```

VB

```
FpSpread1.ActiveSheetView.AllowColumnMove = True
FpSpread1.ActiveSheetView.GroupBarVisible = True
FpSpread1.ActiveSheetView.GroupBarBackColor = Color.Salmon
FpSpread1.ActiveSheetView.GroupBarHeight = 50
FpSpread1.ActiveSheetView.GroupMaximumLevel = 5
FpSpread1.ActiveSheetView.AllowGroup = True
Dim gi As New FarPoint.Web.Spread.GroupInfo
gi.BackColor = Color.Yellow
Dim gi2 As New FarPoint.Web.Spread.GroupInfo
gi2.BackColor = Color.Green
Dim gic As New FarPoint.Web.Spread.GroupInfoCollection()
gic.Add(gi)
FpSpread1.ActiveSheetView.GroupInfos.Add(gic(0))
```

Using the Spread Designer

1. Select the **Settings** menu.
2. Select the **Group** icon in the **Other Settings** section.
3. Set the various properties.
4. Click **OK** to close the dialog.
5. Use the **View** menu to show or hide the group bar.
6. Click **Apply and Exit** to close the Spread Designer.

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.

More information about creating a custom group is available on our online technical support forum (see the Read Me for more information).

Compatibility with Other Features

The grouping feature affects the visual 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) 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, etc.), you must apply the formatting after grouping.

Features Influenced by Grouping

These features do not interoperate with grouping in Spread.

- 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 event.
- Conditional Formatting: Grouping and conditional formatting do not work together. Conditional formatting requires the default data model. Thus, these features do not work together.
- Formulas: Grouping and formulas do not work together. Formulas requires the default data model. Thus, these features do not work with grouping.
- 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.
- Count: 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 **TargetModel ('TargetModel Property' in the on-line documentation)** property of the **GroupDataModel ('GroupDataModel Class' in the on-line documentation)** class.

These features work with grouping in Spread.

- Grouping and hidden columns work together.
- Printing and exporting work with grouping.

Customizing 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 various ways of performing a sort with various properties and methods for each type of sorting. In general, sorting data can be done by any of these ways:

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.

You can sort entire rows or columns in a sheet. 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. The 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.

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. When you sort data, only the data model is getting sorted. The SelectionModel does not get sorted. For more information on the models, refer to **Using Sheet Models**.

Sorting executed by clicking column headers sorts only the displayed data and does not affect the order of actual data in the data model. So 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.

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

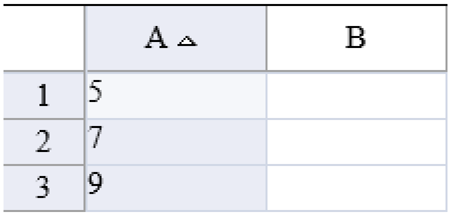
Note: Cell spans become invisible when sorting a sheet.

You can allow the user to sort with the **AllowSort ('AllowSort Property' in the on-line documentation)** property. See the following topic for more information:

- **Allowing User Sorting**

Allowing User Sorting

You can allow the user to sort with the **AllowSort ('AllowSort Property' in the on-line documentation)** property. The following image shows the column after the user has double-clicked on the header.



	A ▵	B
1	5	
2	7	
3	9	

Using Code

Use the **AllowSort ('AllowSort Property' in the on-line documentation)** property to allow user sorting.

Example

The following example sets the **AllowSort ('AllowSort Property' in the on-line documentation)** property.

C#

```
FarPoint.Web.Spread.SheetView sv = new FarPoint.Web.Spread.SheetView();
FpSpread1.ActiveSheetView.SetValue(0, 0, 9);
FpSpread1.ActiveSheetView.SetValue(1, 0, 5);
FpSpread1.ActiveSheetView.SetValue(2, 0, 7);
sv = FpSpread1.ActiveSheetView);
sv.AllowSort = true;
```

VB

```
Dim sv As FarPoint.Web.Spread.SheetView
FpSpread1.ActiveSheetView.SetValue(0, 0, 9)
FpSpread1.ActiveSheetView.SetValue(1, 0, 5)
```

```
FpSpread1.ActiveSheetView.SetValue(2, 0, 7)
sv = FpSpread1.ActiveSheetView
sv.AllowSort = True
```

Using the Spread Designer

1. Select the **Settings** menu in the **Sheet Settings** section.
2. Select the **General** icon.
3. Set the **AllowSort** check box.
4. Click **OK** to close the dialog.
5. Click **Apply and Exit** to close the Spread Designer.

Customizing Interaction with Cells

You can customize the user interaction with individual cells (or a range of cells). To customize this aspect of user interaction, you may perform the following tasks.

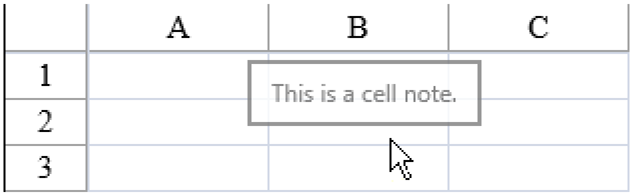
- **Adding a Note to a Cell**
- **Adding a Tag to a Cell**
- **Locking a Cell**
- **Using Conditional Formatting in Cells**

For information on selections of cells, refer to **Customizing Selections of Cells**.

For information on how to change the appearance of cells, refer to **Customizing the Appearance of a Cell**. For information on setting the cell type, refer to **Customizing with Cell Types**.

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. When the pointer is over a cell that has a note, the note text displays in a box next to the cell. Notes cannot be placed in cells in the column or row headers.



Using the Properties Window

1. At design time, in the **Properties** window, select the FpSpread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. Select the **Cells** drop-down.
5. Select the cells for which you want to set the note.
6. Set the **Note** property.
7. Select **OK**.

Using a Shortcut

Set the **Note ('Note Property' in the on-line documentation)** property for the cells in the sheet of the component.

Example

This example code sets the **Note ('Note Property' in the on-line documentation)** property for a range of Cell objects.

C#

```
FpSpread1.Sheets[0].ColumnCount = 4;
FpSpread1.Sheets[0].RowCount = 4;
FpSpread1.Sheets[0].Cells[1, 1, 3, 3].Note = "This is the note that describes the
value.";
FpSpread1.Sheets[0].Cells[1, 1, 3, 3].Value = "Value Here";
```

VB

```
FpSpread1.Sheets(0).ColumnCount = 4
FpSpread1.Sheets(0).RowCount = 4
FpSpread1.Sheets(0).Cells(1, 1, 3, 3).Note = "This is the note that describes the
value."
FpSpread1.Sheets(0).Cells(1, 1, 3, 3).Value = "Value Here"
```

Using Code

Set the **Note ('Note 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 **Note ('Note Property' in the on-line documentation)** property for a range of Cell objects.

C#

```
FarPoint.Web.Spread.Cell range1;
range1 = fpSpread1.ActiveSheetView.Cells[1, 1, 3, 3];
range1.Value = "Value Here";
range1.Note = "This is the note that describes the value.";
```

VB

```
Dim range1 As FarPoint.Web.Spread.Cell
range1 = fpSpread1.ActiveSheetView.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. Another way is to select the **Cells** property and click on the button to call up the **Cell, Column, and Row** editor and select the cells in that editor.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Adding a Tag to a Cell

You can add a 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.

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.

Using the Properties Window

1. At design time, in the **Properties** window, select the FpSpread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. Select the **Cells** drop-down.
5. Select the cells for which you want to set the tag.
6. Set the **Tag** property.
7. Select **OK**.

Using a Shortcut

Set the **Tag ('Tag Property' in the on-line documentation)** property for the cells in the sheet of the component.

Example

This example code sets the **Tag ('Tag Property' in the on-line documentation)** property for a range of **Cell ('Cell Class' in the on-line documentation)** 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 ('Cell Class' in the on-line documentation)** objects.

C#

```
FarPoint.Web.Spread.Cell range1;
range1 = fpSpread1.ActiveSheetView.Cells[1, 1, 3, 3];
range1.Value = "Value Here";
range1.Tag = "This is the tag.";
```

VB

```
Dim range1 As FarPoint.Web.Spread.Cell
range1 = fpSpread1.ActiveSheetView.Cells(1, 1, 3, 3)
range1.Value = "Value Here"
range1.Tag = "This is the tag."
```

Using the Spread Designer

1. In the work area, select the cell or cells for which you want to set the tag to display.
2. In the properties list (in the **Misc** group), select the **Tag** property and type in the text.
   Another way is to select the **Cells** property and click on the button to display 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.

Locking a Cell

You can lock a cell or range of cells and make it unavailable for editing by the end user.

You can lock cells using the Locked 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)**, **AlternatingRow ('AlternatingRow Class' in the on-line documentation)**, or **SheetView ('SheetView Class' in the on-line documentation)** objects.

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.

Using the Properties Window

1. At design time, in the **Properties** window, select the FpSpread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. Select the **Cells** drop-down.
5. Select the cells that you wish to lock.
6. Set the **Locked** property.
7. Select **OK**.

Using a Shortcut

Using the **Locked** property for the **Cell ('Cell Class' in the on-line documentation)**, **Column ('Column Class' in the on-line documentation)**, or **Row ('Row Class' in the on-line documentation)** object, you can mark some cells as locked. The **Protect ('Protect Property' in the on-line documentation)** property, for the sheet, must be set to true if you want the cells 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 some columns of cells and then unlock some of the cells in one row.

C#

```
FpSpread1.ActiveSheetView.Protect = true;
FpSpread1.ActiveSheetView.Columns[0, 3].Locked = true;
FpSpread1.ActiveSheetView.Cells[1,1,1,2].Locked = false;
```

VB

```
FpSpread1.ActiveSheetView.Protect = True
FpSpread1.ActiveSheetView.Columns[0, 3].Locked = True
FpSpread1.ActiveSheetView.Cells[1,1,1,2].Locked = False
```

Using Code

Using the **Locked** property for the **Cell ('Cell Class' in the on-line documentation)**, **Column ('Column Class' in the on-line documentation)**, or **Row ('Row Class' in the on-line documentation)** object, you can mark some cells as locked. The **Protect ('Protect Property' in the on-line documentation)** property, for the sheet, must be set to True if you want the cells 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 some columns of cells and then unlock some of the cells in one row.

C#

```
FpSpread1.ActiveSheetView.Protect = true;
FpSpread1.ActiveSheetView.LockBackColor = Color.LightCyan;
FpSpread1.ActiveSheetView.LockForeColor = Color.Green;

FarPoint.Web.Spread.Column columnobj;
columnobj = fpSpread1.ActiveSheetView.Columns[0, 3];
columnobj.Locked = true;
FarPoint.Web.Spread.Cell cellobj;
cellobj = fpSpread1.ActiveSheetView.Cells[1,1,1,2];
cellobj.Locked = false;

FpSpread1.ActiveSheetView.Cells[1,0,1,4].Text = "First Five";
```

VB

```
FpSpread1.ActiveSheetView.Protect = True
FpSpread1.ActiveSheetView.LockBackColor = Color.LightCyan
FpSpread1.ActiveSheetView.LockForeColor = Color.Green

Dim columnobj As FarPoint.Web.Spread.Column
columnobj = fpSpread1.ActiveSheetView.Columns(0, 3)
columnobj.Locked = True
Dim cellobj As FarPoint.Web.Spread.Cell
cellobj = fpSpread1.ActiveSheetView.Cells(1,1,1,2)
cellobj.Locked = False

FpSpread1.ActiveSheetView.Cells(1,0,1,4).Text = "First Five"
```

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).
2. In the **Misc** section, select the **Locked** property and choose **True**.
(Another way of doing that 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.)
3. Choose the sheet from the drop-down combo to the right of the designer. From the properties list (in the **Misc**

section), set the **Protect** property.

4. In the **Properties** window in the **SheetView Collection Editor**, in the **Misc** section, 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.

Using the Spread Designer

The following steps list a different way to lock cells in the designer.

1. In the work area, select the cell or cells that you want to lock.
2. Select the **Home** menu.
3. Select the **Lock** icon from the **Editing** section.
4. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Using Conditional Formatting in 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 may perform the following tasks:

- **Creating Conditional Formatting with Rules**
- **Conditional Formatting of Cells**

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)**
- **ErrorConditionalFormattingRule Class (on-line documentation)**
- **FormulaConditionalFormattingRule 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.

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

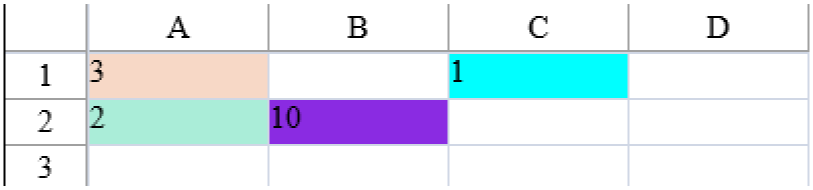
- **Color Scale Rules**
- **Data Bar Rule**
- **Highlighting Rules**
- **Icon Set Rule**
- **Top or Average Rules**

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:



	A	B	C	D
1	3		1	
2	2	10		
3				

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#

```
protected void Page_Load(object sender, System.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;
}

protected void Button1_Click(object sender, EventArgs e)
{
FarPoint.Web.Spread.Model.CellRange celRange1 = new
FarPoint.Web.Spread.Model.CellRange(0, 0, 3, 3);
FarPoint.Web.Spread.ThreeColorScaleConditionalFormattingRule rule = new
FarPoint.Web.Spread.ThreeColorScaleConditionalFormattingRule(Color.Aqua, Color.Bisque,
Color.BlueViolet);
FpSpread1.Sheets[0].SetConditionalFormatting(new FarPoint.Web.Spread.Model.CellRange[]
{ celRange1 }, rule);
}
```

VB

```
Private Sub Page_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

Protected Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
Dim celRange1 As New FarPoint.Web.Spread.Model.CellRange(0, 0, 3, 3)
Dim rule As New
FarPoint.Web.Spread.ThreeColorScaleConditionalFormattingRule(Drawing.Color.Aqua,
Drawing.Color.Bisque, Drawing.Color.BlueViolet)
FpSpread1.Sheets(0).SetConditionalFormatting(New FarPoint.Web.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.

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.

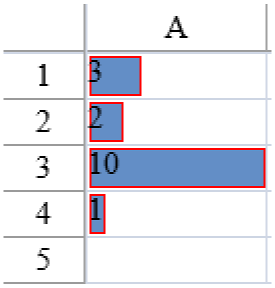
Valid percent values are from 0 (zero) to 100. Percent values should not use a percent sign. Use a percentage when you want to visualize all values proportionally because the distribution of values is proportional.

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.

The following image displays data bars in a cell range:



	A
1	3
2	2
3	10
4	1
5	

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#

```
protected void Page_Load(object sender, System.EventArgs e)
{
FpSpread1.Sheets[0].RowCount = 5;
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;
}

protected void Button1_Click(object sender, EventArgs e)
{
FarPoint.Web.Spread.DatabarConditionalFormattingRule d = new
FarPoint.Web.Spread.DatabarConditionalFormattingRule();
d.BorderColor = Color.Red;
d.ShowBorder = true;
d.Minimum = new FarPoint.Web.Spread.ConditionalFormattingValue(0,
FarPoint.Web.Spread.ConditionalFormattingValueType.Number);
d.Maximum = new FarPoint.Web.Spread.ConditionalFormattingValue(15,
FarPoint.Web.Spread.ConditionalFormattingValueType.Max);
FpSpread1.ActiveSheetView.SetConditionalFormatting(0, 0, 4, 1, d);
}
```

VB

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
FpSpread1.Sheets(0).RowCount = 5
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

Protected Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
Dim d As New FarPoint.Web.Spread.DatabarConditionalFormattingRule()
d.BorderColor = Drawing.Color.Red
d.ShowBorder = True
d.Minimum = New FarPoint.Web.Spread.ConditionalFormattingValue(0,
FarPoint.Web.Spread.ConditionalFormattingValueType.Number)
d.Maximum = New FarPoint.Web.Spread.ConditionalFormattingValue(15,
FarPoint.Web.Spread.ConditionalFormattingValueType.Max)
FpSpread1.ActiveSheetView.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 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)**

Using Code

Set the properties of the rule class and then 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#

```
protected void Page_Load(object sender, System.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;
   }
```

```
protected void Button1_Click(object sender, EventArgs e)
        {
            FarPoint.Web.Spread.BetweenValuesConditionalFormattingRule between = new
FarPoint.Web.Spread.BetweenValuesConditionalFormattingRule(false);
            between.FirstValue = 10;
            between.SecondValue = 20;
            between.IsNotBetween = true;
            between.BackColor = Color.Bisque;
            FpSpread1.ActiveSheetView.SetConditionalFormatting(1, 1, between);
        }
```

VB

```
Private Sub Page_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
```

```
Protected Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
            Dim between As New
FarPoint.Web.Spread.BetweenValuesConditionalFormattingRule(False)
            between.FirstValue = 10
            between.SecondValue = 20
            between.IsNotBetween = True
            between.BackColor = Drawing.Color.Bisque
            FpSpread1.ActiveSheetView.SetConditionalFormatting(1, 1, between)
        End Sub
```

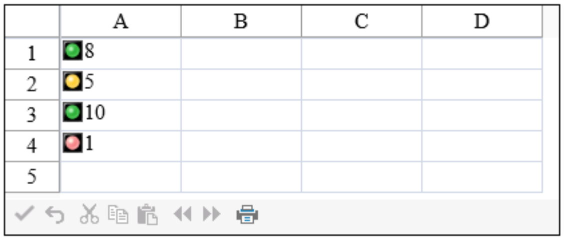
Using the Spread Designer

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.



	A	B	C	D
1	● 8			
2	● 5			
3	● 10			
4	● 1			
5				

Using Code

Set the properties of the **IconSetConditionalFormattingRule ('IconSetConditionalFormattingRule Class' in the on-line documentation)** class and then 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#

```
protected void Page_Load(object sender, System.EventArgs e)
 {
FpSpread1.Sheets[0].RowCount = 5;
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;
 }
protected void Button1_Click(object sender, EventArgs e)
{
FarPoint.Web.Spread.Model.CellRange celRange1 = new FarPoint.Web.Spread.Model.CellRange(0, 0, 4, 1);
FarPoint.Web.Spread.IconSetConditionalFormattingRule rule = new
FarPoint.Web.Spread.IconSetConditionalFormattingRule(FarPoint.Web.Spread.ConditionalFormattingIconSetStyle.ThreeRimmedTrafficLights);
FpSpread1.Sheets[0].SetConditionalFormatting(new FarPoint.Web.Spread.Model.CellRange[] { celRange1 }, rule);
}
```

VB

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
FpSpread1.Sheets(0).RowCount = 5
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
Protected Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
```

```
Dim celRange1 As New FarPoint.Web.Spread.Model.CellRange(0, 0, 4, 1)
Dim rule As New
FarPoint.Web.Spread.IconSetConditionalFormattingRule(FarPoint.Web.Spread.ConditionalFormattingIconSetStyle.ThreeRimmedTrafficLights)
FpSpread1.Sheets(0).SetConditionalFormatting(New FarPoint.Web.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 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 options are available:

- top 10
- top 10%
- bottom 10
- bottom 10%
- above average
- below average

Using Code

Set the properties of the rule class and then 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#

```
protected void Page_Load(object sender, System.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;
}

protected void Button1_Click(object sender, EventArgs e)
{
//Average CF
FarPoint.Web.Spread.AverageConditionalFormattingRule average = new
FarPoint.Web.Spread.AverageConditionalFormattingRule(true, true);
average.IsAbove = true;
average.IsIncludeEquals = true;
average.StandardDeviation = 5;
average.FontStyle = new
FarPoint.Web.Spread.SpreadFontStyle(FarPoint.Web.Spread.UnderlineFontStyle.None);
average.FontStyle.RegularBoldItalic =
FarPoint.Web.Spread.RegularBoldItalicFontStyle.Bold;
FpSpread1.ActiveSheetView.SetConditionalFormatting(1, 1, average);
}
```

VB

```
Private Sub Page_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

Protected Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
'Average CF
Dim average As New FarPoint.Web.Spread.AverageConditionalFormattingRule(True, True)
average.IsAbove = True
average.IsIncludeEquals = True
average.StandardDeviation = 5
average.FontStyle = New
FarPoint.Web.Spread.SpreadFontStyle(FarPoint.Web.Spread.UnderlineFontStyle.None)
average.FontStyle.RegularBoldItalic =
FarPoint.Web.Spread.RegularBoldItalicFontStyle.Bold
FpSpread1.ActiveSheetView.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.

Conditional Formatting of Cells

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 Custom Style for Cells**.

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

Using Code

Use the **SetConditionalFormat ('SetConditionalFormat Method' in the on-line documentation)** method to create a conditional format.

Example

The following example changes the color of the cell if the value is greater than 14.

C#

```
FarPoint.Web.Spread.NamedStyle ns = new FarPoint.Web.Spread.NamedStyle();
ns.BackColor = Color.Crimson;
ns.Name = "mystyle";
FpSpread1.NamedStyles.Add(ns);
FpSpread1.Sheets[0].SetConditionalFormat(0, 0, ns,
FarPoint.Web.Spread.ComparisonOperator.GreaterThan, "14");
```

VB

```
Dim ns As New FarPoint.Web.Spread.NamedStyle
ns.BackColor = Drawing.Color.Azure
ns.Name = "mystyle"
FpSpread1.NamedStyles.Add(ns)
FpSpread1.Sheets(0).SetConditionalFormat(0, 0, ns,
FarPoint.Web.Spread.ComparisonOperator.GreaterThan, "14")
```

Customizing Selections of Cells

You can customize what the user can select and how it appears. To customize aspects of selections, you can perform the following tasks:

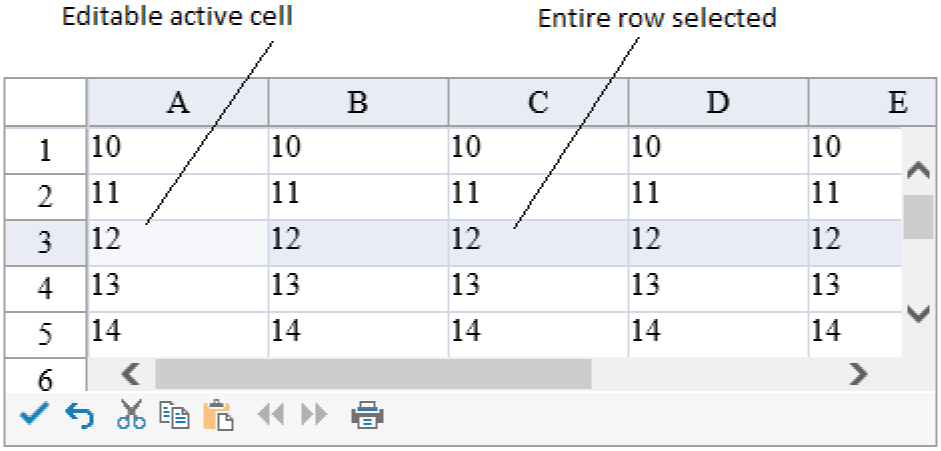
- **Specifying What the User Can Select**
- **Working with Selections of Cells**
- **Customizing the Appearance of Selections**

For more information about the underlying selection model, refer to **Understanding the Selection Model**.

Specifying What the User Can Select

By default, the component allows users to select a cell, a column, a row, a range of cells, or the entire sheet. You can customize what the user can select by working with the operation mode of the sheet (**OperationMode ('OperationMode Property' in the on-line documentation)** property). The settings are based on the **OperationMode ('OperationMode Enumeration' in the on-line documentation)** enumeration. You can specify what the user is allowed to select in normal operation mode with the **SelectionBlockOptions ('SelectionBlockOptions Property' in the on-line documentation)** property. You can allow the user to select multiple blocks in normal operation mode with the **SelectionPolicy ('SelectionPolicy Property' in the on-line documentation)** property.

The settings of the **OperationMode ('OperationMode Property' in the on-line documentation)** property affect user interaction with the sheet, that is, what the user can select, but not necessarily what the application can select.



Editable active cell

Entire row selected

	A	B	C	D	E
1	10	10	10	10	10
2	11	11	11	11	11
3	12	12	12	12	12
4	13	13	13	13	13
5	14	14	14	14	14
6					

The following table summarizes the options available for specifying what users can select and edit on the sheet:

User can select	User can edit	OperationMode Setting
Cell, row, column, any range of cells, entire sheet	Active cell	Normal
Only one row	Active Cell	RowMode
Only one row	Nothing	SingleSelect
Nothing	Nothing	ReadOnly
Multiple contiguous rows	Nothing	MultiSelect
Multiple discontiguous rows	Nothing	ExtendedSelect

Using the Properties Window

1. At design time, in the **Properties** window, select the FpSpread 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. Click **OK** to close the editor.

Using a Shortcut

1. To set the overall user interaction mode of the sheet, set the Sheet’s **OperationMode ('OperationMode Property' in the on-line documentation)** property.

Example

This example code sets the sheet to allow users to select only rows and only edit the active cell.

C#

```
// Set the operation mode and let users select only rows.
fpSpread1.Sheets[0].OperationMode = FarPoint.Web.Spread.OperationMode.RowMode;
```

VB

```
' Set the operation mode and let users select only rows.
FpSpread1.Sheets(0).OperationMode = FarPoint.Web.Spread.OperationMode.RowMode
```

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 ('SheetView Class' in the on-line documentation)** object.
2. 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 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.

C#

```
// Set operation mode and let users select only a row.
FarPoint.Web.Spread.SheetView newsheet = new FarPoint.Web.Spread.SheetView();
newsheet.OperationMode = FarPoint.Web.Spread.OperationMode.RowMode;
// Assign the SheetView object to a sheet.
fpSpread1.Sheets[0] = newsheet;
```

VB

```
' Set operation mode and let users select only a row.
Dim newsheet As New FarPoint.Web.Spread.SheetView()
newsheet.OperationMode = FarPoint.Web.Spread.OperationMode.RowMode
' 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 the **General** icon, then select one of the choices from the **Operation Mode** area.
3. Click **OK** to close the **Sheet Settings** dialog.
4. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Working with Selections of Cells

Besides selections that can be allowed by the end user, you can work with selections to the sheet using code. With code, you can add a selection or remove one or all of the existing selections. Use the **AddSelection ('AddSelection Method' in the on-line documentation)** and **ClearSelection ('ClearSelection Method' in the on-line documentation)** methods to add and remove selections using code in the selection model.

For more information on working with the models, refer to **Understanding the Selection Model**.

For information on changing the appearance of selected cells, refer to **Customizing the Appearance of Selections**.

Using Code

You can get a selection on the server side with the following code.

Example

The following code gets the first selection.

C#

```
FarPoint.Web.Spread.Model.CellRange cr = FarPoint.Web.Spread.Model.CellRange;
cr = FpSpread1.Sheets[0].SelectionModel(0);
```

VB

```
Dim cr as FarPoint.Web.Spread.Model.CellRange
cr = FpSpread1.Sheets(0).SelectionModel(0)
```

Customizing the Appearance of Selections

Selections have a default appearance provided by the component and the selection renderer. You can change that appearance, including the background and text colors as well as the border. By default, when you click on a row header, the entire row is selected and when you click on a column header, the entire column is selected. The active cell retains a different appearance to show you which cell is active. You can change the selection behavior by setting the **OperationMode ('OperationMode Property' in the on-line documentation)** for the sheet.

Set the **SheetView ('SheetView Class' in the on-line documentation)** object's **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 (**SelectionForeColor ('SelectionForeColor Property' in the on-line documentation)** only applies to downlevel browsers). Set the SheetView object's **SelectionBorder ('SelectionBorder Property' in the on-line documentation)** to specify the border for selections (downlevel browser only). Assign the SheetView object you have created to one of the sheets in the component.

You can also change the appearance by setting the selection colors in a custom skin and applying that skin to the sheet. For more information about skins, refer to **Creating a Skin for Sheets** and **Applying a Skin to a Sheet**.

Using the Properties Window

1. At design time, in the **Properties** window, select the FpSpread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. Set the various selection properties.
5. Click **OK** to close the editor.

Using a Shortcut

1. Set the **SelectionBackColorStyle ('SelectionBackColorStyle Property' in the on-line documentation)** property.
2. Set the **SelectionBackColor ('SelectionBackColor Property' in the on-line documentation)** property.
3. Set the **SelectionPolicy ('SelectionPolicy Property' in the on-line documentation)** property.

Example

This example code sets the selection backcolor and policies.

C#

```
FpSpread1.Sheets[0].SelectionBackColorStyle =
FarPoint.Web.Spread.SelectionBackColorStyles.SelectionBackColor;
FpSpread1.Sheets[0].SelectionBackColor = Color.SaddleBrown;
FpSpread1.Sheets[0].SelectionPolicy =
FarPoint.Web.Spread.Model.SelectionPolicy.MultiRange;
```

VB

```
FpSpread1.Sheets(0).SelectionBackColorStyle =
FarPoint.Web.Spread.SelectionBackColorStyles.SelectionBackColor
FpSpread1.Sheets(0).SelectionBackColor = Color.SaddleBrown
FpSpread1.Sheets(0).SelectionPolicy =
FarPoint.Web.Spread.Model.SelectionPolicy.MultiRange
```

Using the Spread Designer

1. Select the **Settings** menu.
2. Select the **Colors** icon from the **Sheet Settings** section.
3. Set the various selection options. Click **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.

Managing Printing

You can print the data area of the spreadsheet. This includes the headers and the cells, but not the tool bars or scroll bars of the component. You can also specify headers and footers for the printed pages.

- **Printing a Spreadsheet**
- **Adding Headers and Footers to Printed Pages**

You can also print to a PDF file. For more information, see **Saving to a PDF File**.

Printing a Spreadsheet

You can print the data area of a spreadsheet by clicking **Print** (or the **Print** icon) in the command bar of the component. The appearance of the print button depends on the setting in the command bar and the type of buttons. For more information refer to **Customizing the Command Buttons**.

At run time, when you click **Print**, the standard print dialog for your machine appears and you can choose various printer settings before clicking OK to print. You can specify page breaks with the column **PageBreak ('PageBreak Property' in the on-line documentation)** property or the row **PageBreak ('PageBreak Property' in the on-line documentation)** property when printing to the client.

In addition to the print button, there is an **IsPrint ('IsPrint Property' in the on-line documentation)** property. Setting this to true brings up a preview of what is to be printed.

> The **ShowColumnHeader ('ShowColumnHeader Property' in the on-line documentation)** and **ShowRowHeader ('ShowRowHeader Property' in the on-line documentation)** properties in the **PrintInfo ('PrintInfo Class' in the on-line documentation)** class apply when printing or saving to PDF.

You can also use the client-side **Print (on-line documentation)** method. For more information, refer to the **Client-Side Scripting Reference (on-line documentation)**.

Using Code

You can set the **FpSpread ('FpSpread Class' in the on-line documentation)** component **IsPrint ('IsPrint Property' in the on-line documentation)** property in code.

Example

The following code sets the **IsPrint ('IsPrint Property' in the on-line documentation)** property.

C#

```
FpSpread1.IsPrint = true;
```

VB

```
FpSpread1.IsPrint = True
```

Using the Spread Designer

1. Select the **File** menu.
2. Select **Print**, **SaveToPDF**, or **Print Preview** from the **Print** menu.
3. Click **Apply and Exit** to close the Spread Designer.

Adding Headers and Footers to Printed Pages

You can add headers and footers to the printed pages by using the Content property in the PrintSheet event. You can use HTML characters in the header or footer string. An example of this would be the line break tag, <BR>, for breaking to a new line.

Using Code

Set the **Content** property in the **PrintSheet ('PrintSheet Event' in the on-line documentation)** event.

Example

The following code adds a header and footer to the printed page.

C#

```
private void FpSpread1_PrintSheet(object sender, FarPoint.Web.Spread.PrintEventArgs e)
   {
     if (e.Header == true)
     {
      e.Content = "Header<BR>Test";
     }
    if (e.Header == false)
    {
     e.Content = "Footer";
    }
   }
```

VB

```
Private Sub FpSpread1_PrintSheet(ByVal sender As Object, ByVal e As
FarPoint.Web.Spread.PrintEventArgs) Handles FpSpread1.PrintSheet
   If e.Header = True Then
    e.Content = "Header<BR>Test"
   End If
   If e.Header = False Then
    e.Content = "Footer"
   End If
End Sub
```

Customizing the Appearance

You can customize the appearance of various parts of the FpSpread component.

The tasks that relate to setting the appearance of parts of the component include:

- **Customizing the Appearance of the Overall Component**
- **Customizing the Appearance of the Sheet**
- **Customizing the Appearance of Rows and Columns**
- **Customizing the Appearance of Headers**
- **Customizing the Appearance of a Cell**

For information on customizing the appearance of selections, refer to **Customizing the Appearance of Selections**

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

For information on the hierarchical display of data, refer to **Displaying Data as a Hierarchy**.

For information on the underlying style model, refer to **Understanding the Style Model**.

Customizing the Appearance of the Overall Component

You can set several aspects that determine the appearance of the component on the HTML page. These include:

- **Customizing the Dimensions of the Component**
- **Customizing the Outline of the Component**
- **Customizing the Default Initial Appearance**
- **Resetting Parts of the Interface**
- **Using the jQuery Theme Roller with Spread**

Other tasks that are related to the appearance of the overall component include:

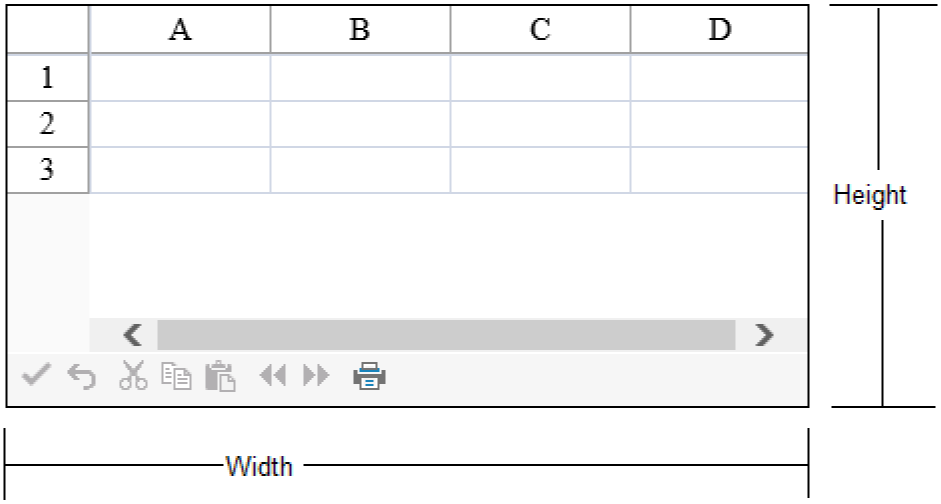
- **Setting the Background Color of the Sheet**
- **Displaying Grid Lines on the Sheet**
- **Displaying the Sheet Names**
- **Adding a Title and Subtitle to a Sheet**

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

Customizing the Dimensions of the Component

You can set the overall dimensions of the component and these remain the same regardless of the size of the sheet or the amount of navigation bars. If there are more rows in a sheet than can be displayed, then the component creates pages inside the component to allow you access to those rows. Refer to **Customizing Page Navigation** for more information. If you display hierarchy information and page navigation bars, then the amount of space dedicated to the sheet is smaller. All the tool bars, scroll bars, and sheet appear inside the overall dimensions of the component. Refer to **Customizing the Tool Bars** for more information.

The following figure shows the dimensions that you can set by setting the number of pixels for each.



Using the Properties Window

Set the dimensions at design time with the Properties window of Visual Studio .NET.

1. Select the 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 using the **Height** or **Width** properties of **FpSpread ('FpSpread Class' in the on-line documentation)** class or both. The default for the unit of measurement is pixels.

Example

This example shows how to set the height of the component to 200 pixels and the width to 400 pixels.

C#

```
FpSpread1.Height = 200;
FpSpread1.Width = 400;
```

VB

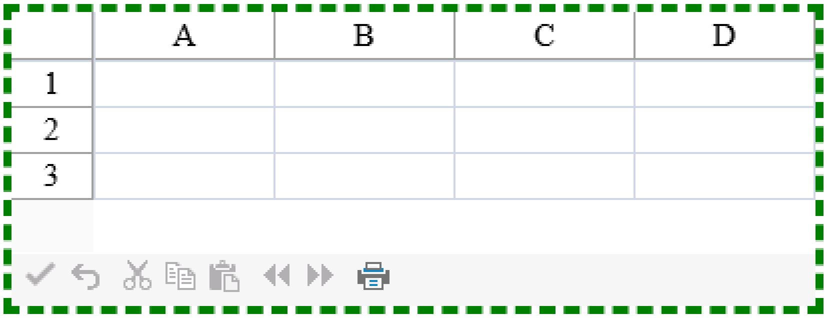
```
FpSpread1.Height = System.Web.UI.WebControls.Unit.Pixel(200)
FpSpread1.Width = System.Web.UI.WebControls.Unit.Pixel(400)
```

Customizing the Outline of the Component

You can set several aspects of the outline (or border) of the component. These aspects include:

- the color of the outline
- the line style of the outline
- the width (or thickness) of the outline

Here is a picture of an example of the outline of the component changed to be a thick dashed green outline. The example below shows the code for these customizations.



	A	B	C	D
1				
2				
3				

Using the Properties Window

Set the border at design time with the Properties window of Visual Studio .NET.

1. Select the component.
2. With the **Properties** window open, in the **Appearance** category, select the **BorderColor** property, the **BorderStyle** property, or the **BorderWidth** property. For the **BorderColor** and **BorderStyle** properties, select a value from the drop-down list. For the **BorderWidth**, type in a value; the unit is pixels. Press **Enter**.
   The new property is now set.
   Refer to the Microsoft .NET Framework documentation for setting the units to something other than the default, which is pixels.

Using Code

Add a line of code that sets the specific border property. The default for the unit of thickness is pixels. For more information, refer to the **BorderColor ('BorderColor Property' in the on-line documentation)**, **BorderStyle ('BorderStyle Property' in the on-line documentation)**, and **BorderWidth** properties in the **Border ('Border Class' in the on-line documentation)** class.

Example

This example shows how to create a green dashed outline that is four pixels thick around the entire component. To see the results, see the figure above.

C#

```
FpSpread1.BorderColor = Drawing.Color.Green;
FpSpread1.BorderStyle = BorderStyle.Dashed;
FpSpread1.BorderWidth = System.Web.UI.WebControls.Unit.Pixel(4);
```

VB

```
FpSpread1.BorderColor = Drawing.Color.Green
FpSpread1.BorderStyle = BorderStyle.Dashed
FpSpread1.BorderWidth = System.Web.UI.WebControls.Unit.Pixel(4)
```

Customizing the Default Initial Appearance

You can set the default initial appearance.

The following API members are involved in the default appearance.

- **CommandBarInfo ('CommandBarInfo Class' in the on-line documentation)** Background Property
- **DefaultSkins ('DefaultSkins Class' in the on-line documentation)** Classic Property

- **DefaultSkins ('DefaultSkins Class' in the on-line documentation)** Default Property
- FpSpread ScrollBar Properties (**ScrollBar3DLightColor ('ScrollBar3DLightColor Property' in the on-line documentation)**, **ScrollBarArrowColor ('ScrollBarArrowColor Property' in the on-line documentation)**, **ScrollBarBaseColor ('ScrollBarBaseColor Property' in the on-line documentation)**, and so on)
- **GroupInfo ('GroupInfo Class' in the on-line documentation)** Background Property and Reset method

You can change the font size for the entire component using the underlying Web controls properties. For example, you can set the font size to 14 point using this command in code:

VB

```
FpSpread1.ActiveSheetView.DefaultStyle.Font.Size =
System.Web.UI.WebControls.FontUnit.Point(14)
```

You can set the default appearance to the version 3 appearance with the following code:

Using Code

Set the **BackgroundImageUrl ('BackgroundImageUrl Property' in the on-line documentation)**, **SelectionBackColor ('SelectionBackColor Property' in the on-line documentation)**, and default header and corner styles.

Example

This example shows how to set the control to the version 3 appearance.

C#

```
FpSpread1.CommandBar.Background.BackgroundImageUrl = NULL;
FpSpread1.Sheets[0].SelectionBackColor = Color.Empty;
FpSpread1.Sheets[0].ColumnHeader.DefaultStyleName = "HeaderDefault";
FpSpread1.Sheets[0].RowHeader.DefaultStyleName = "HeaderDefault";
FpSpread1.Sheets[0].SheetCorner.DefaultStyleName = "HeaderDefault";
```

VB

```
FpSpread1.CommandBar.Background.BackgroundImageUrl = Nothing
FpSpread1.Sheets(0).SelectionBackColor = Color.Empty
FpSpread1.Sheets(0).ColumnHeader.DefaultStyleName = "HeaderDefault"
FpSpread1.Sheets(0).RowHeader.DefaultStyleName = "HeaderDefault"
FpSpread1.Sheets(0).SheetCorner.DefaultStyleName = "HeaderDefault"
```

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

- ResetBackColor Method
- ResetBackground Method
- ResetBorder Method
- ResetCellType Method
- ResetFont Method
- ResetForeColor Method
- ResetHorizontalAlignment Method
- ResetLocked Method
- ResetTabStop Method
- ResetText Method
- ResetValue Method
- ResetVerticalAlignment 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.

- StyleInfo **ResetEditor ('ResetEditor Method' in the on-line documentation)** Method
- StyleInfo **ResetFormatter ('ResetFormatter Method' in the on-line documentation)** Method
- StyleInfo **ResetRenderer ('ResetRenderer Method' in the on-line documentation)** Method

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.

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).

Using the jQuery Theme Roller with Spread

You can apply a Theme Roller theme to the Spread control.

The theme is applied to the following areas.

- Column Header
- Row Header
- Corner
- Footer and Footer Corner
- Command Bar
- Title Bar
- Group Bar

- Gray Header
- Hierarchy Bar
- Pager
- Sheet Tab

The following styles are applied from the jQuery theme.

- Background-Color
- Background-Image
- ForeColor
- Font family
- Font weight
- Font size

If the jQuery theme is enabled, the jQuery theme style has a higher priority than the default style but a lower priority than the custom style. If the sheet skin is DefaultSkins.Default or it is not specified, the jQuery theme takes effect. If the sheet view active skin is a built-in skin (not DefaultSkins.Default) or a custom skin then the jQuery theme has no effect.

The viewport, context menu, filter menu, touch strip, dialogs, group headers, and group footers are not supported. If the SheetView.**SelectionBackColor ('SelectionBackColor Property' in the on-line documentation)** property is set, then the theme highlight is not applied.

For more information about themes, refer to the Theme Roller web site, http://jqueryui.com/themeroller/.

Using Code

- Add a reference to the theme.
- Set the **EnablejQueryTheme ('EnablejQueryTheme Property' in the on-line documentation)** property.

Example

This example displays a theme.

Script

```
<head runat="server">
  <title>Demo page</title>
  <link href="jquery-ui-themes-1.10.4/themes/ui-darkness/jquery-ui.min.css"
rel="stylesheet" />
  <link href="jquery-ui-themes-1.10.4/themes/ui-darkness/jquery.ui.theme.css"
rel="stylesheet" />
</head>
```

C#

```
FpSpread1.EnablejQueryTheme = true;
```

VB

```
FpSpread1.EnablejQueryTheme = True
```

Customizing the Appearance of the Sheet

You can set many different properties for the appearance of the data area of the spreadsheet.

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 entire sheet inside the component:

- **Working with the Active Sheet**
- **Working with Multiple Sheets**
- **Adding a Sheet**
- **Removing a Sheet**
- **Showing or Hiding a Sheet**
- **Setting the Background Color of the Sheet**
- **Adding a Title and Subtitle to a Sheet**
- **Customizing the Page Size (Rows to Display)**
- **Displaying Grid Lines on the Sheet**
- **Customizing the Sheet Corner**
- **Displaying a Footer for Columns or Groups**
- **Adding an Image to the Sheet (on-line documentation)**

You can quickly customize the appearance of a sheet by applying a "skin" to it. Skins are provided with Spread to create common formats. You can also create your own skin and save it, to use again, similar to a template.

> **Note:** Be aware that some settings for skins are affected by the setting of the **EnableClientScript ('EnableClientScript Property' in the on-line documentation)** property of the component.

The tasks you can perform when working with skins include:

- **Creating a Skin for Sheets**
- **Applying a Skin to a Sheet**

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 manipulate the model. Most developers who are not changing anything drastically find it easy to manipulate the short cut objects.

For more information on the sheet properties, refer to the **SheetView ('SheetView Class' in the on-line documentation)** class.

For more information on the **SheetView Collection** editor, refer to **SheetView Collection Editor**.

For information on displaying the sheet names, refer to **Displaying the Sheet Names**.

Working with the Active Sheet

The active sheet is the sheet that currently receives any user interaction. You can specify the active sheet programmatically. Use the **ActiveSheetView ('ActiveSheetView Property' in the on-line documentation)** property of the **FpSpread ('FpSpread Class' in the on-line documentation)** object.

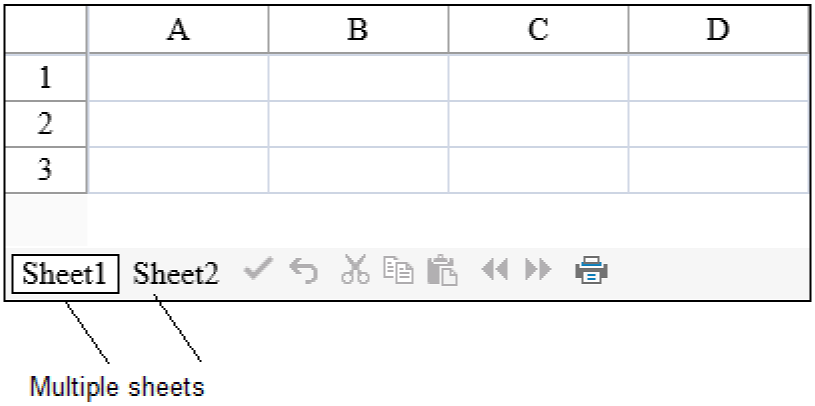
Usually, the active sheet is displayed on top of the other sheets in the component.

For information about adding a sheet, refer to **Adding a Sheet**.

For information on working with multiple sheets, refer to **Working with Multiple Sheets**.

Working with Multiple Sheets

The component allows multiple sheets. Set the **Count ('Count Property' in the on-line documentation)** property to specify the number of sheets. For information on adding sheets with the designer, see the **SheetView Collection Editor**.



For information about the display of the sheet names in the sheet name buttons, refer to **Displaying the Sheet Names**.

Formulas in a cell on one sheet can refer to a value or a cell on another sheet. For more information about formulas, refer to **Managing Formulas**.

You can name the sheets. 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.

For information about adding a sheet, refer to **Adding a Sheet**.

Using Code

This example sets the number of sheets.

C#

```
FpSpread1.Sheets.Count = 2;
```

VB

```
FpSpread1.Sheets.Count = 2
```

Adding a Sheet

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

In code, you can simply change the number of sheets by changing the **Count ('Count Property' in the on-line documentation)** property or you can explicitly add the sheet(s) by defining new sheets or by using the **Add ('Add Method' in the on-line documentation)** method. The following instructions describe how to add a sheet.

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

Using the 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. Click the **Add** button to add a sheet to the collection.
   A new sheet named Sheet*n* (where *n* is an integer) is added to the component.
4. 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.

5. Click **OK** to close the editor.

Using Code

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. Use the **Add ('Add Method' in the on-line documentation)** method with the **FpSpread ('FpSpread Class' in the on-line documentation)** component **Sheets ('Sheets Property' in the on-line documentation)** shortcut to add the new **SheetView ('SheetView Class' in the on-line documentation)** object to the collection of sheets (**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.Web.Spread.SheetView newsheet = new FarPoint.Web.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.Web.Spread.SheetView()
newsheet.SheetName = "North"
newsheet.ColumnCount = 10
newsheet.RowCount = 100

' Add the new sheet to the component.
FpSpread1.Sheets.Add(newsheet)
```

Using the Spread Designer

1. Select the **Data** menu. Click on the **Insert** icon.
2. Click the **Insert Sheet** option.
   A new sheet named Sheet*n* (where *n* is an integer) is added to the component.
3. If you want to change the name of the new sheet, click the new sheet in the Properties for Sheet*n* list, and change the **SheetName** property in the property list.
4. Click **Apply and Exit** to close the Spread Designer.

Removing a Sheet

You can remove a sheet or several sheets from the component. The sheet index is zero-based. In code, you can simply change the number of sheets using the **Count ('Count Property' in the on-line documentation)** property or you can explicitly remove the sheet(s) using the **Remove ('Remove Method' in the on-line documentation)** method.

Removing an existing sheet does not change the default sheet names provided to the other sheets. For example, a

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.

For information on adding a sheet, refer to **Adding a Sheet**.

Using the 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. In the **Members** list, select the sheet to remove.
4. Click the **Remove** button to remove the sheet from the collection.
5. Click **OK** to close the editor.

Using a Shortcut

Use the **Remove ('Remove Method' in the on-line documentation)** method with the **FpSpread ('FpSpread Class' in the on-line documentation)** component **Sheets ('Sheets Property' in the on-line documentation)** shortcut to remove the **SheetView ('SheetView Class' in the on-line documentation)** object from the collection of sheets (**SheetViewCollection ('SheetViewCollection Class' in the on-line documentation)**).

Example

This example code removes the second sheet from a 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))
```

Using the Spread Designer

1. Select the **Data** menu. Select the sheet you wish to delete (click on the sheet name at the bottom of the designer window).
2. Select the **Delete** icon.
3. Click **Delete Sheet**.
4. The Spread Designer asks you if you are sure you want to remove the sheet. Click **Yes** to remove the sheet.
5. Click **Apply and Exit** to close the Spread Designer.

Showing or Hiding a Sheet

You can hide a sheet so that it is not displayed to the user while still keeping it in the component.

Hiding a sheet does not change the default sheet names provided to the other sheets. For example, a 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**. Use the **Visible ('Visible Property' in**

the on-line documentation) property of the **FpSpread ('FpSpread Class' in the on-line documentation)** component to show or hide the sheet.

Using the 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. In the **Members** list, select the sheet to hide.
4. Select the **Visible** property in the property list and set to false.
5. Click **OK** to close the editor.

Using a Shortcut

Set the **Visible ('Visible Property' in the on-line documentation)** property of the **FpSpread ('FpSpread Class' in the on-line documentation)** component **Sheets ('Sheets Property' in the on-line documentation)** shortcut.

Example

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

C#

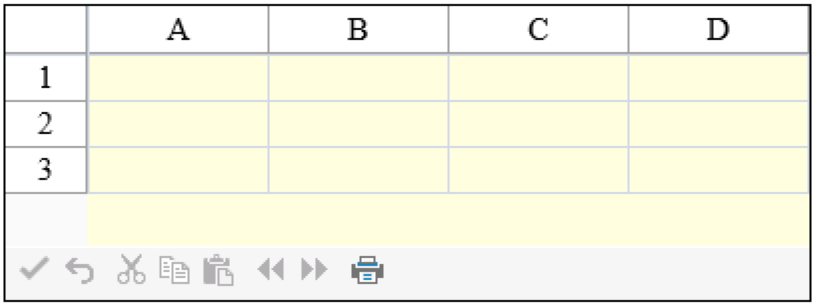
```
// Hide the second sheet.
FpSpread1.Sheets[1].Visible = false;
```

VB

```
' Hide the second sheet.
FpSpread1.Sheets[1].Visible = False
```

Setting the Background Color of the Sheet

You can customize the background color of the data area of the sheet. The sheet background color is displayed as the cell background color, unless you set specific cell colors (as explained in **Customizing the Colors of a Cell**). It is also the color in the rest of the sheet where cells are not displayed (empty area), as shown in the following figure, where the background color is set to light yellow.



	A	B	C	D
1				
2				
3				

The background color for the sheet can be set either with the **BackColor ('BackColor Property' in the on-line documentation)** property of the sheet (**SheetView ('SheetView Class' in the on-line documentation)** class) or the **BackColor ('BackColor Property' in the on-line documentation)** property of the **SheetSkin ('SheetSkin Class' in the on-line documentation)** class and the skin applied to the sheet.

Using the 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. In the **Members** list, select the sheet for which to set the background color.
4. Select the **BackColor** property in the property list, and then click the drop-down button to display the color picker.
5. Select a color in the color picker.
6. Click **OK** to close the editor.

Using a Shortcut

Set the **BackColor ('BackColor Property' in the on-line documentation)** property of the **FpSpread ('FpSpread Class' in the on-line documentation)** component **Sheets ('Sheets Property' in the on-line documentation)** shortcut.

Example

This example code sets the background color of the first sheet to light yellow.

C#

```
// Set the first sheet's background color to light yellow.
FpSpread1.Sheets[0].BackColor = Color.LightYellow;
```

VB

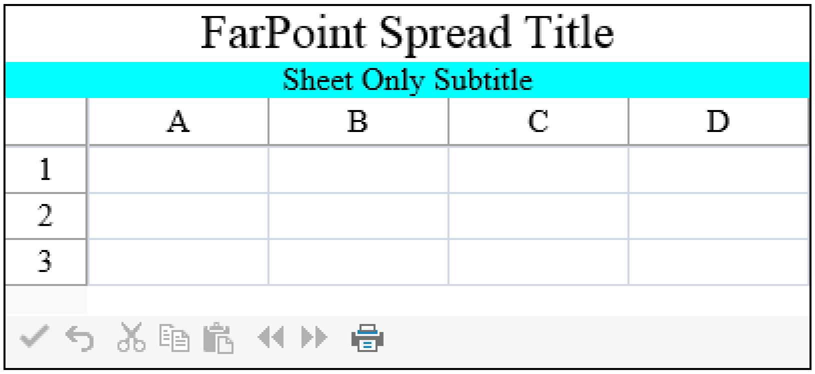
```
' Set the first sheet's background color to light yellow.
FpSpread1.Sheets(0).BackColor = Color.LightYellow
```

Using the Spread Designer

1. From the sheets displayed at the bottom, select the sheet for which you want to set the background color.
2. Select the **BackColor** property in the property list.
3. Click the drop-down arrow to display the color picker.
4. Select a color from the color picker.
5. Click **Apply and Exit** to close the Spread Designer.

Adding a Title and Subtitle to a Sheet

You can add a specially formatted area at the top of the spreadsheet that includes either a title or a subtitle or both. An example is shown here of a spreadsheet that has a title and a subtitle.



The title is set using the TitleInfo property at the FpSpread level. The subtitle is set using the **TitleInfo ('TitleInfo Property' in the on-line documentation)** property at the sheet level. So titles apply to the overall Spread component, while subtitles can be different for each sheet.

The API members that are involved include:

- **TitleInfo ('TitleInfo Class' in the on-line documentation)** Class and all members
- SheetView **TitleInfo ('TitleInfo Property' in the on-line documentation)** Property
- FpSpread **TitleInfo ('TitleInfo Property' in the on-line documentation)** Property

Using the Properties Window

1. At design time, in the **Properties** window, select the FpSpread component or the sheet.
2. Select **TitleInfo**.
3. Set the **Visible** property to true and other properties as needed.
4. Click **OK** to close the editor.

Using a Shortcut

Set the properties of the **TitleInfo ('TitleInfo Class' in the on-line documentation)** class at the FpSpread and Sheet level.

Example

This example code sets and displays a title for the component and subtitle for the sheet.

C#

```
// Show the title for the entire spreadsheet component.
FpSpread1.TitleInfo.Visible = true;
FpSpread1.TitleInfo.Text = "FarPoint Spread Title";
// 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 = HorizontalAlign.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"
' 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 = HorizontalAlign.Center
FpSpread1.Sheets(0).TitleInfo.BackColor = System.Drawing.Color.Aqua
```

Using the Spread Designer

1. Select the **Settings** menu.
2. Select the **Title** icon under the **Spread Settings** section.
3. Set the title and/or subtitle.
4. Click **Apply and Exit** to close the Spread Designer.

Customizing the Page Size (Rows to Display)

In this Web Form version of Spread, a page is the amount of data of a sheet that can be displayed at one time. When the sheet contains more rows than can be displayed in the component, Spread automatically creates pages that contain the other rows. (These are not to be confused with HTML pages.) For sheets that have more rows than fit in the display area, the sheet has multiple pages.

The page size is the number or rows that are displayed at one time. By default, the page size is ten, so ten rows are displayed. If you would like to display more than ten rows (or ten records for a bound spreadsheet), set the PageSize property to the number of records you want to display on every page.

For more information on setting the page navigation, refer to **Customizing Page Navigation**.

Using Shortcut Object

Set the **PageSize ('PageSize Property' in the on-line documentation)** property for the sheet using the **ActiveSheetView ('ActiveSheetView Property' in the on-line documentation)** shortcut of the FpSpread component.

Example

The following code shows how to set the page size to display 15 rows.

C#

```
FpSpread1.ActiveSheetView.PageSize = 15;
```

VB

```
FpSpread1.ActiveSheetView.PageSize = 15
```

Using Code

Set the **PageSize ('PageSize Property' in the on-line documentation)** property for the **SheetView ('SheetView Class' in the on-line documentation)** class.

Example

The following code shows how to set the page size to display 15 rows.

C#

```
FarPoint.Web.Spread.SheetView sv = FpSpread1.ActiveSheetView;
sv.PageSize = 15;
```

VB

```
Dim sv As FarPoint.Web.Spread.SheetView
sv = FpSpread1.ActiveSheetView
sv.PageSize = 15
```

Using the Spread Designer

1. Select the **Settings** tab.
2. Select the **General** icon under the **Sheet Settings** section.
3. Check the **AllowPage** check box and specify the number of rows in the **Page Size** edit box.

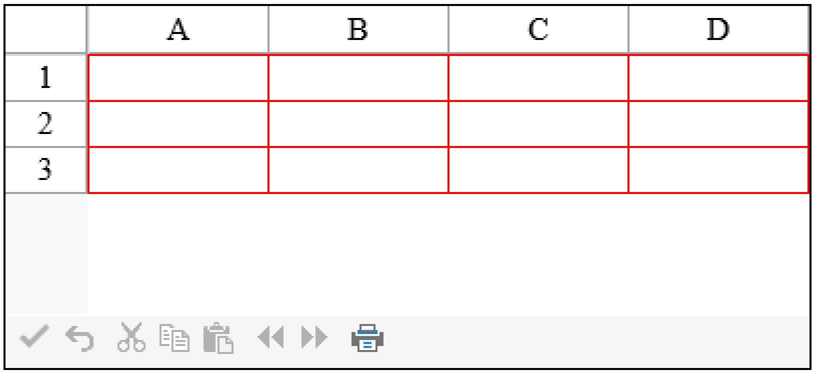
4. Click **Apply and Exit** to close the Spread Designer.

Displaying Grid Lines on the Sheet

You can display grid lines on the sheet that distinguish rows or columns (or both). You can set the grid lines to display using the **GridLines ('GridLines Property' in the on-line documentation)** property for the sheet (which uses the settings of the **GridLines** enumeration in the underlying Microsoft .NET framework).

- To show only the grid lines for the rows (horizontal lines), set the property to GridLines.Horizontal.
- To show only the grid lines for the columns (vertical lines), set the property to GridLines.Vertical.
- To show the grid lines for both the rows and the columns (both the vertical and horizontal lines), set the property to GridLines.Both.
- To turn off (or hide) the grid lines, set the property to GridLines.None.

You can set the color of the grid lines. In the following figure, where both the horizontal and vertical lines are displayed, the grid lines are red.



	A	B	C	D
1				
2				
3				

You can also set the grid lines and the grid lines color by defining these properties in the **SheetSkin ('SheetSkin Class' in the on-line documentation)** class and then applying the skin to the sheet. For more information creating and applying skins, refer to **Applying a Skin to a Sheet** and **Creating a Skin for Sheets**.

To set borders around individual cells, refer to **Customizing Cell Borders**.

Using the 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. In the **Members** list, select the sheet for which you want to set the grid line color.
4. Select the **GridLineColor** property in the property list, and then click the drop-down button to display the color picker.
5. Select a color in the color picker.
6. Click **OK** to close the editor.

Using a Shortcut

Set the **GridLines ('GridLines Property' in the on-line documentation)** and **GridLineColor ('GridLineColor Property' in the on-line documentation)** properties of the **FpSpread ('FpSpread Class' in the on-line documentation)** component **Sheets ('Sheets Property' in the on-line documentation)** shortcut.

Example

This example code sets the grid line color to red and displays the row grid lines.

C#

```
// Set the grid line color to red.
FpSpread1.Sheets[0].GridLineColor = System.Drawing.Color.Red;
FpSpread1.Sheets[0].GridLines = GridLines.Horizontal;
```

VB

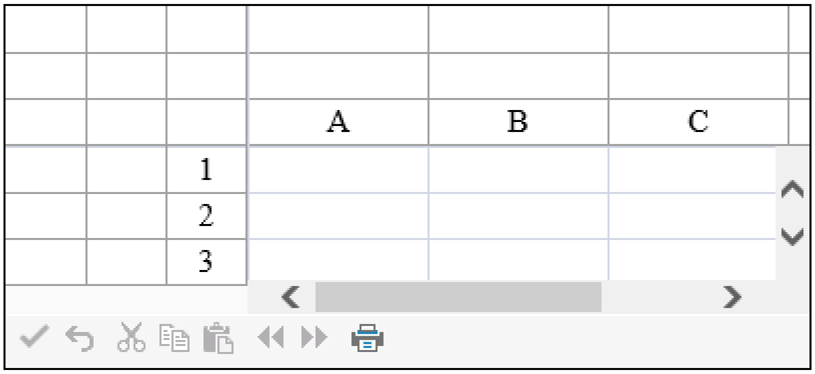
```
' Set the grid line color to red.
FpSpread1.Sheets(0).GridLineColor = System.Drawing.Color.Red
FpSpread1.Sheets(0).GridLines = GridLines.Horizontal
```

Using the Spread Designer

1. Select the **Settings** menu.
2. From the sheets displayed at the bottom, select the sheet for which you want to set the grid line color.
3. Select the **GridLines** icon in the **Sheet Setting** section.
4. Set the color or any other grid properties.
5. Click **OK** to close the dialog.
6. Click **Apply and Exit** to close the Spread Designer.

Customizing the Sheet Corner

You can customize the appearance of the sheet corner, the header cell in the upper left corner of the sheet, for each sheet. Sheet corners can display grid lines, have a different background color from the rest of the headers, and more. You can set the style of the sheet corner. You can set the style of the sheet corner as you would any cell in the spreadsheet and you can set the text that appears in the corner. Any of the properties of the **StyleInfo ('StyleInfo Class' in the on-line documentation)** object can be set for the cells in the corner of the sheet. In the following figure, the sheet corner uses default values (the sheet corner row and column count have been set to three).



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.

The parts of the API that affect the sheet corner include:

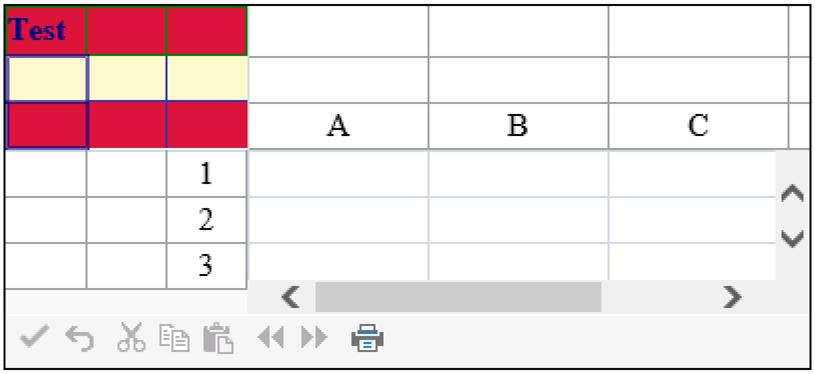
- FpSpread **SheetCorner ('SheetCorner Property' in the on-line documentation)** Property
- SheetView **AllowTableCorner ('AllowTableCorner Property' in the on-line documentation)** Property
- SheetView **SheetCorner ('SheetCorner Property' in the on-line documentation)** Property
- SheetView **SheetCornerStyle ('SheetCornerStyle Property' in the on-line documentation)** Property

- SheetView **SheetCornerStyleName ('SheetCornerStyleName Property' in the on-line documentation)** Property
- **SheetCorner ('SheetCorner Class' in the on-line documentation)** Class - all members

Several of the **StyleInfo** object properties 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 (see
- 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 figure below shows an example (see the example code below) that specifies a sheet corner with alternating row colors and a column border.



Using Shortcut Object

1. Set the given property of the sheet corner style (**SheetCornerStyle ('SheetCornerStyle Property' in the on-line documentation)** property) of the **FpSpread ('FpSpread Class' in the on-line documentation)** component **Sheets ('Sheets Property' in the on-line documentation)** shortcut.

Example

This example code sets the text, border colors, text colors, and row colors.

C#

```
FarPoint.Web.Spread.StyleInfo altrowstyle = new FarPoint.Web.Spread.StyleInfo();
altrowstyle.BackColor = System.Drawing.Color.LemonChiffon;
altrowstyle.ForeColor = System.Drawing.Color.Navy;
altrowstyle.Font.Bold = true;
FpSpread1.Sheets[0].AllowTableCorner = true;
FpSpread1.Sheets[0].SheetCorner.RowCount = 3;
FpSpread1.Sheets[0].SheetCorner.ColumnCount = 3;
FpSpread1.Sheets[0].SheetCorner.AlternatingRows[0].BackColor =
System.Drawing.Color.Crimson;
FpSpread1.Sheets[0].SheetCorner.Cells[0, 0].Text = "Test";
FpSpread1.Sheets[0].SheetCorner.Columns[0].Border = new
FarPoint.Web.Spread.Border(System.Web.UI.WebControls.BorderStyle.Double,
System.Drawing.Color.DarkBlue, 2);
FpSpread1.Sheets[0].SheetCorner.Rows[0].Border = new
FarPoint.Web.Spread.Border(System.Drawing.Color.Green);
FpSpread1.Sheets[0].SheetCornerStyle = new FarPoint.Web.Spread.StyleInfo(altrowstyle);
```

VB

```
Dim altrowstyle As New FarPoint.Web.Spread.StyleInfo()
altrowstyle.BackColor = Drawing.Color.LemonChiffon
altrowstyle.ForeColor = Drawing.Color.Navy
altrowstyle.Font.Bold = True
FpSpread1.Sheets(0).AllowTableCorner = True
FpSpread1.Sheets(0).SheetCorner.RowCount = 3
FpSpread1.Sheets(0).SheetCorner.ColumnCount = 3
FpSpread1.Sheets(0).SheetCorner.AlternatingRows(0).BackColor = Drawing.Color.Crimson
FpSpread1.Sheets(0).SheetCorner.Cells(0, 0).Text = "Test"
FpSpread1.Sheets(0).SheetCorner.Columns(0).Border = New
FarPoint.Web.Spread.Border(System.Web.UI.WebControls.BorderStyle.Double,
Drawing.Color.DarkBlue, 2)
FpSpread1.Sheets(0).SheetCorner.Rows(0).Border = New
FarPoint.Web.Spread.Border(Drawing.Color.Green)
FpSpread1.Sheets(0).SheetCornerStyle = New FarPoint.Web.Spread.StyleInfo(altrowstyle)
```

Using the Spread Designer

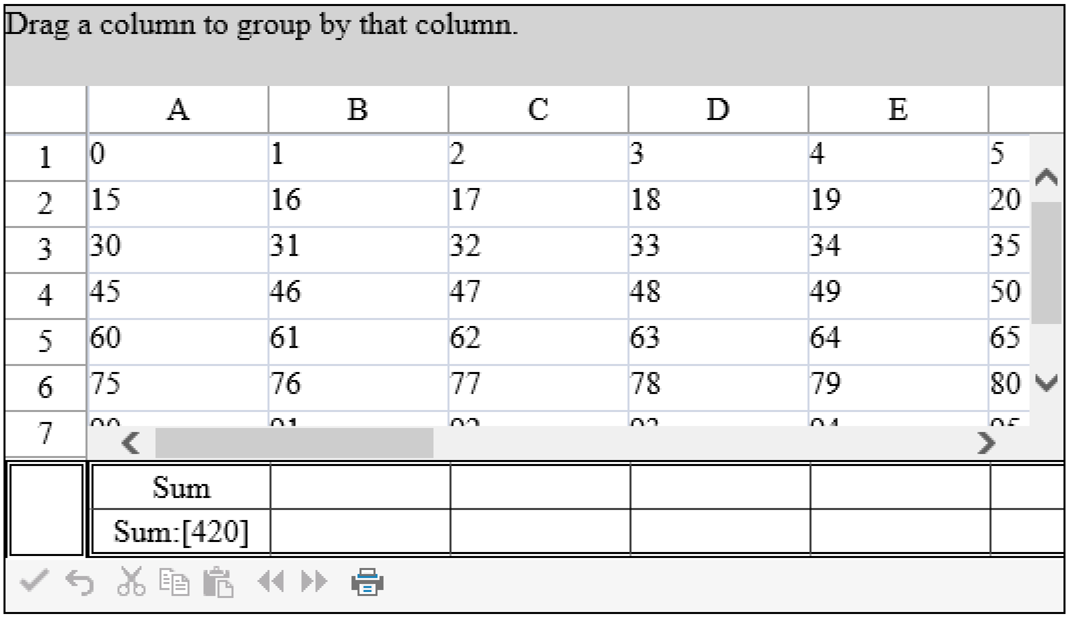
1. Select the sheet tab for the sheet for which you want to display the sheet corner.
2. Select the **Settings** menu.
3. Select the **Header Editor** icon in the **Other Settings** section.
4. Select **Sheet Corner** in the **Selected Header** drop-down box.
5. Set the various formatting properties in the property grid.
6. 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 or 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 under each group when 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.

In order to calculate the column footer or group footer result with a formula, set the **AggregationType ('AggregationType Property' in the on-line documentation)** property of the **Column ('Column Class' in the on-line documentation)** object to the correct formula type for that column. The **Aggregate ('Aggregate Event' in the on-line documentation)** event is raised after setting this property and can be used to add information to column and group footers. The following figure displays a column footer with a formula in the first column:

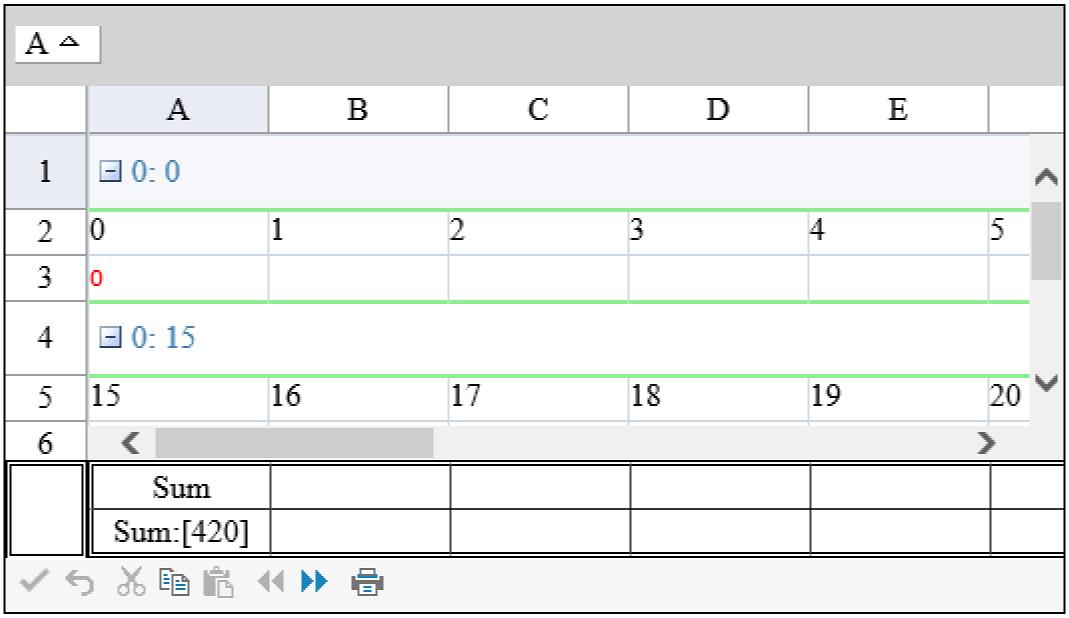


Drag a column to group by that column.

	A	B	C	D	E	
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
6	75	76	77	78	79	80
7						

	Sum					
	Sum:[420]					

The group footer is an extra row that is displayed below the group after grouping by a column header. The following figure shows the result of a sum formula in column A for each row below a group. The rows are grouped by the data in column A.



	A	B	C	D	E	
1	⊟ 0: 0					
2	0	1	2	3	4	5
3	0					
4	⊟ 0: 15					
5	15	16	17	18	19	20
6						

	Sum					
	Sum:[420]					

The **Grouped ('Grouped Event' in the on-line documentation)** or **Grouping ('Grouping Event' in the on-line documentation)** events can be used to set style information in the group footer after a user has created the group.

For more information on column appearance, refer to **Customizing the Appearance of Rows and Columns**.

For more information on grouping, refer to **Customizing Grouping of Rows of User Data**.

Using the 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. Select **DefaultStyle** under **ColumnFooter** in order to set additional column footer properties such as color.
5. Select **GroupInfoFooter** in order to set additional group footer properties such as color.
6. 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 with a span, puts text in a cell, and sets the text color.

C#

```
FpSpread1.Sheets[0].RowCount = 10;
FpSpread1.Sheets[0].ColumnCount = 15;
// Show the column footer.
FpSpread1.ActiveSheetView.ColumnFooter.Visible = true;
FpSpread1.ActiveSheetView.ColumnFooter.RowCount = 2;
FpSpread1.ActiveSheetView.ColumnFooter.DefaultStyle.ForeColor = Color.Purple;
FpSpread1.ActiveSheetView.ColumnFooter.DefaultStyle.Border.BorderStyle =
BorderStyle.Double;
FpSpread1.ActiveSheetView.ColumnFooter.Columns[12].HorizontalAlign =
HorizontalAlign.Left;
FpSpread1.ActiveSheetView.ColumnFooter.Cells[0, 12].RowSpan = 2;
FpSpread1.ActiveSheetView.ColumnFooter.Cells[0, 0].Value = "test";
```

VB

```
FpSpread1.Sheets(0).RowCount = 10
FpSpread1.Sheets(0).ColumnCount = 15
' Show the footer.
FpSpread1.ActiveSheetView.ColumnFooter.Visible = true
FpSpread1.ActiveSheetView.ColumnFooter.RowCount = 2
FpSpread1.ActiveSheetView.ColumnFooter.DefaultStyle.ForeColor = Color.Purple
FpSpread1.ActiveSheetView.ColumnFooter.DefaultStyle.Border.BorderStyle =
BorderStyle.Double
FpSpread1.ActiveSheetView.ColumnFooter.Columns(12).HorizontalAlign =
HorizontalAlign.Left
FpSpread1.ActiveSheetView.ColumnFooter.Cells(0, 12).RowSpan = 2
FpSpread1.ActiveSheetView.ColumnFooter.Cells(0, 0).Value = "test
```

Using a Shortcut

Set the **AggregationType ('AggregationType Property' in the on-line documentation)** property for the column.

Example

This example sums the values in the first column and displays them in the column and group footers.

C#

```
FpSpread1.Sheets[0].RowCount=8;
FpSpread1.Sheets[0].ColumnCount = 15;
this.FpSpread1.ActiveSheetView.GroupBarVisible = true;
this.FpSpread1.ActiveSheetView.AllowGroup = true;
this.FpSpread1.ActiveSheetView.GroupFooterVisible = true;
this.FpSpread1.ActiveSheetView.ColumnFooter.Visible = true;
this.FpSpread1.ActiveSheetView.ColumnFooter.RowCount = 2;
this.FpSpread1.ActiveSheetView.ColumnFooter.DefaultStyle.Border.BorderStyle =
BorderStyle.Double;
this.FpSpread1.ActiveSheetView.ColumnFooter.Columns[12].HorizontalAlign =
HorizontalAlign.Left;
this.FpSpread1.ActiveSheetView.ColumnFooter.Cells[0, 12].RowSpan = 2;
//Value
for (int r = 0; r < this.FpSpread1.ActiveSheetView.RowCount; r++)
{
for (int j = 0; j < this.FpSpread1.ActiveSheetView.ColumnCount; j++) {
FpSpread1.ActiveSheetView.DataModel.SetValue(r, j, j + r *
FpSpread1.ActiveSheetView.ColumnCount);
}
}
int i = 0;
this.FpSpread1.ActiveSheetView.Columns[i].AggregationType =
FarPoint.Web.Spread.Model.AggregationType.Sum;
this.FpSpread1.ActiveSheetView.ColumnFooter.Cells[0, i].Value = "Sum";
this.FpSpread1.ActiveSheetView.ColumnFooter.Cells[1, i].Value = "Sum:[{0}]";
//Use the Grouped event to set style information
protected void FpSpread1_Grouped(object sender, EventArgs e)
{
FarPoint.Web.Spread.Model.GroupFooter gf =
default(FarPoint.Web.Spread.Model.GroupFooter);
FarPoint.Web.Spread.GroupInfo gi = default(FarPoint.Web.Spread.GroupInfo);
gf = ((FarPoint.Web.Spread.Model.GroupDataModel
)FpSpread1.ActiveSheetView.DataModel).GetGroupFooter(2);
gi = FpSpread1.ActiveSheetView.GetGroupFooterInfo(gf);
gi.Font.Name = "Verdana";
gi.Font.Size = 8;
gi.ForeColor = System.Drawing.Color.Red;
}
```

VB

```
FpSpread1.Sheets(0).RowCount = 8
FpSpread1.Sheets(0).ColumnCount = 15
FpSpread1.ActiveSheetView.GroupBarVisible = True
FpSpread1.ActiveSheetView.AllowGroup = True
FpSpread1.ActiveSheetView.GroupFooterVisible = True
FpSpread1.ActiveSheetView.ColumnFooter.Visible = True
FpSpread1.ActiveSheetView.ColumnFooter.RowCount = 2
FpSpread1.ActiveSheetView.ColumnFooter.DefaultStyle.Border.BorderStyle =
BorderStyle.Double
'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.ActiveSheetView.DataModel.SetValue(r, j, j + r *
FpSpread1.ActiveSheetView.ColumnCount)
Next j
Next r
Dim i As Integer
```

```
i = 0
FpSpread1.ActiveSheetView.Columns(0).AggregationType =
FarPoint.Web.Spread.Model.AggregationType.Sum
FpSpread1.ActiveSheetView.ColumnFooter.Cells(0, i).Value = "Sum"
FpSpread1.ActiveSheetView.ColumnFooter.Cells(1, i).Value = "Sum:[{0}]"
'Use the Grouped event to set style information
Protected Sub FpSpread1_Grouped(ByVal sender As Object, ByVal e As System.EventArgs)
Handles FpSpread1.Grouped
Dim gf As FarPoint.Web.Spread.Model.GroupFooter
Dim gi As FarPoint.Web.Spread.GroupInfo
gf = CType(FpSpread1.ActiveSheetView.DataModel,
FarPoint.Web.Spread.Model.GroupDataModel).GetGroupFooter(2)
gi = FpSpread1.ActiveSheetView.GetGroupFooterInfo(gf)
gi.Font.Name = "Verdana"
gi.Font.Size = 8
gi.ForeColor = System.Drawing.Color.Red
End Sub
```

Using the Spread Designer

1. Select the sheet tab for the sheet for which you want to display the column footer.
2. Select the **Settings** menu.
3. Select the **Header Editor** icon in the **Other Settings** section (**Group Footer Editor** icon for group footers).
4. Select **Column Footer** in the **Selected Header** drop-down box.
5. Set the various formatting properties in the property grid.
6. Set the **Visible** property for the column or group footer under **Sheet Settings**, **Headers**, and the **Column** or **Group Footer** tab.
7. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Creating a Skin for Sheets

You can quickly customize the appearance of a sheet by applying a "skin" to it. Some built-in (default) skins are provided with Spread to create common formats. You can create your own custom skin and save it to use again or share it, similar to a template. A skin, whether built-in or custom, can be applied to any number of sheets in a Spread component. Just as a style can be applied to cells, so a skin can be applied to an entire sheet. Because a skin can be saved to a file, you can use it across projects and share it with other developers.

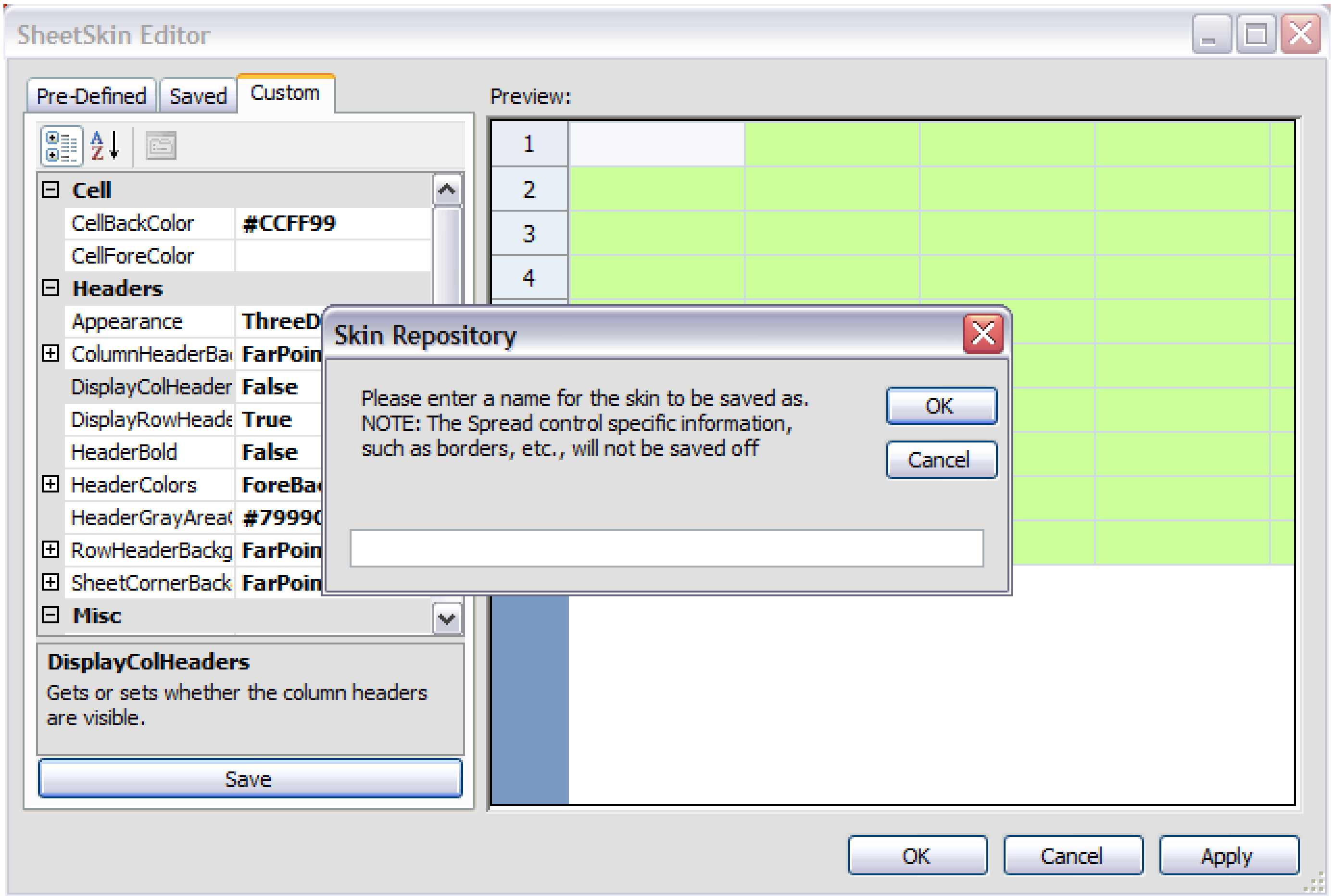
A skin includes the following appearance settings:

- cell colors
- header colors
- headers displayed or not
- header text bold
- row colors
- selection colors
- Spread background color
- grid lines
- cell spacing

You can save several appearance properties of a sheet as a custom skin, which can be used in other projects or shared with other developers. Any custom skins that you create can be saved to a file in the Spread Designer using the **Sheet Skin** editor.

Use the **Custom** tab to create your own skin. From here you can create a custom skin by altering the properties in the

Property window. When you are ready, click **Save**. The **Skin Repository** dialog is displayed and you can enter a name for your skin.



Cell	
CellBackColor	**#CCFF99**
CellForeColor	
Headers	
Appearance	**ThreeD**
ColumnHeaderBa	**FarPoin**
DisplayColHeader	**False**
DisplayRowHeade	**True**
HeaderBold	**False**
HeaderColors	**ForeBa**
HeaderGrayArea(	**#79990**
RowHeaderBackg	**FarPoin**
SheetCornerBack	**FarPoin**
Misc	

DisplayColHeaders
Gets or sets whether the column headers are visible.

Please enter a name for the skin to be saved as.
NOTE: The Spread control specific information,
such as borders, etc., will not be saved off

After you select **OK**, the file name is displayed in the list of saved custom skins in the **Saved** tab. You can use the custom skin again later by selecting the **Saved** tab and then applying the appropriate custom skin.

For more information on SheetSkin object, refer to **SheetSkin ('SheetSkin Class' in the on-line documentation)** in the API reference documentation.

For information on the **SheetSkin** editor in the Spread Designer, refer to **SheetSkin Editor**.

For instructions for applying the built-in sheet skins, see **Applying a Skin to a Sheet**.

For instructions on creating and applying your own cell-level styles, see **Creating and Applying a Custom Style for Cells**.

Using Code

Use the **SheetSkin ('SheetSkin Class' in the on-line documentation)** object constructor, and set its parameters to specify the settings for the skin.

Example

This example code sets the sheet to use a custom skin.

C#

```
FarPoint.Web.Spread.SheetSkin myskin = new FarPoint.Web.Spread.SheetSkin("MySkin",
Color.BlanchedAlmond, Color.Bisque, Color.Navy, 2, Color.Blue, GridLines.Both,
Color.Beige, Color.BurlyWood, Color.AntiqueWhite, Color.Brown, Color.Bisque,
Color.Bisque, true, true, true, true, false);
myskin.Apply(FpSpread1.Sheets[0]);
```

VB

```
Dim myskin As New FarPoint.Web.Spread.SheetSkin("MySkin", Color.BlanchedAlmond,
Color.Bisque, Color.Navy, 2, Color.Blue, GridLines.Both, Color.Beige, Color.BurlyWood,
Color.AntiqueWhite, Color.Brown, Color.Bisque, Color.Bisque, True, True, True, True,
False)
myskin.Apply(FpSpread1.Sheets(0))
```

Using the Spread Designer

1. Select the **Settings** menu.
2. Select the **SheetSkin** icon.
3. Select the **Custom** tab to create a custom skin.
4. Set the properties.
5. Click **Save** and type in a name for the custom skin.
6. Click **OK**.
7. Click **Apply and Exit** to close the Spread Designer.

Applying a Skin to a Sheet

You can quickly customize the appearance of a sheet by applying a "skin" to it. Built-in (default) skins are provided with Spread to create common formats. You can also create your own custom skin and save it to use again, similar to a template.

For an overview and illustrations of sheet skins, see **Creating a Skin for Sheets**.

Using Code

1. If you want to create your own sheet skin, follow the instructions provided in **Creating a Skin for Sheets** to create a sheet skin and apply it. To apply a default sheet skin, follow the rest of these directions.
2. Use the **GetAt ('GetAt Method' in the on-line documentation)** method of the **DefaultSkins ('DefaultSkins Class' in the on-line documentation)** object to specify the index of the default skin to return, then the default skin **Apply ('Apply Method' in the on-line documentation)** method to assign it to a specific FpSpread component, collection of sheets, or sheet.

Example

This example code sets the first sheet to use the Colorful2 predefined skin.

C#

```
FarPoint.Web.Spread.DefaultSkins.Colorful2.Apply(FpSpread1.Sheets[0]);
```

VB

```
FarPoint.Web.Spread.DefaultSkins.Colorful2.Apply(FpSpread1.Sheets(0))
```

Using the Spread Designer

1. Select the **Settings** menu.
2. Select the **SheetSkin** icon.
3. In the **Pre-Defined** tab or the **Custom** tab, choose the skin to use for the sheet.
4. Click **OK**.
5. Click **Apply and Exit** to close the Spread Designer.

Customizing the Appearance of Rows and Columns

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

- **Customizing the Number of Rows or Columns**
- **Adding a Row or Column**
- **Removing a Row or Column**
- **Showing or Hiding Rows or Columns**
- **Setting the Row Height or Column Width**
- **Setting the Top Row to Display**
- **Creating Alternating Rows**
- **Creating Row Templates (Multiple-Line Columns)**

When you work with rows and columns, you can manipulate 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)**, etc.) or you can directly manipulate the model. Most developers who are not changing anything drastically find it easy to manipulate the shortcut objects.

Remember that 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 information on headers, refer to **Customizing the Appearance of Headers**.

For more information, refer to the **Row ('Row Class' in the on-line documentation)** class or **Column ('Column Class' in the on-line documentation)** class.

For more information about the underlying axis model, refer to **Understanding the Axis Model**.

Customizing the Number of Rows or Columns

When you create a sheet, it is automatically created with three rows and four columns. You can change the number to up to two billion rows and columns.

Using the Properties Window

1. At design time, in the **Properties** window, select the FpSpread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. Set **RowCount** and **ColumnCount** under the **Layout** section.
5. Click **OK** to close the editor.

Using a Shortcut

Set the **RowCount ('RowCount Property' in the on-line documentation)** or **ColumnCount ('ColumnCount Property' in the on-line documentation)** property for the **Sheets ('Sheets Property' in the on-line documentation)** shortcut.

Example

This example code sets the first sheet to have 10 columns and 100 rows.

C#

```
FpSpread1.Sheets[0].RowCount = 100;
FpSpread1.Sheets[0].ColumnCount = 10;
```

VB

```
FpSpread1.Sheets(0).RowCount = 100
FpSpread1.Sheets(0).ColumnCount = 10
```

Using Code

Set the **RowCount ('RowCount Property' in the on-line documentation)** or **ColumnCount ('ColumnCount Property' in the on-line documentation)** property for a **SheetView ('SheetView Class' in the on-line documentation)** class.

Example

This example code sets the first sheet to have 100 rows and 10 columns.

C#

```
FarPoint.Web.Spread.SheetView Sheet0;
Sheet0 = FpSpread1.Sheets[0];
Sheet0.RowCount = 100;
Sheet0.ColumnCount = 10;
```

VB

```
Dim Sheet0 As FarPoint.Web.Spread.SheetView
Sheet0 = FpSpread1.Sheets(0)
Sheet0.RowCount = 100
Sheet0.ColumnCount = 10
```

Using the Spread Designer

1. Select the **Settings** menu.
2. At the bottom, select the sheet for which you want to set the number of rows or columns.
3. Select the **General** icon under the **Sheet Settings** section and change the **RowCount** or **ColumnCount** setting.
4. Click **OK**.
5. Click **Apply and Exit** to close the Spread Designer.

Adding a Row or Column

You can add one or more rows or columns to a sheet, and specify where the row or column 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.

For adding an unbound row to a sheet in a component that is bound to a data source, refer to **Adding an Unbound Row**.

Using a Shortcut

1. Use the **AddRows ('AddRows Method' in the on-line documentation)** method or **AddColumns ('AddColumns Method' in the on-line documentation)** method for the **Sheets ('Sheets Property' in the on-line documentation)** shortcut.
2. Set the *row* or *column* parameter to specify the row or column before which to add the rows or columns.
3. Set the *count* parameter to specify the number of rows or columns 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)** method or **AddColumns ('AddColumns 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 add the rows or columns.
3. Set the *count* parameter to specify the number of rows or columns to add.

Example

This example code adds two columns before column 6.

C#

```
FarPoint.Web.Spread.SheetView Sheet0;
Sheet0 = FpSpread1.Sheets[0];
Sheet0.AddColumns(6,2);
```

VB

```
Dim Sheet0 As FarPoint.Web.Spread.SheetView
Sheet0 = FpSpread1.Sheets(0)
Sheet0.AddColumns(6, 2)
```

Using the Spread Designer

1. Select the **Data** menu.
2. At the bottom, select the sheet for which you want to add rows or columns.
3. Select an existing row or column (the new row or column is inserted before this row or column).
4. Click the **Insert** icon and then select **Insert Row** or **Insert Column**.
5. Click **OK**.
6. Click **Apply and Exit** to close the Spread Designer.

Removing a Row or Column

You can remove one or more rows or columns from a sheet and you can allow the end user to remove rows or prohibit them from removing rows.

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 Rows or Columns**.

Using a Shortcut

1. Use the **RemoveRows ('RemoveRows Method' in the on-line documentation)** or **RemoveColumns ('RemoveColumns Method' in the on-line documentation)** method for the **Sheets ('Sheets Property' in the on-line documentation)** shortcut.
2. Set the *row* or *column* parameter to specify the first row or column to remove.
3. Set the *count* parameter to specify the number of rows or columns to remove.

Example

This example code removes two rows.

C#

```
FpSpread1.Sheets[0].RemoveRows(6,2);
```

VB

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

Using Code

1. Use 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 first row or column to remove.
3. Set the *count* parameter to specify the number of rows or columns to remove.

Example

This example code removes two rows.

C#

```
FarPoint.Web.Spread.SheetView Sheet0;
Sheet0 = FpSpread1.Sheets[0];
Sheet0.RemoveRows(6,2);
```

VB

```
Dim Sheet0 As FarPoint.Web.Spread.SheetView
Sheet0 = FpSpread1.Sheets(0)
Sheet0.RemoveRows(6, 2)
```

Using the Spread Designer

1. Select the **Data** menu.
2. At the bottom, select the sheet for which you want to remove rows or columns.
3. In the **Rows or Columns** area, select an existing row or column in the list.

4. Click the **Delete** icon. Select **Delete Row** or **Delete Column** to remove a row or column.
5. Click **OK**.
6. Click **Apply and Exit** to close the Spread Designer.

Showing or Hiding Rows or Columns

You can hide a row or column so that it is not visible to the user. You can also hide only the row headers or column headers; follow the procedures in **Showing or Hiding Headers**.

When you hide a row or column, the value of the row height or column width is kept by the fpSpread component. If you display the row or column again, it is displayed at the value it was before it was hidden. The data is still available to other parts of the sheet; the only change is that the row or column is not displayed.

If you want to remove the row or column, refer to **Removing a Row or Column**.

Using a Shortcut

Set the **Visible ('Visible Property' in the on-line documentation)** property for the **Row ('Row Class' in the on-line documentation)** shortcut object or the **Visible ('Visible Property' in the on-line documentation)** property for the **Column ('Column Class' in the on-line documentation)** shortcut object, or use the **SetRowVisible ('SetRowVisible Method' in the on-line documentation)** or **SetColumnVisible ('SetColumnVisible Method' in the on-line documentation)** method in the **SheetView ('SheetView Class' in the on-line documentation)** object.

Example

This example code hides the second row and hides the third column.

C#

```
FpSpread1.Sheets[0].SetRowVisible(1, false);
FpSpread1.Sheets[0].SetColumnVisible(2, false);
```

VB

```
FpSpread1.Sheets[0].SetRowVisible(1, false)
FpSpread1.Sheets[0].SetColumnVisible(2, false)
```

Using Code

Set the **SetRowVisible ('SetRowVisible Method' in the on-line documentation)** or **SetColumnVisible ('SetColumnVisible Method' in the on-line documentation)** method in the **SheetView ('SheetView Class' in the on-line documentation)** object.

Example

This example code sets the first sheet to have 100 rows and 10 columns.

C#

```
FarPoint.Web.Spread.SheetView sv;
sv = FpSpread1.ActiveSheetView;
sv.SetRowVisible(1, false);
sv.SetColumnVisible(2, false);
```

VB

```
Dim sv As FarPoint.Web.Spread.SheetView
sv = FpSpread1.ActiveSheetView
sv.SetRowVisible(1, False)
sv.SetColumnVisible(2, false)
```

Using the Spread Designer

1. In Spread Designer, select the row or column.
2. In the properties window, set the **Visible** property to false.
3. Click **Apply and Exit** to close the Spread Designer.

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.

You can set the column width with the **Width ('Width Property' in the on-line documentation)** property of the **FpSpread ('FpSpread Class' in the on-line documentation)** component **Columns ('Columns Property' in the on-line documentation)** object. You can set the row height with the **Height ('Height Property' in the on-line documentation)** property of the **FpSpread ('FpSpread Class' in the on-line documentation)** component **Rows ('Rows Property' in the on-line documentation)** object.

You can also use the **DefaultColumnWidth ('DefaultColumnWidth Property' in the on-line documentation)** property for the sheet to set the width for all columns in the sheet; use the **SetColumnWidth ('SetColumnWidth Method' in the on-line documentation)** method or use the Column **Width ('Width Property' in the on-line documentation)** property to set the width for a specific column. For row heights, set the **DefaultRowHeight ('DefaultRowHeight Property' in the on-line documentation)** property for the sheet to set the height for all the rows in the sheet; use the **SetRowHeight ('SetRowHeight Method' in the on-line documentation)** method or use the Row **Height ('Height Property' in the on-line documentation)** property to set the height for a specific row.

Users can change the row height or column width by dragging the header lines between rows or columns.

Using the Properties Window

1. To change the default column width setting or row height, 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. In the **Members** list, select the sheet for which to set the default column width or row height.
4. Select the **DefaultColumnWidth** property in the property list or the **DefaultRowHeight**, and specify the width or height in pixels.
5. Click **OK** to close the editor.
6. To change the width or height for a specific column or row, select the **Columns** or **Rows** collection after selecting the sheet. Then select a column or row and set the width or height properties.
7. Click **OK** to close the editor.

Using Code

Typically, set the default column width for the sheet and set the width of individual columns on that sheet as needed. Similarly, set the default row height for the sheet and set the height of individual rows on that sheet as needed.

Example

This example code changes the default width of all columns for the first sheet to 50 pixels, but makes the width of the second column 100 pixels.

C#

```
// Set default width to 50, but second column 100.
FpSpread1.Sheets[0].DefaultColumnWidth = 50;
FpSpread1.Sheets[0].Columns[1].Width = 100;
```

VB

```
' Set default width to 50, but second column 100.
FpSpread1.Sheets(0).DefaultColumnWidth = 50
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 width of the second column to 100 pixels.

C#

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

VB

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

Using the Spread Designer

1. To set the default column width or row height,
    a. Select the sheet for which you want to set the default column width or row height.
    b. Select **DefaultColumnWidth** or **DefaultRowHeight** and set the width or height properties.
    c. 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 or row height,
    a. Select the column or row for which you want to change the width or height.
    b. In the properties list for that column or row, change the **Width** property or the **Height** property.
    c. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Setting the Top Row to Display

You can set the display of a sheet to display a particular row as the top row with the SheetView.**TopRow ('TopRow Property' in the on-line documentation)** property. Rows above that top row are not rendered in the client. Use the **ScrollTo (on-line documentation)** method to move the specified cell.

If the **AllowPage ('AllowPage Property' in the on-line documentation)** property is False, the TopRow property takes effect. If **AllowPage ('AllowPage Property' in the on-line documentation)** is True; then TopRow only takes effect the first time the page is loaded.

Using Code

1. Set the **AllowPage ('AllowPage Property' in the on-line documentation)** property.
2. Set the **TopRow ('TopRow Property' in the on-line documentation)** property.

Example

This example code sets the top row.

C#

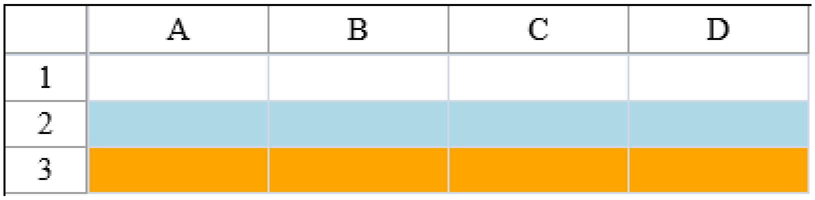
```
FpSpread1.ActiveSheetView.RowCount = 20;
FpSpread1.ActiveSheetView.ColumnCount = 7;
FpSpread1.ActiveSheetView.AllowPage = false;
FpSpread1.ActiveSheetView.TopRow = 4;
FpSpread1.ActiveSheetView.IgnoreHiddenRowsWhenPaging = false;
```

VB

```
FpSpread1.ActiveSheetView.AllowPage = False
FpSpread1.ActiveSheetView.ColumnCount = 7
FpSpread1.ActiveSheetView.RowCount = 20
FpSpread1.ActiveSheetView.IgnoreHiddenRowsWhenPaging = False
FpSpread1.ActiveSheetView.TopRow = 4
```

Creating Alternating Rows

You might want to set up your sheet so that alternating rows have 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. The following image displays alternating row colors:



	A	B	C	D
1				
2				
3				

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. Refer to the **AlternatingRow ('AlternatingRow Class' in the on-line documentation)** class.

Using a Shortcut

1. Set the **Count ('Count Property' in the on-line documentation)** property for the **AlternatingRows ('AlternatingRows Class' in the on-line documentation)** shortcut object.
2. Set the various appearance and other properties of the **AlternatingRows ('AlternatingRows Class' in the on-line documentation)** shortcut object, such as the **BackColor ('BackColor Property' in the on-line documentation)** and **ForeColor ('ForeColor Property' in the on-line documentation)** properties.
3. Create additional alternating row appearances by setting properties for additional **AlternatingRows ('AlternatingRows Class' in the on-line documentation)** 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 dark blue text, and the third row has an orange background with dark red text. This pattern repeats for all subsequent rows.

C#

```
FpSpread1.Sheets[0].AlternatingRows.Count = 3;
FpSpread1.Sheets[0].AlternatingRows[1].BackColor = Color.LightBlue;
FpSpread1.Sheets[0].AlternatingRows[1].ForeColor = Color.DarkBlue;
FpSpread1.Sheets[0].AlternatingRows[2].BackColor = Color.Orange;
FpSpread1.Sheets[0].AlternatingRows[2].ForeColor = Color.DarkRed;
```

VB

```
FpSpread1.Sheets(0).AlternatingRows.Count = 3
FpSpread1.Sheets(0).AlternatingRows(1).BackColor = Color.LightBlue
FpSpread1.Sheets(0).AlternatingRows(1).ForeColor = Color.DarkBlue
FpSpread1.Sheets(0).AlternatingRows(2).BackColor = Color.Orange
FpSpread1.Sheets(0).AlternatingRows(2).ForeColor = Color.DarkRed
```

Using the Spread Designer

1. From the **Settings** menu, select the **AlternatingRow** editor icon located under the **Other Settings** section.
2. In the **AlternatingRow** editor, add the number of alternating row objects you want to provide, using the **AltRow** section.
3. Select the alternating row object for which you want to set properties from the list of row objects.
4. Set the properties for the alternating row object in the list of properties.
5. Repeat steps 3 and 4 for each alternating row object you want to customize.
6. Click **OK** to close the editor.
7. Click **Apply and Exit** to close the Spread Designer.

Creating Row Templates (Multiple-Line Columns)

You can create row templates, also called aggregation subtotals or multiple-line columns. You can display multiple lines within a column, such as to display address information together in one column that involves multiple fields of information.

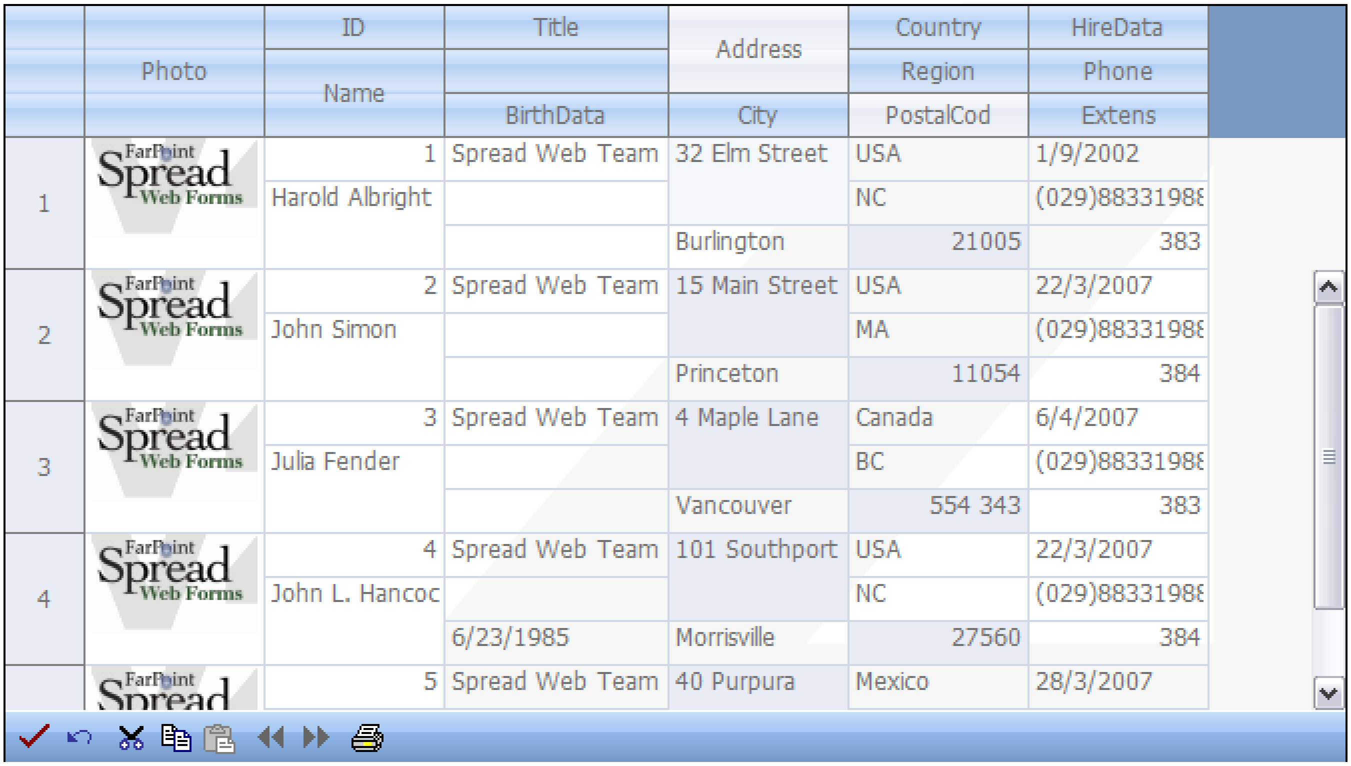


	Photo	ID	Title	Address	Country	HireData
		Name			Region	Phone
			BirthData	City	PostalCod	Extens
1	FarPoint Spread Web Forms	1	Spread Web Team	32 Elm Street	USA	1/9/2002
		Harold Albright			NC	(029)8833198
				Burlington	21005	383
2	FarPoint Spread Web Forms	2	Spread Web Team	15 Main Street	USA	22/3/2007
		John Simon			MA	(029)8833198
				Princeton	11054	384
3	FarPoint Spread Web Forms	3	Spread Web Team	4 Maple Lane	Canada	6/4/2007
		Julia Fender			BC	(029)8833198
				Vancouver	554 343	383
4	FarPoint Spread Web Forms	4	Spread Web Team	101 Southport	USA	22/3/2007
		John L. Hancoc			NC	(029)8833198
			6/23/1985	Morrisville	27560	384
	FarPoint Spread	5	Spread Web Team	40 Purpura	Mexico	28/3/2007

In this figure, the ID and name information appear staggered in a single column and the street address and city information appear in the same column.

The parts of the API that are involved with this feature include:

- SheetView.**LayoutMode ('LayoutMode Property' in the on-line documentation)** property
- SheetView.**WorksheetTemplate ('WorksheetTemplate Property' in the on-line documentation)**
- Worksheet.**RowTemplate ('RowTemplate Property' in the on-line documentation)** property
- **LayoutTemplate ('LayoutTemplate Class' in the on-line documentation)** class
- **LayoutCell ('LayoutCell Class' in the on-line documentation)** class
- **LayoutCells ('LayoutCells Class' in the on-line documentation)** class
- **LayoutColumn ('LayoutColumn Class' in the on-line documentation)** class
- **LayoutColumns ('LayoutColumns Class' in the on-line documentation)** class
- **LayoutRow ('LayoutRow Class' in the on-line documentation)** class
- **LayoutRows ('LayoutRows Class' in the on-line documentation)** class
- **SheetView.LayoutModeType ('SheetView.LayoutModeType Enumeration' in the on-line documentation)** enumeration

The worksheet template contains a column header template and a row template. Layout information such as cell spans, column count, and row count is stored in the worksheet template.

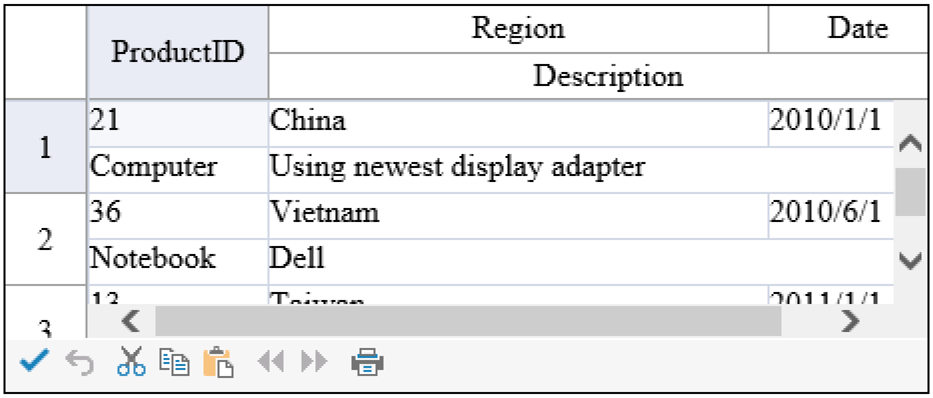
This feature has the following effects on other features:

- The row count of the column header and the column count of the row header are limited.
- This does not support changing the row height by the drag and drop operation, but it does support changing the column width by the drag and drop operation.
- The frozen columns feature is not supported, but the frozen rows feature is supported.The selection operation only supports the single selection policy of a sheet (SheetView object).
- The Axis model of Spread is limited: you cannot set row height or column width for the viewport, the row header, or the column header.
- The Span model of Spread is limited: you cannot set spans in the viewport, row header, column header, or

column footer. You can get similar effects by spanning cells in the row template.

- This does not support the operation of moving a column by dragging and dropping.
- Automatic merging is no longer supported.

The following code example creates this image.



	ProductID	Region	Date
		Description	
1	21	China	2010/1/1
	Computer	Using newest display adapter	
2	36	Vietnam	2010/6/1
	Notebook	Dell	
3	13	Taiwan	2011/1/1

Using Code

1. Set the **LayoutMode ('LayoutMode Property' in the on-line documentation)** property for the sheet.
2. Set the template to the **WorksheetTemplate ('WorksheetTemplate Property' in the on-line documentation)** property for the sheet.
3. Set the **ColumnCount ('ColumnCount Property' in the on-line documentation)** property for the template.
4. Set the row count for the column header template and the row template.
5. Set the cell spans for the column header and row templates.
6. Create data for the cells.
7. Use the **DataIndex ('DataIndex Property' in the on-line documentation)** property to put data in the cell.

Example

This example assigns a layout mode for the column headers.

C#

```
protected void Page_Load(object sender, System.EventArgs e)
{
if (this.IsPostBack) return;
FpSpread1.ActiveSheetView.LayoutMode =
FarPoint.Web.Spread.SheetView.LayoutModeType.RowTemplateLayoutMode;
FarPoint.Web.Spread.WorksheetTemplate template1 =
FpSpread1.Sheets[0].WorksheetTemplate;
template1.ColumnCount = 3;
template1.ColumnHeaderTemplate.RowCount = 2;
template1.RowTemplate.RowCount = 2;
template1.LayoutColumns[1].Width = 250;
//Set row template's layout
template1.RowTemplate.LayoutCells[1, 1].ColumnSpan = 2;
//set column header template's layout
template1.ColumnHeaderTemplate.LayoutCells[0, 0].RowSpan = 2;
template1.ColumnHeaderTemplate.LayoutCells[1, 1].ColumnSpan = 2;
DataTable dt = new DataTable();
dt.Columns.Add("ProductID");
dt.Columns.Add("ProductName");
dt.Columns.Add("Region");
dt.Columns.Add("Date");
dt.Columns.Add("Description");
dt.Rows.Add(new object[] { 21, "Computer", "China", "2010/1/1", "Using newest display
adapter" });
dt.Rows.Add(new object[] { 36, "Notebook", "Vietnam", "2010/6/1", "Dell" });
dt.Rows.Add(new object[] { 13, "Hard disk", "Taiwan", "2011/1/1", "Speed is 7200" });
FpSpread1.Sheets[0].DataSource = dt;
template1.LayoutCells[0, 0].DataIndex = 0;
template1.LayoutCells[1, 0].DataIndex = 1;
template1.LayoutCells[0, 1].DataIndex = 2;
template1.LayoutCells[0, 2].DataIndex = 3;
template1.LayoutCells[1, 1].DataIndex = 4;
}
```

VB

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
If (IsPostBack) Then
    Return
End If
FpSpread1.ActiveSheetView.LayoutMode =
FarPoint.Web.Spread.SheetView.LayoutModeType.RowTemplateLayoutMode
Dim template1 As FarPoint.Web.Spread.WorksheetTemplate =
FpSpread1.Sheet(0).WorksheetTemplate
template1.ColumnCount = 3
template1.ColumnHeaderTemplate.RowCount = 2
template1.RowTemplate.RowCount = 2
template1.LayoutColumns(1).Width = 250
'Set row template's layout
template1.RowTemplate.LayoutCells(1, 1).ColumnSpan = 2
'set column header template's layout
template1.ColumnHeaderTemplate.LayoutCells(0, 0).RowSpan = 2
template1.ColumnHeaderTemplate.LayoutCells(1, 1).ColumnSpan = 2
Dim dt As New DataTable()
dt.Columns.Add("ProductID")
dt.Columns.Add("ProductName")
dt.Columns.Add("Region")
dt.Columns.Add("Date")
dt.Columns.Add("Description")
dt.Rows.Add(New Object() {21, "Computer", "China", "2010/1/1", "Using newest display
adapter"})
dt.Rows.Add(New Object() {36, "Notebook", "Vietnam", "2010/6/1", "Dell"})
dt.Rows.Add(New Object() {13, "Hard disk", "Taiwan", "2011/1/1", "Speed is 7200"})
FpSpread1.Sheets(0).DataSource = dt
template1.LayoutCells(0, 0).DataIndex = 0
template1.LayoutCells(1, 0).DataIndex = 1
template1.LayoutCells(0, 1).DataIndex = 2
template1.LayoutCells(0, 2).DataIndex = 3
template1.LayoutCells(1, 1).DataIndex = 4
End Sub
```

Using the Spread Designer

1. Select the **Settings** menu.

2. Select the **Row Template** icon under the **Other Settings** section.
3. Set the various template properties.
4. Click **OK**.
5. Click **Apply and Exit** to close the Spread Designer.

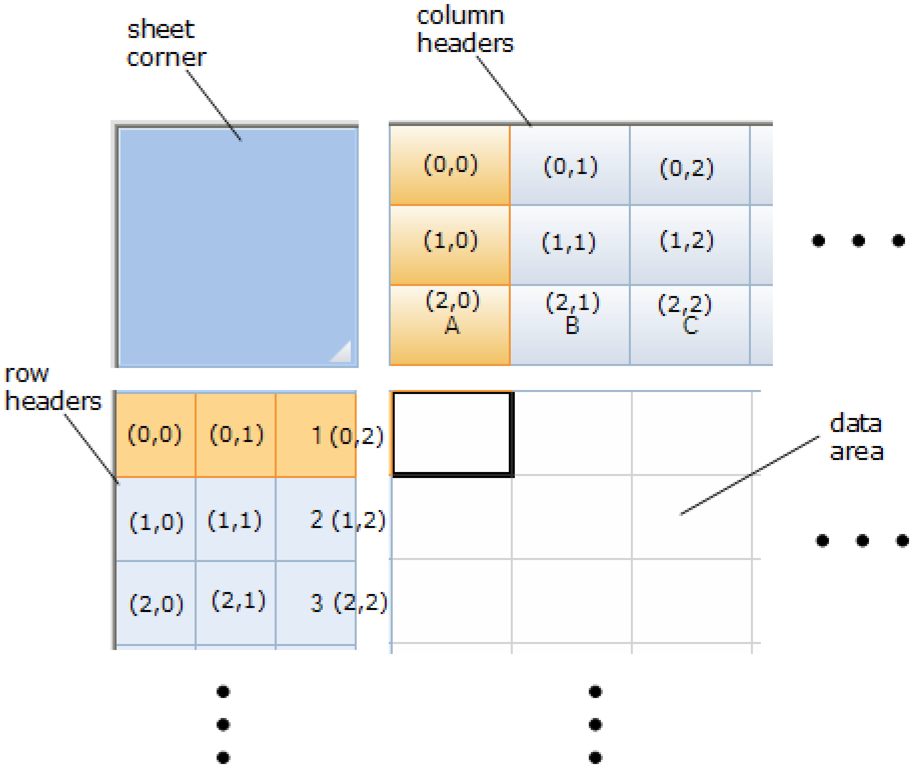
Customizing the Appearance of Headers

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

- **Customizing the Style of Header Cells**
- **Showing or Hiding Headers**
- **Customizing the Default Header Labels**
- **Customizing Header Label Text**
- **Setting the Size of Header Cells**
- **Customizing the Header Empty Areas**
- **Creating a Header with Multiple Rows or Columns**
- **Creating a Span in a Header**

Headers provide labels to identify the columns and rows. They appear at the top (for columns) and to the left (for rows) of the data cells and are formatted differently to be clearly seen. You may customize the appearance of header cells as you would any of the cells in the spreadsheet component. When you work with row headers and column headers, you can manipulate the objects using the short cuts in code (**RowHeader ('RowHeader Class' in the on-line documentation)** and **ColumnHeader ('ColumnHeader Class' in the on-line documentation)** classes), or you can directly manipulate the model. Most developers who are not changing anything drastically find it easy to manipulate the short cut objects.

The figure below shows the parts of the headers and the coordinates of cells in headers that have multiple rows and columns.



For more information on the Cell and Cells objects, refer to the **Assembly Reference (on-line documentation)**.

For more information on models, refer to **Using Sheet Models**.

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

Customizing the Style of Header Cells

You can customize the style of header cells if you want to change the default appearance. Set the default style of the header cells by setting the **DefaultStyle** property in the **RowHeader ('RowHeader Class' in the on-line documentation)** or **ColumnHeader ('ColumnHeader Class' in the on-line documentation)** class. For more information on what can be set, refer to the StyleInfo object and the RowHeader and ColumnHeader objects in the **Assembly Reference (on-line documentation)**.

You can also change some of the properties of the **SheetSkin ('SheetSkin Class' in the on-line documentation)** class that customize the appearance of header cells and apply the skin the sheet. These properties include **FlatRowHeader ('FlatRowHeader Property' in the on-line documentation)** and **FlatColumnHeader ('FlatColumnHeader Property' in the on-line documentation)**. For more information creating and applying skins, refer to **Applying a Skin to a Sheet** and **Creating a Skin for Sheets**.

When defining and applying a custom style to header cells, be sure to set the text alignment. The default renderer (without any style applied) centers the text; if you apply a style, and do not set the alignment, the text is left-aligned not centered.

You can also set the grid lines around the header cells with the **Border ('Border Class' in the on-line documentation)** property.

Using the Properties Window

1. At design time, in the **Properties** window, select the FpSpread component.
2. Select the **NamedStyles Collection** drop-down button.
3. Add a style and set the properties.
4. Click **OK**. Select the cells you want to apply the style to and set the **StyleName** property.
5. Use the **SheetView Collection Editor** to set the **DefaultStyleName** for a column header, row header, column footer, or sheet corner.
6. Click **OK** to close the editor.

Using a Shortcut

1. To change the style for the column header, define a style and then set the **ColumnHeader ('ColumnHeader Class' in the on-line documentation) 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 ('RowHeader Class' in the on-line documentation) 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.

C#

```
// Define a new style.
FarPoint.Web.Spread.StyleInfo darkstyle = new FarPoint.Web.Spread.StyleInfo();
darkstyle.BackColor = Color.Teal;
darkstyle.ForeColor = Color.Yellow;
darkstyle.Border = new FarPoint.Web.Spread.Border(Color.Crimson);
```

```
// Apply the new style.
FpSpread1.ActiveSheetView.ColumnHeader.DefaultStyle = darkstyle;
```

VB

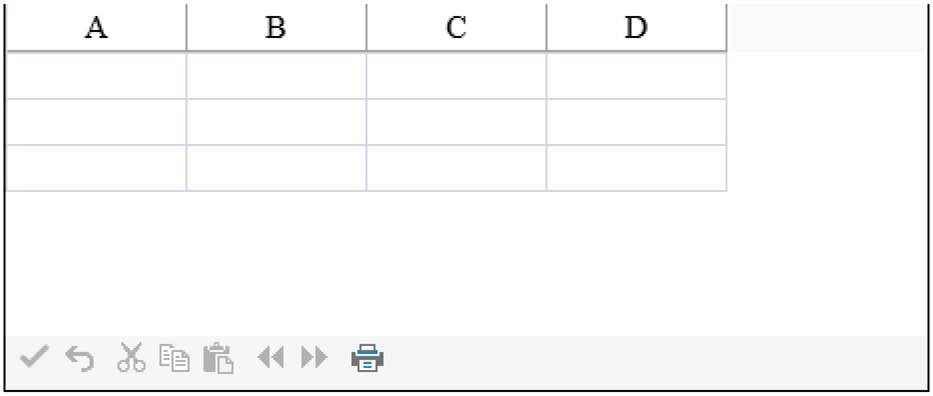
```
' Define a new style.
Dim darkstyle As New FarPoint.Web.Spread.StyleInfo()
darkstyle.BackColor = Color.Teal
darkstyle.ForeColor = Color.Yellow
darkstyle.Border = New FarPoint.Web.Spread.Border(Color.Crimson)
' Apply the new style.
FpSpread1.ActiveSheetView.ColumnHeader.DefaultStyle = darkstyle
```

Using the Spread Designer

1. Select the **Settings** menu.
2. Select the **Named Style** icon under the **Appearance Settings** section.
3. Use the new style icon to create a style name. Use the edit style icon to set properties for the style.
4. Click **OK**. Close the **NamedStyle** dialog.
5. In order to apply the style to a header, select the **Header Editor** icon under **Other Settings**. Use the **Selected Header** option to specify column header, row header, column footer, or sheet corner.
6. Set the **DefaultStyleName**. Click **Apply** and **OK**.
7. For cells, select the **Cells, Columns, and Rows Editor**.
8. Select the cells you want to apply the style to.
9. Set the **StyleName**.
10. Choose **Apply** and **OK** to apply your changes to the component.
11. Click **Apply and Exit** to close the Spread Designer.

Showing or Hiding Headers

By default, the row headers and column headers are displayed in the component. You can hide the row headers, the 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 the property window 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 Rows or Columns**.

The display of headers is done by simply setting the row header **Visible ('Visible Property' in the on-line documentation)** property or column header **Visible ('Visible Property' in the on-line documentation)** property.

Alternatively, if you prefer you can customize the headers by providing custom text or headers with multiple columns or rows, as explained in **Customizing Header Label Text** and **Creating a Header with Multiple Rows or Columns**.

Using the Properties Window

1. At design time, in the **Properties** window, select the FpSpread 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 **Visible** property for the ColumnHeader object or the **Visible** property for the RowHeader object to false to turn off the display of the header.
6. Click **OK** to close the editor.

Using a Shortcut

Set the **ColumnHeader ('ColumnHeader Class' in the on-line documentation) Visible ('Visible Property' in the on-line documentation)** (or **SheetView ('SheetView Class' in the on-line documentation) ColumnHeaderVisible ('ColumnHeaderVisible Property' in the on-line documentation)**) property or **RowHeader ('RowHeader Class' in the on-line documentation) Visible ('Visible Property' in the on-line documentation)** (or **SheetView ('SheetView Class' in the on-line documentation) RowHeaderVisible ('RowHeaderVisible Property' in the on-line documentation)**) property for a sheet.

Example

This example turns off the display of the row header.

C#

```
// Turn off the display of row headers.
FpSpread1.Sheets[0].RowHeader.Visible = false;
FpSpread1.Sheets[0].ColumnCount = 4;
```

VB

```
' Turn off the display of row headers.
FpSpread1.Sheets(0).RowHeader.Visible = False
FpSpread1.Sheets[0].ColumnCount = 4
```

Using Code

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

Example

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

C#

```
// Create a new sheet.
FarPoint.Web.Spread.SheetView newsheet = new FarPoint.Web.Spread.SheetView();
```

```
newsheet.ColumnHeaderVisible = false;
// Set first sheet equal to SheetView object.
FpSpread1.Sheets[0] = newsheet;
```

VB

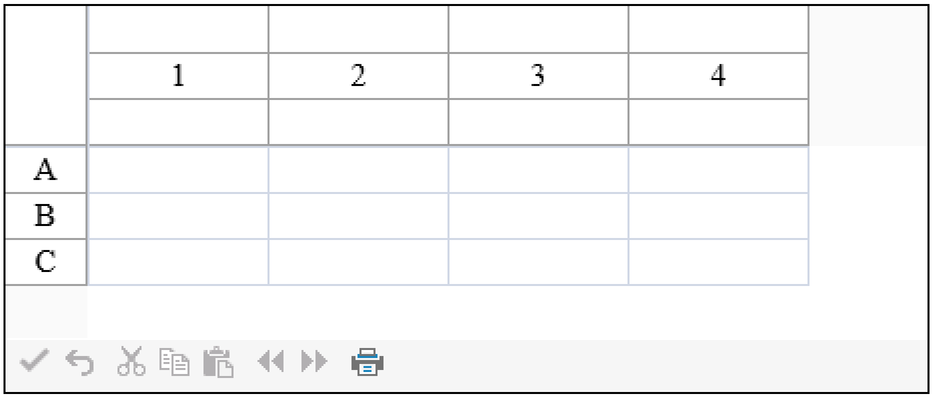
```
' Create a new sheet.
Dim newsheet As New FarPoint.Web.Spread.SheetView()
newsheet.ColumnHeaderVisible = False
' Set first sheet equal to SheetView object.
FpSpread1.Sheets(0) = newsheet
```

Using the Spread Designer

1. Select the sheet tab for the sheet for which you want to turn off header display.
2. In the **View** menu select or deselect the **Row Header** or **Column Header** check box.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Customizing the Default Header Labels

By default the 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 display no 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.

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 FpSpread 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 ColumnHeader AutoText or RowHeader AutoText property.
6. To change the row or column in the header in which the label is displayed, change the setting of the ColumnHeader AutoTextIndex or RowHeader AutoTextIndex property (the index is zero-based).
7. Click **OK** to close the editor.

Using a Shortcut

1. To change the settings for the column header, set the **ColumnHeaderAutoText ('ColumnHeaderAutoText Property' in the on-line documentation)** property for the sheet (or the **AutoText ('AutoText Property' in the on-line documentation)** property of the **ColumnHeader ('ColumnHeader Class' in the on-line documentation)** object) and the **ColumnHeaderAutoTextIndex ('ColumnHeaderAutoTextIndex Property' in the on-line documentation)** property of the sheet (or the **AutoTextIndex ('AutoTextIndex Property' in the on-line documentation)** property of the **ColumnHeader ('ColumnHeader Class' in the on-line documentation)** object). Use the **HeaderAutoText ('HeaderAutoText Enumeration' in the on-line documentation)** enumeration with the auto text properties.
2. To change the settings for the row header, set the **RowHeaderAutoText ('RowHeaderAutoText Property' in the on-line documentation)** property for the sheet (or the **AutoText ('AutoText Property' in the on-line documentation)** property of the **RowHeader ('RowHeader Class' in the on-line documentation)** object) and the **RowHeaderAutoTextIndex ('RowHeaderAutoTextIndex Property' in the on-line documentation)** property for the sheet (or the **AutoTextIndex ('AutoTextIndex Property' in the on-line documentation)** property of the **RowHeader ('RowHeader Class' in the on-line documentation)** object).

Example

This example code sets the column header to display numbers instead of letters and changes the row header to letters.

C#

```
// Set the column header to display numbers instead of letters.
FpSpread1.Sheets[0].ColumnHeaderAutoTextIndex = 1;
FpSpread1.Sheets[0].ColumnHeaderAutoText = FarPoint.Web.Spread.HeaderAutoText.Numbers;
// Change row headers to letters
FpSpread1.Sheets[0].RowHeaderAutoText = FarPoint.Web.Spread.HeaderAutoText.Letters;
```

VB

```
' Set the column header to display numbers instead of letters.
FpSpread1.Sheets(0).ColumnHeaderAutoTextIndex = 1
FpSpread1.Sheets(0).ColumnHeaderAutoText = FarPoint.Web.Spread.HeaderAutoText.Numbers
' Change row headers to letters
FpSpread1.Sheets(0).RowHeaderAutoText = FarPoint.Web.Spread.HeaderAutoText.Letters
```

Using the Spread Designer

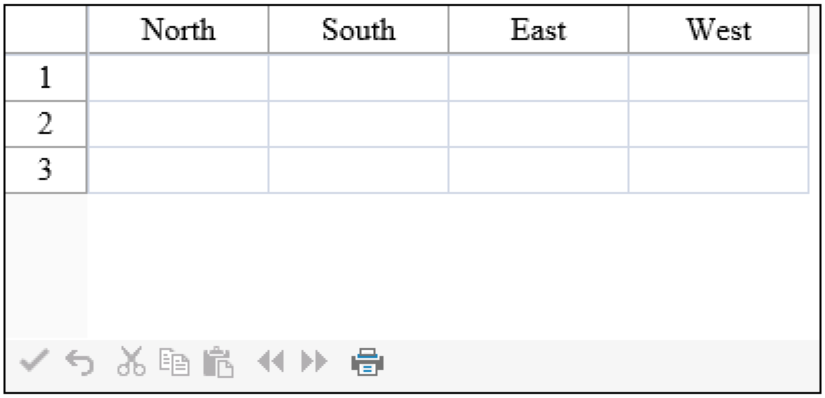
1. Select the sheet for which you want to modify the header label (automatic text) settings.
2. Select the **Settings** menu. Select the **Headers** icon in the **Sheet Settings** section. Select the **Column** or **Row Header** tab.
3. Change the settings of the **AutoText** and **AutoTextIndex** properties to specify the header label to display and which column or row in the header should display the automatic text. You can also specify other header

properties such as the starting column or row number.

4. Click **OK** to close the dialog.
5. 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 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 customize the header label text, as shown in the following figure where the first four columns have custom labels.



	North	South	East	West
1				
2				
3				

To specify the custom text for a header label, you can use 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 or you can use the **Cell ('Cell Class' in the on-line documentation) Text ('Text Property' in the on-line documentation)** property. For headers with multiple columns and multiple rows, you use the **Text ('Text Property' in the on-line documentation)** property of the Cells shortcut objects. Refer to the example in **Creating a Header with Multiple Rows or Columns**.

To customize the sequential letters in column headers and sequential numbers in row headers that are displayed by default, refer to **Customizing the Default Header Labels**.

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 figuring header cell coordinates.

Using the Properties Window

1. At design time, in the **Properties** window, select the FpSpread 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 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 **Cells ('Cells Class' in the on-line documentation)** object of the **ColumnHeader ('ColumnHeader Class' in the on-line documentation)** to the custom text you want to display. If you want to set the text for multiple header cells, call the **SetClip ('SetClip Method' in the on-line documentation)** or **SetClipValue ('SetClipValue Method' in the on-line documentation)** methods for the **ColumnHeader ('ColumnHeader Class' in the on-line documentation)**. The same applies to a **RowHeader ('RowHeader Class' in the on-line documentation)** object.
- If you want to change the labels displayed, set the **Label ('Label Property' in the on-line documentation)** property for the **Columns ('Columns Class' in the on-line documentation)** 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].ColumnCount = 4;
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).ColumnCount = 4
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 at the bottom of the designer.
2. Select the **Settings** menu.
3. Select the **Header** editor in the **Other Settings** section.
4. Select the header you wish to edit.
5. Set the **Label** property in the property grid.
6. Choose **Apply** and **OK** to apply your changes to the component.
7. Click **Apply and Exit** to close the Spread Designer.

Setting the Size of Header Cells

You can customize the appearance of header cells by changing the row height or column width, or both, for any of the rows or columns of headers. You can change the size by setting 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, or use the respective **SheetView ('SheetView Class' in the on-line documentation)** class properties. Use these properties:

- ColumnHeader.**Height ('Height Property' in the on-line documentation)**
- RowHeader.**Width ('Width Property' in the on-line documentation)**
- SheetView.**ColumnHeaderHeight ('ColumnHeaderHeight Property' in the on-line**

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

You can also use the Spread Designer to set the width and height of header cells.

For information on setting the size of cells in the data area, refer to **Setting the Row Height or Column Width**.

Using Code

Set the height or width for the headers.

Example

This example code sets the height and width of the headers.

C#

```
// Set the height and width for the headers.
FpSpread1.Sheets[0].ColumnHeader.Height = 60;
FpSpread1.Sheets[0].RowHeader.Width = 60;
```

VB

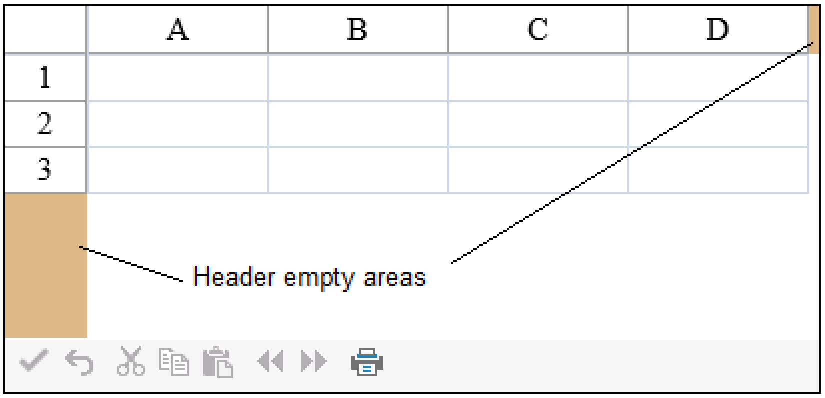
```
' Set the height and width for the headers.
FpSpread1.Sheets(0).ColumnHeader.Height = 60
FpSpread1.Sheets(0).RowHeader.Width = 60
```

Using the Spread Designer

1. Select the **Settings** menu.
2. Select the **Header Editor** icon in the **Other Settings** section.
3. Select the header you wish to edit.
4. Set the height or width properties.
5. Choose **Apply** and **OK** to apply your changes to the component.
6. Click **Apply and Exit** to close the Spread Designer.

Customizing the Header Empty Areas

By default the component displays a color in the empty areas not filled in with column or row headers. These are the header empty areas as illustrated in the figure.



To customize the color of the header empty area, you can use the **SheetView ('SheetView Class' in the on-line**

documentation) HeaderGrayAreaColor ('HeaderGrayAreaColor Property' in the on-line documentation) property. You can add images to the empty area with the **HeaderGrayAreaBackgroundImageUrl ('HeaderGrayAreaBackgroundImageUrl Property' in the on-line documentation)** property.

Using the 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. In the **Members** list, select the sheet for which you want to set the color of the header empty area.
4. Select the **HeaderGrayAreaColor** property (or **HeaderGrayAreaBackgroundImageUrl** and select an image) in the property list, and then click the drop-down button to display the color picker.
5. Select a color in the color picker.
6. Click **OK** to close the editor.

Using Code

Set the **HeaderGrayAreaBackgroundImageUrl ('HeaderGrayAreaBackgroundImageUrl Property' in the on-line documentation)** property.

Example

This example code sets the color or image properties for the gray area.

C#

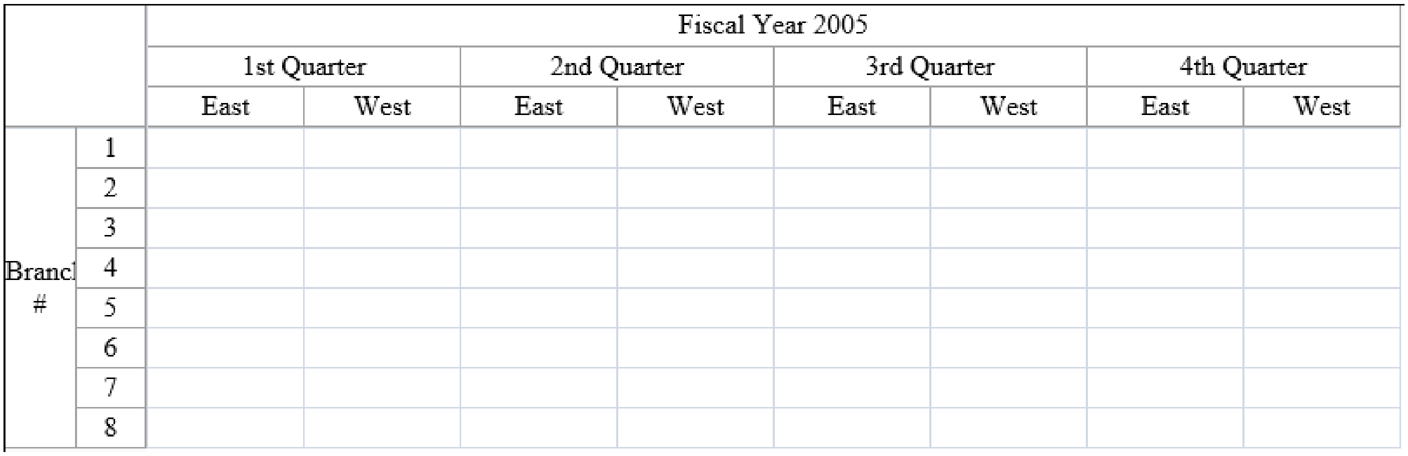
```
// Set a color or an image for the gray area.
FpSpread1.Sheets[0].HeaderGrayAreaBackgroundImageUrl = "happy.bmp";
//FpSpread1.Sheets[0].HeaderGrayAreaColor = Color.BurlyWood;
```

VB

```
' Set a color or an image for the gray area.
FpSpread1.Sheets(0).HeaderGrayAreaBackgroundImageUrl = "happy.bmp
'FpSpread1.Sheets(0).HeaderGrayAreaColor = Color.BurlyWood
```

Creating a Header with Multiple Rows or Columns

You can provide multiple rows in the column header and multiple columns in the row header. As shown in the following figure, the headers may have different numbers of columns and rows.



		Fiscal Year 2005							
		1st Quarter		2nd Quarter		3rd Quarter		4th Quarter	
		East	West	East	West	East	West	East	West
Brancl #	1								
	2								
	3								
	4								
	5								
	6								
	7								
	8								

The rows or columns in the header can also contain spans, for example, if you want to have a header cell that explains two header 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**.

Using the Properties Window

1. At design time, in the **Properties** window, select the FpSpread 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 ColumnHeader **RowCount** property to the number or rows you want in the column header or the RowHeader **ColumnCount** property to the number of columns you want in the row header.
6. Click **OK** to close the editor.

Using a Shortcut

Set the **RowCount ('RowCount Property' in the on-line documentation)** property for the **ColumnHeader ('ColumnHeader Class' in the on-line documentation)** object and the **ColumnCount ('ColumnCount Property' in the on-line documentation)** property for the **RowHeader ('RowHeader Class' in the on-line documentation)** object.

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#

```
FpSpread1.Sheets[0].ColumnCount = 8;
FpSpread1.Sheets[0].RowCount = 8;
// Set the number or rows and columns in the headers.
FpSpread1.Sheets[0].ColumnHeader.RowCount = 3;
FpSpread1.Sheets[0].RowHeader.ColumnCount = 2;

// Span the header cells as needed.
FpSpread1.Sheets[0].ColumnHeaderSpanModel.Add(0, 0, 1, 8);
FpSpread1.Sheets[0].RowHeaderSpanModel.Add(0,0,12,1);
```

```
FpSpread1.Sheets[0].ColumnHeaderSpanModel.Add(1, 0, 1, 2);
FpSpread1.Sheets[0].ColumnHeaderSpanModel.Add(1, 2, 1, 2);
FpSpread1.Sheets[0].ColumnHeaderSpanModel.Add(1, 4, 1, 2);
FpSpread1.Sheets[0].ColumnHeaderSpanModel.Add(1, 6, 1, 2);
FpSpread1.Sheets[0].ColumnHeaderSpanModel.Add(1, 8, 1, 2);

// Set the labels as needed --
// using the Label property or the cell Text property.
FpSpread1.Sheets[0].ColumnHeader.Cells[0, 0].Text = "Fiscal Year 2005";
FpSpread1.Sheets[0].RowHeader.Cells[0, 0].Text = "Branch #";
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";

FpSpread1.Sheets[0].ColumnHeader.Cells[2, 0].Text = "East";
FpSpread1.Sheets[0].ColumnHeader.Cells[2, 1].Text = "West";
FpSpread1.Sheets[0].ColumnHeader.Cells[2, 2].Text = "East";
FpSpread1.Sheets[0].ColumnHeader.Cells[2, 3].Text = "West";
FpSpread1.Sheets[0].ColumnHeader.Cells[2, 4].Text = "East";
FpSpread1.Sheets[0].ColumnHeader.Cells[2, 5].Text = "West";
FpSpread1.Sheets[0].ColumnHeader.Cells[2, 6].Text = "East";
FpSpread1.Sheets[0].ColumnHeader.Cells[2, 7].Text = "West";
```

VB

```
FpSpread1.Sheets(0).RowCount = 8
FpSpread1.Sheets(0).ColumnCount = 8
’ Set the number or rows and columns in the headers.
FpSpread1.Sheets(0).ColumnHeader.RowCount = 3
FpSpread1.Sheets(0).RowHeader.ColumnCount = 2

'Span the header cells as needed.
FpSpread1.Sheets(0).ColumnHeaderSpanModel.Add(0, 0, 1, 8)
FpSpread1.Sheets(0).RowHeaderSpanModel.Add(0,0,12,1)

FpSpread1.Sheets(0).ColumnHeaderSpanModel.Add(1, 0, 1, 2)
FpSpread1.Sheets(0).ColumnHeaderSpanModel.Add(1, 2, 1, 2)
FpSpread1.Sheets(0).ColumnHeaderSpanModel.Add(1, 4, 1, 2)
FpSpread1.Sheets(0).ColumnHeaderSpanModel.Add(1, 6, 1, 2)
FpSpread1.Sheets(0).ColumnHeaderSpanModel.Add(1, 8, 1, 2)

'Set the labels as needed --
'using the Label property or the cell Text property.
FpSpread1.Sheets(0).ColumnHeader.Cells(0, 0).Text = "Fiscal Year 2005"
FpSpread1.Sheets(0).RowHeader.Cells(0, 0).Text = "Branch #"

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"

FpSpread1.Sheets(0).ColumnHeader.Cells(2, 0).Text = "East"
FpSpread1.Sheets(0).ColumnHeader.Cells(2, 1).Text = "West"
FpSpread1.Sheets(0).ColumnHeader.Cells(2, 2).Text = "East"
FpSpread1.Sheets(0).ColumnHeader.Cells(2, 3).Text = "West"
FpSpread1.Sheets(0).ColumnHeader.Cells(2, 4).Text = "East"
```

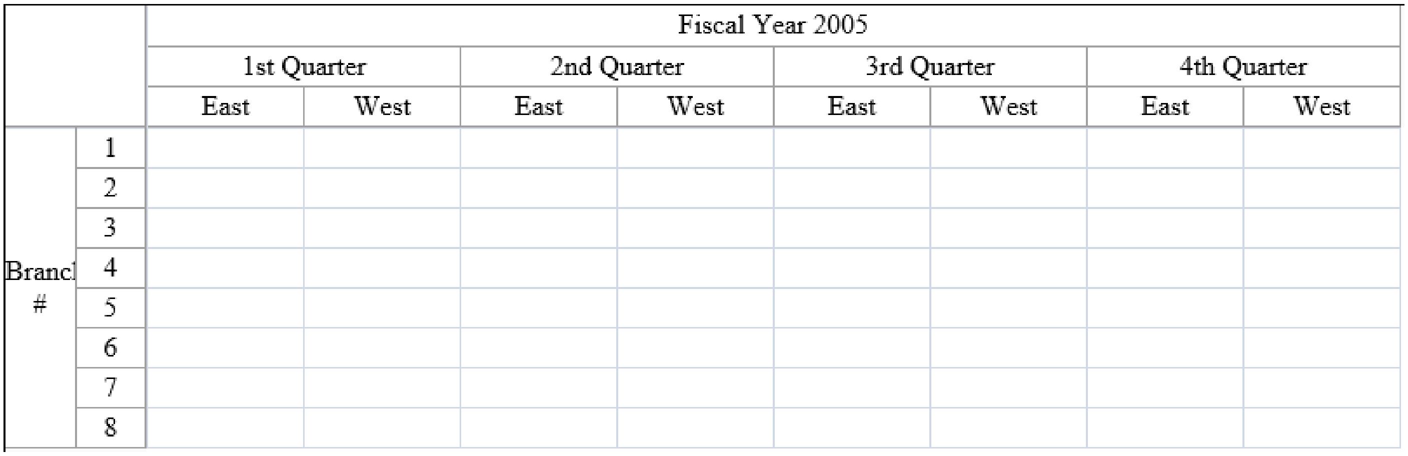
```
FpSpread1.Sheets(0).ColumnHeader.Cells(2, 5).Text = "West"
FpSpread1.Sheets(0).ColumnHeader.Cells(2, 6).Text = "East"
FpSpread1.Sheets(0).ColumnHeader.Cells(2, 7).Text = "West"
```

Using the Spread Designer

1. Select the sheet tab for the sheet for which you want to display multiple header rows or columns.
2. Select the **Settings** menu.
3. Select the **Header Editor** icon in the **Other Settings** section.
4. Select **Column Header** or **Row Header** in the **Selected Header** drop-down box.
5. 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 in the property grid.
6. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Creating a Span in a Header

You can create cell spans in a header, for example, to make a header for multiple columns of data, as shown in the following figure.



		Fiscal Year 2005							
		1st Quarter		2nd Quarter		3rd Quarter		4th Quarter	
		East	West	East	West	East	West	East	West
Branc #	1								
	2								
	3								
	4								
	5								
	6								
	7								
	8								

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 create cell spans in either the column or row headers or both. For more background about creating cell spans, refer to **Spanning Cells**.

You can customize the labels in these headers. For instructions for customizing the labels, see **Customizing Header Label Text**.

Using a Shortcut

Define the number of rows in the column header (and columns in the rows header) and then set the header cells to span using the **Add ('Add Method' in the on-line documentation)** property in the span model.

Example

This example creates multiple headers and adds cell spans.

C#

```
// Set the number or rows and columns in the headers.
```

```
FpSpread1.Sheets[0].ColumnHeader.RowCount = 3;
FpSpread1.Sheets[0].RowHeader.ColumnCount = 2;
// Span the header cells as needed.
FpSpread1.Sheets[0].ColumnHeaderSpanModel.Add(0, 0, 1, 8);
FpSpread1.Sheets[0].RowHeaderSpanModel.Add(0,0,12,1);
FpSpread1.Sheets[0].ColumnHeaderSpanModel.Add(1, 0, 1, 2);
FpSpread1.Sheets[0].ColumnHeaderSpanModel.Add(1, 2, 1, 2);
FpSpread1.Sheets[0].ColumnHeaderSpanModel.Add(1, 4, 1, 2);
FpSpread1.Sheets[0].ColumnHeaderSpanModel.Add(1, 6, 1, 2);
FpSpread1.Sheets[0].ColumnHeaderSpanModel.Add(1, 8, 1, 2);
```

VB

```
'Set the number or rows and columns in the headers.
FpSpread1.Sheets(0).ColumnHeader.RowCount = 3
FpSpread1.Sheets(0).RowHeader.ColumnCount = 2
'Span the header cells as needed.
FpSpread1.Sheets(0).ColumnHeaderSpanModel.Add(0, 0, 1, 8)
FpSpread1.Sheets(0).RowHeaderSpanModel.Add(0,0,12,1)
FpSpread1.Sheets(0).ColumnHeaderSpanModel.Add(1, 0, 1, 2)
FpSpread1.Sheets(0).ColumnHeaderSpanModel.Add(1, 2, 1, 2)
FpSpread1.Sheets(0).ColumnHeaderSpanModel.Add(1, 4, 1, 2)
FpSpread1.Sheets(0).ColumnHeaderSpanModel.Add(1, 6, 1, 2)
FpSpread1.Sheets(0).ColumnHeaderSpanModel.Add(1, 8, 1, 2)
```

Using the Spread Designer

1. Select the **Settings** menu, and then the **Header Editor** icon.
2. Select a cell (the cells represent the header cells in this editor). Set the number of columns or rows to span with the **ColumnSpan** or **RowSpan** property.
3. When done, choose **Apply** and **OK** to apply your changes.
4. Click **Apply and Exit** to close the Spread Designer.

Customizing the Appearance of a Cell

You can set the appearance of individual cells in the data area of the spreadsheet. These tasks relate to setting the appearance of individual cells:

- **Working with the Active Cell**
- **Customizing the Colors of a Cell**
- **Aligning Cell Contents**
- **Customizing Cell Borders**
- **Customizing the Margins and Spacing of the Cell**
- **Creating and Applying a Custom Style for Cells**
- **Assigning a Cascading Style Sheet to a Cell**
- **Creating a Range of Cells**
- **Spanning Cells**
- **Allowing Cells to Merge Automatically**
- **Using Sparklines**

When you work with cells, you can manipulate 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 directly manipulate the model. Most developers who are not changing anything drastically find it easy to manipulate the shortcut objects.

Note: We use the word "appearance" in the general sense of the look and feel of the cell, not simply the settings in the **Appearance ('Appearance Class' in the on-line documentation)** 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.

Remember that 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 with Cell Types**.

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 with Cells**.

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

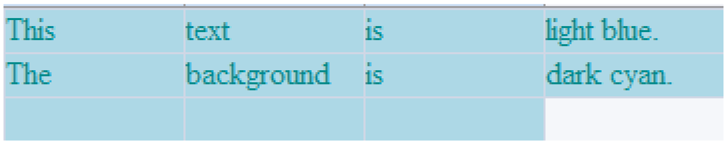
Working with the Active Cell

The active cell is the cell that currently receives any user interaction. Typically, the active cell appears with some form of highlighting to distinguish it from the other cells in the data area and to indicate that it is the active cell.

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 Appearance of Selections**.

Customizing the Colors of a Cell

You can set the background and foreground (text) colors for a cell or for a group of cells. The following figure shows the background and text colors of the data area changed from the default values with light blue text on a dark background.



This	text	is	light blue.
The	background	is	dark cyan.

To change the text and background colors, use the **BackColor ('BackColor Property' in the on-line documentation)** and **ForeColor ('ForeColor Property' in the on-line documentation)** properties of the **Cell ('Cell Class' in the on-line documentation)** class or the **BackColor ('BackColor Property' in the on-line documentation)** and **ForeColor ('ForeColor Property' in the on-line documentation)** properties of the **StyleInfo ('StyleInfo Class' in the on-line documentation)** class and apply the style to the cells. Alternatively, you can set the **CellBackColor ('CellBackColor Property' in the on-line documentation)** and **CellForeColor ('CellForeColor Property' in the on-line documentation)** properties of the **SheetSkin ('SheetSkin Class' in the on-line documentation)** class and apply the skin to the sheet. For more information on styles for cells, refer to **Creating and Applying a Custom Style for Cells**. For more information on skins to apply to sheets, refer to **Creating a Skin for Sheets** and **Applying a Skin to a Sheet**.

You can also set the color for cells to change when they are selected. You can set the **SelectionBackColor ('SelectionBackColor Property' in the on-line documentation)** and **SelectionForeColor ('SelectionForeColor Property' in the on-line documentation)** to change the background color and text color of selected cells. This is done either to the sheet directly with the **SheetView ('SheetView Class' in the on-line documentation)** class or with the skin that is applied to a sheet with the **SheetSkin ('SheetSkin Class' in the on-line documentation)** class. For more information refer to **Customizing the Appearance of Selections**.

For information about cascading style sheets, refer to **Assigning a Cascading Style Sheet to a Cell**.

Using the Properties Window

1. At design time, in the **Properties** window, select the FpSpread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. Select the **Cells** collection and then select **BackColor** under the **Misc.** section.
5. Click the **BackColor** drop-down button to display the color picker and choose the color from the available colors.
6. Click **OK**.
7. Click **Apply** and **OK** to apply the changes.

Using a Shortcut

Set the **BackColor ('BackColor Property' in the on-line documentation)** property or **ForeColor ('ForeColor Property' in the on-line documentation)** property for the **FpSpread ('FpSpread Class' in the on-line documentation) Cells ('Cells Class' in the on-line documentation)** object.

Example

This example code sets the background color for cell A1 to Azure, then sets the background color for cells C3 through D4 to Bisque.

C#

```
FpSpread1.Sheets[0].RowCount = 4;
FpSpread1.Sheets[0].ColumnCount = 4;
FpSpread1.Sheets[0].Cells[0,0].BackColor = Color.Azure;
FpSpread1.Sheets[0].Cells[2,2,3,3].BackColor = Color.Bisque;
```

VB

```
FpSpread1.Sheets(0).RowCount = 4
FpSpread1.Sheets(0).ColumnCount = 4
FpSpread1.Sheets(0).Cells(0, 0).BackColor = Color.Azure
FpSpread1.Sheets(0).Cells(2, 2, 3, 3).BackColor = Color.Bisque
```

Using Code

Set the **BackColor ('BackColor Property' in the on-line documentation)** property or **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.Web.Spread.SheetView count;
count = FpSpread1.Sheets[0];
count.RowCount = 4;
count.ColumnCount = 4;
FarPoint.Web.Spread.Cell cellA1;
cellA1 = FpSpread1.Cells[0, 0];
cellA1.BackColor = Color.Azure;
cellA1.ForeColor = Color.Navy;
FarPoint.Web.Spread.Cell cellrange;
```

```
cellrange = FpSpread1.Cells[2,2,3,3];
cellrange.BackColor = Color.Bisque;
```

VB

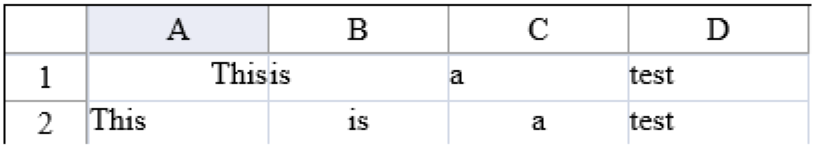
```
Dim count as FarPoint.Web.Spread.SheetView
count = FpSpread1.Sheets(0)
count.RowCount = 4
count.ColumnCount = 4
Dim cellA1 As FarPoint.Web.Spread.Cell
cellA1 = FpSpread1.Cells(0, 0)
cellA1.BackColor = Color.Azure
cellA1.ForeColor = Color.Navy
Dim cellrange As FarPoint.Web.Spread.Cell
cellrange = FpSpread1.Cells(2, 2, 3, 3)
cellrange.BackColor = Color.Bisque
```

Using the Spread Designer

1. Select the cells to apply the changes to.
2. Select the **Home** menu and then select the **Fill Color** icon under the **Font** section.
   You can also select the **Settings** menu and then select the **Cells, Columns, and Rows** icon under the **Other Settings** section.
3. Click the **BackColor** drop-down button to display the color picker and choose the color from the available colors.
4. Click **OK** to apply the changes.
5. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Aligning Cell Contents

You can set the horizontal or vertical alignment for the contents of a cell or a group of cells.



	A	B	C	D
1	This	is	a	test
2	This	is	a	test

Using the Properties Window

1. At design time, in the **Properties** window, select the FpSpread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. Select the **Cells** collection and then select **HorizontalAlign** or **VerticalAlign** under the **Misc.** section (select the cells or cells).
5. You can also select a cell or cells and select **HorizontalAlign** or **VerticalAlign** (**Selected Item** drop-down option).
6. Click **Apply** and **OK**.

Using a Shortcut

Set the **HorizontalAlign ('HorizontalAlign Property' in the on-line documentation)** and **VerticalAlign ('VerticalAlign Property' in the on-line documentation)** properties for the **FpSpread ('FpSpread Class' in the on-line documentation)** object's **Cells ('Cells Class' in the on-line documentation)** property.

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 top-aligned, and the horizontal alignment of cells from B2 to C3 to be centered.

C#

```
FpSpread1.Sheets[0].Cells[0,0].HorizontalAlign = HorizontalAlign.Right;
FpSpread1.Sheets[0].Cells[0,0].VerticalAlign = VerticalAlign.Top;
FpSpread1.Sheets[0].Cells[1,1,2,2].HorizontalAlign = HorizontalAlign.Center;
```

VB

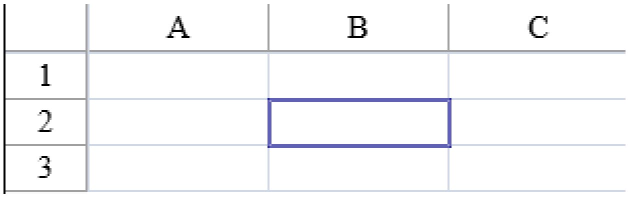
```
FpSpread1.Sheets(0).Cells(0,0).HorizontalAlign = HorizontalAlign.Right
FpSpread1.Sheets(0).Cells(0,0).VerticalAlign = VerticalAlign.Top
FpSpread1.Sheets(0).Cells(1,1,2,2).HorizontalAlign = HorizontalAlign.Center
```

Using the Spread Designer

1. Select the **Home** menu.
2. Select the cells you wish to change.
3. Select the appropriate icon in the **Alignment** section.
4. Click **Apply** to apply the changes.
5. 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 borders of the cell. You can specify whether a cell or range of cells has a border. Borders can be displayed on the left, right, top, or bottom, or around all four sides of the cell or cell range.



Cell borders follow the precedence used by the sheet. For more information on precedence, refer to the list in **Object Parentage**.

If you import cell border information from an Excel file, the width of the cell border may be changed. If the **EnableClientScript ('EnableClientScript Property' in the on-line documentation)** property of the FpSpread component is set to True, then BorderCollapse is set to True. This helps the appearance of most borders and grid lines together. However, if a cell border is set to a size equal to 1, the left and top borders are overwritten by the grid lines due to the behavior of HTML tables. So the solid, dash, dot, etc. borders are set to a width of 2 so that they are displayed when loaded into Spread from Excel. The "hair" border is not set to a size of 1.

Set border color, style, and size with strict compliance mode to get the best results.

Borders are different from grid lines in that they create a border around a cell or group of cells rather than distinguishing rows and columns. For more information about grid lines, which are set for an entire sheet, refer to **Displaying Grid Lines on the Sheet**.

For information about cascading style sheets, refer to **Assigning a Cascading Style Sheet to a Cell**.

Using a Shortcut

Set the **Border ('Border Property' in the on-line documentation)** property to the new border that you specify.

Example

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

C#

```
// Set cell border to the bevel border.
fpSpread1.Sheets[0].Cells[2, 2].Border = new
FarPoint.Web.Spread.Border(System.Web.UI.WebControls.BorderStyle. Double,
Color.DarkBlue, 2);
```

VB

```
' Set cell border to the bevel border.
FpSpread1.Sheets(0).Cells(2, 2).Border = New
FarPoint.Web.Spread.Border(System.Web.UI.WebControls.BorderStyle. Double,
Color.DarkBlue, 2)
```

Using the Spread Designer

There are several ways to set cell borders in the Spread Designer. The first method is the following:

1. Select the cells to apply the changes to.
2. Select the **Home** menu and then select the **Cell Border** icon under the **Font** section.
3. Select a border type or select more borders to set colors and other border properties.
4. Click **OK** to apply the changes.

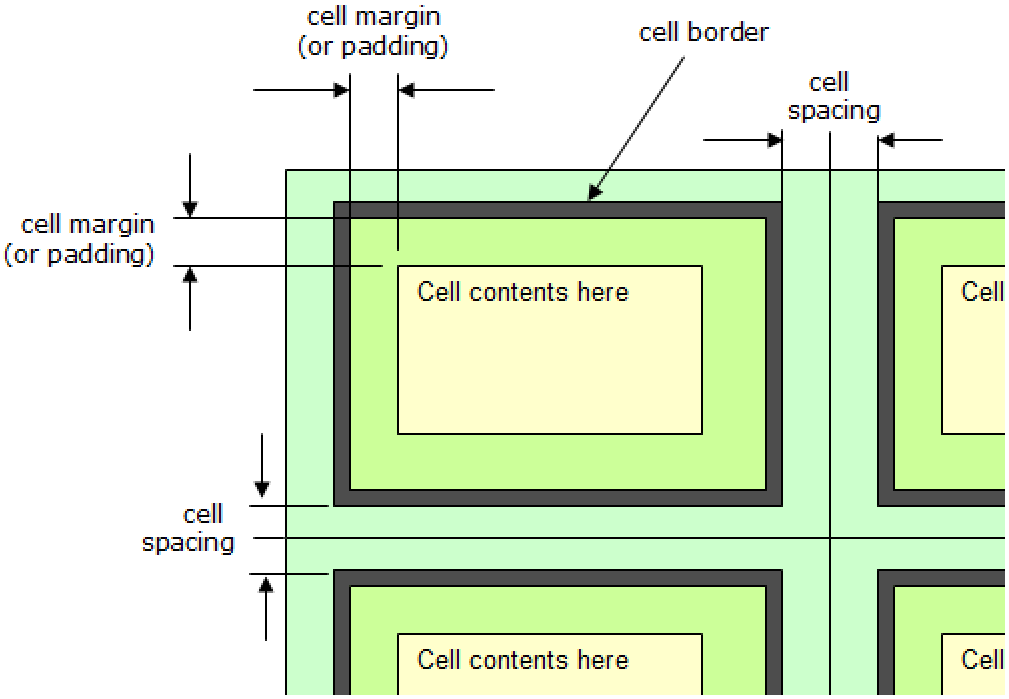
Using the Spread Designer

This is the second method:

1. Select the cells to apply the changes to.
2. Select the **Settings** menu and then select the **Cells, Columns, and Rows** icon under the **Other Settings** section.
3. Click the **Border** property in the property grid to set the border options.
4. Click **OK** to apply the changes.
5. Click **Apply and Exit** to close the Spread Designer.

Customizing the Margins and Spacing of the Cell

You can customize the margins within a cell and the spacing between cells in a sheet. The cell margin is the distance between the cell border and the cell contents and is specified for all four sides of a cell (or table cell in the displayed HTML). The cell spacing is the distance between the cells and is specified for the entire sheet (or table in the displayed HTML).



To set the cell margin (or padding) use the **Margin ('Margin Property' in the on-line documentation)** property in the **Cell ('Cell Class' in the on-line documentation)** (or **Column ('Column Class' in the on-line documentation)** or **Row ('Row Class' in the on-line documentation)**) class, or set the **Margin ('Margin Property' in the on-line documentation)** property in the **StyleInfo ('StyleInfo Class' in the on-line documentation)** class and apply the style to the cell or cells.

To set the cell spacing for the entire sheet, use the **CellSpacing ('CellSpacing Property' in the on-line documentation)** property in the Sheet or set the **CellSpacing ('CellSpacing Property' in the on-line documentation)** property in the **SheetSkin ('SheetSkin Class' in the on-line documentation)** class and apply the skin to the sheet or sheets.

To set the border, use the **Border ('Border Property' in the on-line documentation)** property as described in **Customizing Cell Borders**.

Using a Shortcut

1. Set the Cell shortcut object **Margin ('Margin Property' in the on-line documentation)** property to the new margins.
2. Set the Sheet **CellSpacing ('CellSpacing Property' in the on-line documentation)** property to the new cell spacing value.

Example

This example code creates a cell margin and sets cell spacing for the sheet.

C#

```
FarPoint.Web.Spread.Cell mycell;
FarPoint.Web.Spread.Inset margin = new FarPoint.Web.Spread.Inset(20, 40, 50, 20);
mycell = FpSpread1.Cells[0, 0];
mycell.Value = "Margin";
mycell.Locked = true;
mycell.Margin = margin;
```

```
FpSpread1.ActiveSheetView.Rows[0].Height = 80;
FpSpread1.Sheets[0].CellSpacing = 5;
```

VB

```
Dim mycell As FarPoint.Web.Spread.Cell
Dim margin As New FarPoint.Web.Spread.Inset(20, 40, 50, 20)
mycell = FpSpread1.Cells(0, 0)
mycell.Value = "Margin"
mycell.Locked = True
mycell.Margin = margin
FpSpread1.ActiveSheetView.Rows(0).Height = 80
FpSpread1.Sheets(0).CellSpacing = 5
```

Using the Spread Designer

1. Select the **Settings** menu.
2. Select the cells to apply the changes to.
3. Use the following to set the margin:
   Select the **Cells, Columns, and Rows** icon under the **Other Settings** section.

   Click on the **Margin** property in the property grid to create an inset for the cell in pixel units.

   Click **OK** to apply the changes.
4. To apply the cell spacing, do the following:
   Select the **Settings** menu.

   Select the **SheetSkin** icon in the **Appearance Settings** section.

   Select the custom tab in the **SheetSkin** editor.

   Set **CellSpacing** in the **Misc.** section.

   Click **OK** to apply the changes.
5. Click **Apply and Exit** to close the Spread Designer.

Creating and Applying a Custom Style for Cells

You can quickly customize the appearance of a cell or range of cells (or rows or columns) by applying a style, which is a set of appearance settings defined in a single **StyleInfo ('StyleInfo Class' in the on-line documentation)** object. You can create your own style and save it to use again, similar to a template. The style includes appearance settings that apply to cells, such as background color, text color, font, borders, and cell type. For more information, refer to the **StyleInfo ('StyleInfo Class' in the on-line documentation)** object in the **Assembly Reference (on-line documentation)**.

Note: The word "appearance" is used to refer to the general look and feel of the cell, not the Appearance class, which is used for the appearance of several parts of the interface.

A cell style includes the following appearance settings:

- text alignment of cell contents
- border colors, border style, border width and which sides have them
- cell colors
- font for text in cell
- margins
- cell type

You can specify the style for a cell by setting the **StyleName ('StyleName Property' in the on-line documentation)** property for the **Cell ('Cell Class' in the on-line documentation)** object.

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. You typically set the style for the cell by using the **StyleName ('StyleName Property' in the on-line documentation)** property to define which **StyleInfo ('StyleInfo Class' in the on-line documentation)** object is used by that cell. When you set the style (**StyleName ('StyleName Property' in the on-line documentation)**), 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 with the specified name.

You can also use the **ParentStyleName ('ParentStyleName Property' in the on-line documentation)** property to set a style for a range of cells that may individually have different **StyleName ('StyleName Property' in the on-line documentation)** values set but which all inherit a common set of appearance settings from a parent style. A cell inherits all the style information from the parent style (**ParentStyleName ('ParentStyleName Property' in the on-line documentation)**). 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 ('Parent Property' in the on-line documentation)** property of the **NamedStyle ('NamedStyle Class' in the on-line documentation)** object. So different cells (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). The default parent style is set in the **DataAreaDefault ('DataAreaDefault Field' in the on-line documentation)** field in the **DefaultStyleCollection ('DefaultStyleCollection Class' in the on-line documentation)** class. For more information on the order of inheritance, refer to **Object Parentage**.

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 Skin for Sheets**.

Using the Property Window

1. In the **Form** window, click the FpSpread component for which you want to create the style in the NamedStyleCollection. For the FpSpread component, in the **Appearance** category, select the **NamedStyles** property.
2. Click on the button to launch the **NamedStyleCollection Editor**.
3. In the **NamedStyleCollection Editor**, click the **Add** button.
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 Code

1. Create the style using the **NamedStyle ('NamedStyle Class' in the on-line documentation)** object constructor and set the style properties.
2. Add the styles.
3. Set the **StyleName ('StyleName Property' in the on-line documentation)** property to assign the style to the cells.

Example

This example code creates several styles and sets a parent style. This causes the cells to display the same background color but, different text colors.

C#

```
\\ Create a style with a blue background color.
FarPoint.Web.Spread.NamedStyle backstyle = new
FarPoint.Web.Spread.NamedStyle("BlueBack");
backstyle.BackColor = Color.Blue;
\\ Create a style with an orange text color and assign it a parent style.
FarPoint.Web.Spread.NamedStyle text1style = new
FarPoint.Web.Spread.NamedStyle("OrangeText", "BlueBack");
text1style.ForeColor = Color.Orange;
\\ Create a style with a yellow text color and assign it a parent style.
FarPoint.Web.Spread.NamedStyle text2style = new
FarPoint.Web.Spread.NamedStyle("YellowText", "BlueBack");
text2style.ForeColor = Color.Yellow;
FpSpread1.NamedStyles.Add(backstyle);
FpSpread1.NamedStyles.Add(text1style);
FpSpread1.NamedStyles.Add(text2style);
FpSpread1.ActiveSheetView.Cells[0,0,2,0].StyleName = "OrangeText";
FpSpread1.ActiveSheetView.Cells[0,1,2,1].StyleName = "YellowText";
```

VB

```
' Create a style with a blue background color.
Dim backstyle As New FarPoint.Web.Spread.NamedStyle("BlueBack")
backstyle.BackColor = Color.Blue
' Create a style with an orange text color and assign it a parent style.
Dim text1style As New FarPoint.Web.Spread.NamedStyle("OrangeText", "BlueBack")
text1style.ForeColor = Color.Orange
' Create a style with a yellow text color and assign it a parent style.
Dim text2style As New FarPoint.Web.Spread.NamedStyle("YellowText", "BlueBack")
text2style.ForeColor = Color.Yellow
FpSpread1.NamedStyles.Add(backstyle)
FpSpread1.NamedStyles.Add(text1style)
FpSpread1.NamedStyles.Add(text2style)
FpSpread1.ActiveSheetView.Cells(0,0,2,0).StyleName = "OrangeText"
fpSpread1.ActiveSheetView.Cells(0,1,2,1).StyleName = "YellowText"
```

Using the Spread Designer

1. Select the **Settings** menu.
2. Select the **Named Style** icon under the **Appearance Settings** section.
3. Use the **New** style icon to create a new style and use the **Edit** style icon to set properties for the style.
4. Select **Apply** and **OK**.
5. Select the cell or cells and set the **StyleName** property.
6. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Assigning a Cascading Style Sheet to a Cell

You can assign a cascading style sheet (CSS) to a cell or group of cells as a way of conveniently defining the appearance settings of the cell or cells. The CSS class name is a property of the cell type.

This assumes that the **RenderCSSClass ('RenderCSSClass Property' in the on-line documentation)** property in the **FpSpread ('FpSpread Class' in the on-line documentation)** class is set to True, which is the default. If it is set to False, the component uses the in-line style attributes instead of the cascading style sheet.

The CSS Class should be outside the head tag and after the Spread HTML code for best results with strict compliance.

Using Code

1. Define the **CSSClass ('CssClass Property' in the on-line documentation)** property for a given cell type.
2. Assign that cell type to the cell or cells.

Example

The following attaches a style sheet named myCssClass to be used for the specified column header and cells:

C#

```
FarPoint.Web.Spread.GeneralCellType mycelltype = new
FarPoint.Web.Spread.GeneralCellType();
myCellType.CssClass = "myCssClass";
FpSpread1.ColumnHeader.Cells[0, 0].CellType = myCellType;
FpSpread1.Cells[0, 1].CellType = myCellType;
```

VB

```
Dim myCellType As New FarPoint.Web.Spread.GeneralCellType
myCellType.CssClass = "myCssClass"
FpSpread1.ColumnHeader.Cells(0, 0).CellType = myCellType
FpSpread1.Cells(0, 1).CellType = myCellType
```

Example

When the pointer is over data in the Spread, the mouse cursor turns into an I-beam shape indicating that the user is allowed to edit the information. This happens even if the columns are locked. To have the pointer remain an arrow when hovering over data in the row, or to change the cursor to whatever you want, you can use the CSSClass object, which assigns a cascading style sheet class to a cell. For example, you can use the following code to use a style for the cursor over a locked column.

C#

```
FpSpread1.Sheets[0].Columns[0].Locked = true;
CType(FpSpread1.Sheets[0].StyleModel.GetCompositeInfo(-1, 0, -1, Nothing).CellType,
FarPoint.Web.Spread.GeneralCellType).CssClass = "myCell";
```

VB

```
FpSpread1.Sheets(0).Columns(0).Locked = True
CType(FpSpread1.Sheets(0).StyleModel.GetCompositeInfo(-1, 0, -1, Nothing).CellType,
FarPoint.Web.Spread.GeneralCellType).CssClass = "myCell"
```

Then in HTML code you can add the myCell style:

HTML

```
<style>.myCell { CURSOR: default } </style>
```

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.

Using the Properties Window

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

2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. Select the sheet.
5. Select the button for the **Cells** object (or **Columns** or **Rows** collections).
6. Select a block of cells.
7. Set properties as needed.
8. Click **OK** to close each editor.

Using Code

1. Specify the range of cells.
2. Set the **Note ('Note Property' in the on-line documentation)** property for the **Cell ('Cell Class' in the on-line documentation)** object for that range.

Example

This example code sets the **Note** property for a range of **Cell** objects.

C#

```
FarPoint.Web.Spread.Cell range1;
range1 = fpSpread1.ActiveSheetView.Cells[0, 0, 2, 2];
range1.Value = "Value Here";
range1.Note = "This is the note that describes the value.";
```

VB

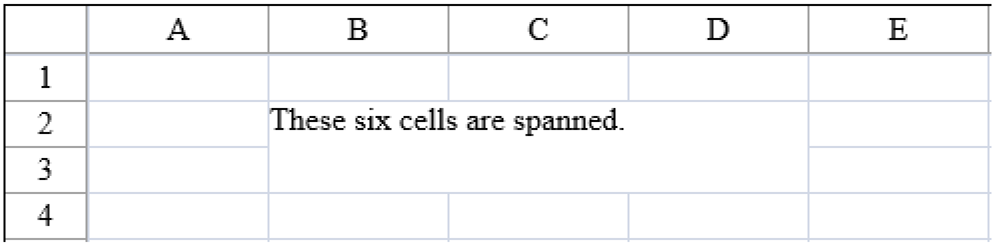
```
Dim range1 As FarPoint.Web.Spread.Cell
range1 = fpSpread1.ActiveSheetView.Cells(0, 0, 2, 2)
range1.Value = "Value Here"
range1.Note = "This is the note that describes the value."
```

Using the Spread Designer

1. Select a block of cells.
2. Set properties as needed.
3. From the **File** menu choose **Apply and Exit** to apply your changes to the component and exit Spread Designer.

Spanning Cells

You can group cells together to form one large cell. This is called a spanning cells and the large cell that is created is called a cell span. You can add spans in headers or in data cells. Creating a span of cells creates one large cell where there had previously been several. For example, if you create a span of cells from cell B2 to cell D3, cell B2 then appears to occupy the space from cell B2 through cell D3.



	A	B	C	D	E
1					
2		These six cells are spanned.			
3					
4					

The component is divided into four parts: sheet corner, column headers, rows headers, and data area. You can create spans within a part, but you cannot create a span that goes across parts. For example, you cannot span cells in the data area with cells in the row headers and you cannot 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 (called 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.

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 by calling 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: You cannot sort a spreadsheet that has spanned cells.

For more information on allowing automatic merging of cells with identical content, refer to **Allowing Cells to Merge Automatically**.

For more information about the underlying span model, refer to **Understanding the Span Model**.

Using the Properties Window

1. At design time, in the **Properties** window, select the FpSpread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. Select the sheet.
5. In the **Properties** window on the right side of the **SheetView Collection Editor**, select the **Cells** property for the sheet.
6. Select the cell from which to start the span.
7. Click the button to display the **Cell, Column, and Row Editor**.
8. In the editor, select either the **RowSpan** or **ColumnSpan** 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.
9. If you want to apply this change, click **Apply**.
10. Click **OK** to close each editor.

Using a Shortcut

To span cells (or remove spanning) use any of the following methods of the **FpSpread ('FpSpread Class' in the on-line documentation)** component **Sheets ('Sheets Property' in the on-line documentation)** shortcut:

- **AddSpanCell ('AddSpanCell Method' in the on-line documentation)**
- **GetSpanCell ('GetSpanCell Method' in the on-line documentation)**
- **RemoveSpanCell ('RemoveSpanCell Method' in the on-line documentation)**

Example

This example code defines some content then spans six adjoining cells.

C#

```
// Create some content in two cells.
FpSpread1.ActiveSheetView.Cells[1,1].Text = "These six cells are spanned.";
FpSpread1.ActiveSheetView.Cells[2,2].Text = "This is text in 2,2.";
// Span six cells including the ones with different content.
FpSpread1.ActiveSheetView.AddSpanCell(1, 1, 2, 3);
```

VB

```
' Create some content in two cells.
FpSpread1.ActiveSheetView.Cells(1,1).Text = "These six cells are spanned."
FpSpread1.ActiveSheetView.Cells(2,2).Text = "This is text in 2,2."
' Span six cells including the ones with different content.
FpSpread1.ActiveSheetView.AddSpanCell(1, 1, 2, 3)
```

Using the Spread Designer

1. Select the sheet.
2. Select the cells to span.
3. Do one of the following:
   From the **Home** menu, select the merge icon.

   Another way is to right-click and select **Span** or in the property list (in the **Misc** category), select either the **RowSpan** or **ColumnSpan** property and set the number to a value greater than one to span cells. To remove a span, set the value back to 1.

   The Designer shows the cells spanned.
4. 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.

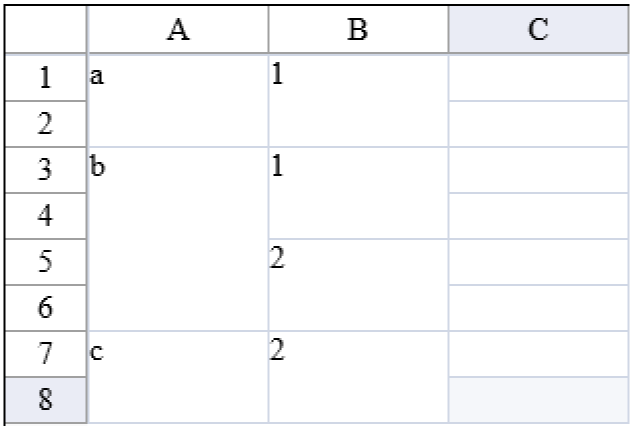
Unlike spanning cells, merging is an automatic feature. You tell the component which columns and rows allow cells to be combined automatically, and any cells within that set that have the same contents are combined for you.

For more information on spanning cells, refer to **Spanning Cells**.

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 the merge policy for column B is Always, the cells in column B are merged into two blocks B1:B4 and B5:B8. If the merge policy for column A is Always and the merge policy for column B is Restricted then the cells in column B are merged into four blocks B1:B2, B3:B4, B5:B6, and B7:B8. For example:



	A	B	C
1	a	1	
2			
3	b	1	
4			
5		2	
6			
7	c	2	
8			

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 the top-left merged cell; when you leave edit mode, if the contents of that cell are no longer identical to the cell or cells with which it was previously merged, the cells are no longer displayed as merged when the Spread is updated. How cells are merged is only changed when the Spread is updated.

Cells that are different cell types but have the same contents can merge. For example, a date cell might contain the contents "01/31/02" and the adjacent edit cell might contain the same contents; if the column containing the cells is set to merge, the cells will merge. If the contents change or the merge is removed, the cells maintain their cell types as well as their data.

To set cells to be merged if they have the same value, use the following members:

- **GetColumnMerge ('GetColumnMerge Method' in the on-line documentation)** and **SetColumnMerge ('SetColumnMerge Method' in the on-line documentation)**
- **GetRowMerge ('GetRowMerge Method' in the on-line documentation)** and **SetRowMerge ('SetRowMerge Method' in the on-line documentation)**
- **GetMergePolicy ('GetMergePolicy Method' in the on-line documentation)** and **SetMergePolicy ('SetMergePolicy Method' in the on-line documentation)**
- **MergePolicy ('MergePolicy Enumeration' in the on-line documentation)** enumeration settings

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 Model namespace in the **Assembly Reference (on-line documentation)**.

For more information on creating spans of cells with identical content, refer to **Spanning Cells**.

Using a Shortcut

Use the **SetColumnMerge ('SetColumnMerge Method' in the on-line documentation)** or **SetRowMerge ('SetRowMerge Method' in the on-line documentation)** method for the **FpSpread ('FpSpread Class' in the on-line documentation)** component **Sheets ('Sheets Property' in the on-line documentation)** shortcut.

Example

This example code sets the row and column merge policies for all rows and all columns.

C#

```
FpSpread1.Sheets[0].SetRowMerge(-1, FarPoint.Web.Spread.Model.MergePolicy.Always);
FpSpread1.Sheets[0].SetColumnMerge(-1, FarPoint.Web.Spread.Model.MergePolicy.Always);
```

VB

```
FpSpread1.Sheets(0).SetRowMerge(-1, FarPoint.Web.Spread.Model.MergePolicy.Always)
FpSpread1.Sheets(0).SetColumnMerge(-1, FarPoint.Web.Spread.Model.MergePolicy.Always)
```

Using Code

Set the **SetColumnMerge ('SetColumnMerge Method' in the on-line documentation)** or **SetRowMerge ('SetRowMerge Method' in the on-line documentation)** method for a **SheetView ('SheetView Class' in the on-line documentation)** object.

Example

This example code sets the row and column merge policies for all rows and all columns.

C#

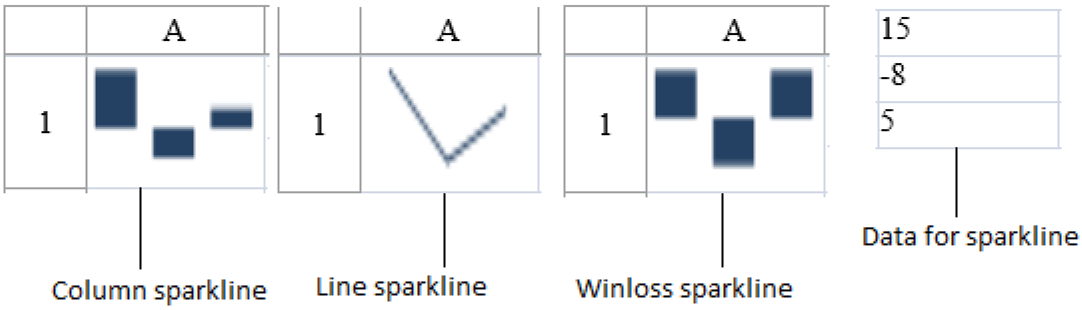
```
FarPoint.Web.Spread.SheetView Sheet0;
Sheet0 = fpSpread1.Sheets[0];
Sheet0.SetRowMerge(-1, FarPoint.Web.Spread.Model.MergePolicy.Always);
Sheet0.SetColumnMerge(-1, FarPoint.Web.Spread.Model.MergePolicy.Always);
```

VB

```
Dim Sheet0 As FarPoint.Web.Spread.SheetView
Sheet0 = fpSpread1.Sheets(0)
Sheet0.SetRowMerge(-1, FarPoint.Web.Spread.Model.MergePolicy.Always)
Sheet0.SetColumnMerge(-1, FarPoint.Web.Spread.Model.MergePolicy.Always)
```

Using Sparklines

You can create a small graph in a cell 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. The 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.

The Sparkline graph requires the following information in the web.config file. This example is based on IIS7.

Code

```
<system.webServer>
<validation validateIntegratedModeConfiguration="false"/>
<handlers>
...
<add name="chart" path="FpChart.axd" verb="*"
type="FarPoint.Web.Chart.ChartImageHttpHandler"/>
</handlers>
// If you are using integrated managed pipeline mode,
//set validateIntegratedModeConfiguration to false.
<validation validateIntegratedModeConfiguration="false"/>
```

This example is based on IIS8.

Code

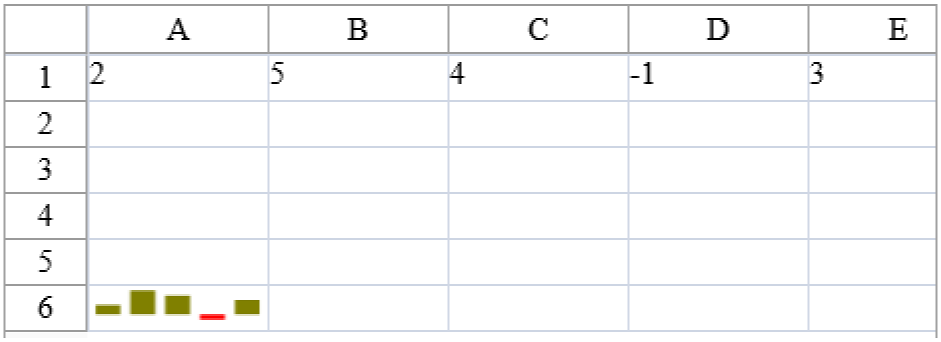
```
<system.webServer>
    <handlers>
      <add name="FpChart" verb="*" path="FpChart.axd" preCondition="integratedMode"
type="FarPoint.Web.Chart.ChartImageHttpHandler"/>
    </handlers>
    <validation validateIntegratedModeConfiguration="false"/>
  </system.webServer>
```

For more information, see the following topics:

- **Adding a Sparkline to a Cell**
- **Customizing Markers and Pointers**
- **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 to create the sparkline in.
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 with the **AddSparkline ('AddSparkline Method' in the on-line documentation)** method.

Example

This example creates a column sparkline in a cell and shows negative and series colors.

C#

```
FarPoint.Web.Spread.SheetView sv = new FarPoint.Web.Spread.SheetView();
FarPoint.Web.Spread.Chart.SheetCellRange data = new
FarPoint.Web.Spread.Chart.SheetCellRange(sv, 0,0,1, 5);
FarPoint.Web.Spread.Chart.SheetCellRange data2 = new
FarPoint.Web.Spread.Chart.SheetCellRange(sv, 5,0,1,1);
FarPoint.Web.Spread.ExcelSparklineSetting ex = new
FarPoint.Web.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;
fpSpread1.Sheets[0].RowCount = 6;
fpSpread1.Sheets[0].ColumnCount = 6;
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.Web.Spread.SparklineType.Column,
ex);
```

VB

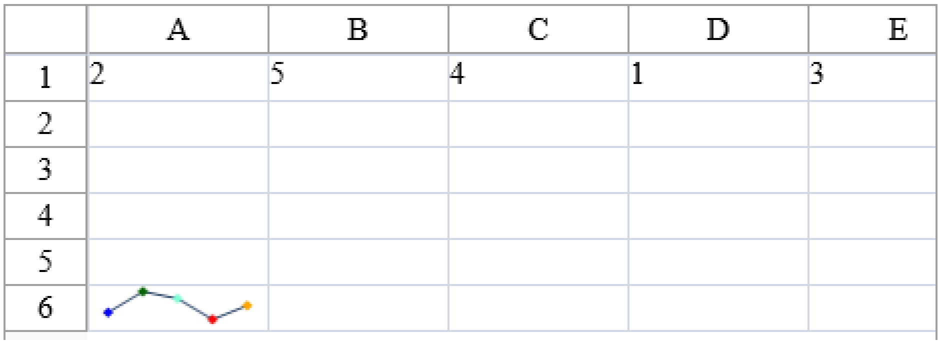
```
Dim sv As New FarPoint.Web.Spread.SheetView()
Dim data As New FarPoint.Web.Spread.Chart.SheetCellRange(sv, 0, 0, 1, 5)
Dim data2 As New FarPoint.Web.Spread.Chart.SheetCellRange(sv, 5, 0, 1, 1)
Dim ex As New FarPoint.Web.Spread.ExcelSparklineSetting()
ex.ShowMarkers = True
ex.ShowNegative = True
ex.NegativeColor = Color.Red
' Use with a Column or Winloss type
ex.SeriesColor = Color.Olive
FpSpread1.Sheets(0) = sv
FpSpread1.Sheets(0).RowCount = 6
FpSpread1.Sheets(0).ColumnCount = 6
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.Web.Spread.SparklineType.Column,
ex)
```

Using the Spread Designer

1. Type data in a cell or a column or row of cells in the designer.
2. Select a cell for the sparkline.
3. Select the **Insert** menu.
4. Select a sparkline type.
5. Set the **Data Range** in the **Create Sparklines** dialog (such as =Sheet1!$E$1:$E$3).
6. Select **OK.**
7. Select **Apply and Exit** from the **File** menu to save your changes and close the designer.

Customizing Markers and Pointers

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 ('SeriesColor Property' in the on-line documentation)** property applies to the line for the line spark type. The **MarkersColor ('MarkersColor Property' in the on-line documentation)** 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 ('ShowFirst Property' in the on-line documentation)** and **FirstMarkerColor ('FirstMarkerColor Property' in the on-line documentation)**).
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.Web.Spread.SheetView sv = new FarPoint.Web.Spread.SheetView();
FarPoint.Web.Spread.Chart.SheetCellRange data = new
FarPoint.Web.Spread.Chart.SheetCellRange(sv, 0,0,1, 5);
FarPoint.Web.Spread.Chart.SheetCellRange data2 = new
FarPoint.Web.Spread.Chart.SheetCellRange(sv, 5,0,1,1);
FarPoint.Web.Spread.ExcelSparklineSetting ex = new
FarPoint.Web.Spread.ExcelSparklineSetting();
```

```
ex.AxisColor = Color.SaddleBrown;
ex.ShowNegative = true;
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;
fpSpread1.Sheets[0].RowCount = 6;
fpSpread1.Sheets[0].ColumnCount = 6;
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.Web.Spread.SparklineType.Line,
ex);
```

VB

```
Dim sv As New FarPoint.Web.Spread.SheetView()
Dim data As New FarPoint.Web.Spread.Chart.SheetCellRange(sv, 0, 0, 1, 5)
Dim data2 As New FarPoint.Web.Spread.Chart.SheetCellRange(sv, 5, 0, 1, 1)
Dim ex As New FarPoint.Web.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
FpSpread1.Sheets(0).RowCount = 6
FpSpread1.Sheets(0).ColumnCount = 6
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.Web.Spread.SparklineType.Line,
ex)
```

Using the Spread Designer

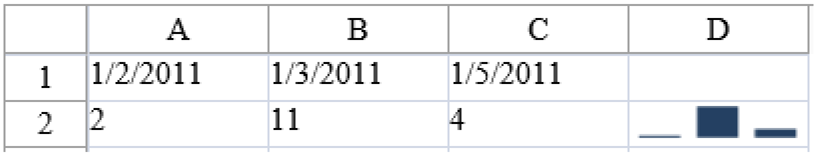
1. Type data in a cell or a column or row of cells in the designer.
2. Select a cell for the sparkline.
3. Select the **Insert** menu.

4. Select a sparkline type.
5. Set the **Data Range** in the **Create Sparklines** dialog (such as =Sheet1!$E$1:$E$3).
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 type. The general type specifies that all the points are painted along the axis at the same distance.

Set the **DisplayXAxis ('DisplayXAxis Property' in the on-line documentation)** property if you wish to display the axis line. The following image displays all three points on the graph.



	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 ('ManualMax Property' in the on-line documentation),ManualMin ('ManualMin Property' in the on-line documentation)**, **MaxAxisType ('MaxAxisType Property' in the on-line documentation)**, and **MinAxisType ('MinAxisType Property' in the on-line documentation)** properties in the **ExcelSparklineSetting ('ExcelSparklineSetting Class' in the on-line documentation)** class for vertical axis examples.

Using Code

1. Create an **ExcelSparklineSetting ('ExcelSparklineSetting Class' in the on-line documentation)** object.
2. Set the **DisplayXAxis ('DisplayXAxis Property' in the on-line documentation)** property to true if you wish to display the axis.
3. Add values and dates to the cells.
4. Add the sparkline to the cell.

Example

This example creates a column sparkline in a cell with a horizontal axis.

C#

```
FarPoint.Web.Spread.ExcelSparklineSetting ex = new
FarPoint.Web.Spread.ExcelSparklineSetting();
ex.DisplayXAxis = true;
ex.Formula = "Sheet1!$A$1:$C$1";
fpSpread1.Sheets[0].RowCount = 4;
fpSpread1.Sheets[0].ColumnCount = 4;
```

```
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.Web.Spread.SparklineType.Column, ex);
```

VB

```
Dim ex As New FarPoint.Web.Spread.ExcelSparklineSetting()
ex.DisplayXAxis = True
ex.Formula = "Sheet1!$A$1:$C$1"
FpSpread1.Sheets(0).RowCount = 4
FpSpread1.Sheets(0).ColumnCount = 4
FpSpread1.Sheets(0).Cells(0, 0).Text = "1/2/2011"
FpSpread1.Sheets(0).Cells(0, 1).Text = "1/3/2011"
FpSpread1.Sheets(0).Cells(0, 2).Text = "1/5/2011"
FpSpread1.Sheets(0).Cells(1, 0).Value = 2
FpSpread1.Sheets(0).Cells(1, 1).Value = 11
FpSpread1.Sheets(0).Cells(1, 2).Value = 4
FpSpread1.Sheets(0).AddSparkline("Sheet1!$A$2:$C$2", "Sheet1!$D$2:$D$2",
FarPoint.Web.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. Set the **Data Range** in the **Create Sparklines** dialog (such as =Sheet1!$E$1:$E$3).
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, clear, 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.

You can clear a sparkline in code with the **ClearSparklines ('ClearSparklines Method' in the on-line documentation)** method in the **SheetView ('SheetView Class' in the on-line documentation)** class.

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 ('SheetView Class' in the on-line documentation)** 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#

```
fpSpread1.Sheets[0].GroupSparkline(new FarPoint.Web.Spread.Model.CellRange[] { new
FarPoint.Web.Spread.Model.CellRange(5, 0, 3, 1) });
// fpSpread1.Sheets[0].UnGroupSparkline(new FarPoint.Web.Spread.Model.CellRange[] { new
FarPoint.Web.Spread.Model.CellRange(5, 0, 3, 1) });
```

VB

```
FpSpread1.Sheets(0).GroupSparkline(New FarPoint.Web.Spread.Model.CellRange() {New
FarPoint.Web.Spread.Model.CellRange(5, 0, 3, 1)})
'FpSpread1.Sheets(0).UnGroupSparkline(New FarPoint.Web.Spread.Model.CellRange() {New
FarPoint.Web.Spread.Model.CellRange(5, 0, 3, 1)})
```

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 with Cell Types

Cell types define the type of information that appears in a cell, and 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.

- **Understanding How Cell Types Work**
    - **Understanding Cell Type Basics**
    - **Understanding How Cell Types Display Data**
    - **Understanding How Cell Type Affects Model Data**
    - **Determining the Cell Type of a Cell**
- **Working with Editable Cell Types**
    - **Setting a Currency Cell**
    - **Limiting Values for a Currency Cell**
    - **Setting a Date-Time Cell**
    - **Setting a Double Cell**
    - **Setting a General Cell**
    - **Setting an Integer Cell**
    - **Setting a Percent Cell**
    - **Setting a Regular Expression Cell**
    - **Setting a Text Cell**
- **Working with Graphical Cell Types**
    - **Setting a Button Cell**
    - **Setting a Check Box Cell**
    - **Setting a Combo Box Cell**
    - **Setting a Hyperlink Cell**
    - **Setting an Image Cell**
    - **Setting a Label Cell**
    - **Setting a List Box Cell**
    - **Setting a Multiple-Column Combo Box Cell**
    - **Setting a Radio Button List Cell**
    - **Setting a Tag Cloud Cell**
- **Working with ASP.NET AJAX Extender Cell Types**
    - **Setting an Automatic-Completion Cell**
    - **Setting a Combo Box Cell**
    - **Setting a Calendar Cell**
    - **Setting a Filtered Text Cell**
    - **Setting a Masked Edit Cell**
    - **Setting a Mutually Exclusive Check Box Cell**
    - **Setting a Numeric Spin Cell**
    - **Setting a Rating Cell**
    - **Setting a Slider Cell**
    - **Setting a Slide Show Cell**
    - **Setting a Text Box with Watermark Cell**
- **Using Validation Controls**

For other ways to set the appearance of the cell, refer to **Customizing the Appearance of a Cell**. For other ways to set the user interaction at the cell level, refer to **Customizing Interaction with Cells**.

For information on automatic assignment of cell types for bound data, refer to **Setting the Cell Types for Bound**

Data.

For information on the various cell type classes, refer to the cell type classes in the **Assembly Reference (on-line documentation)**.

Understanding How Cell Types Work

These topics describe how cell types work:

- **Understanding Cell Type Basics**
- **Understanding How Cell Types Display Data**
- **Understanding How Cell Type Affects Model Data**
- **Determining the Cell Type of a Cell**

Understanding Cell Type Basics

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.

Editor, Formatter, and Renderer

A cell type consists of an editor, a renderer, and a formatter. 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 editor is responsible for creating and managing the cell's edit control when in edit mode. The formatter decides how the displayed text appears. 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 is code that paints that control inside the cell rectangle when the editor is not there (paints the cell when 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. 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.

BaseCellType

The design of cell editing requires that the cell type return an editor control which 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). The **BaseCellType ('BaseCellType 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, etc.) are derived. The class can also be used to derive custom cell types that are text based. 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; header cells are renderable but are not editable. 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 act as your header cells.

Understanding How Cell Types Display Data

The cell type affects how the contents of the cell are displayed. The cell contains formatted and unformatted data.

The **Text ('Text Property' in the on-line documentation)** property contains the formatted data which is displayed in the cell; the **Value ('Value Property' in the on-line documentation)** property contains the unformatted data which is saved in the model. You can use the SheetView **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. For more information about data methods and properties, refer to **Placing and Retrieving Data**.

The following table lists the editable cell types, and how each cell type works with the data, whether formatted (**Text ('Text Property' in the on-line documentation)** property) or unformatted (**Value ('Value Property' in the on-line documentation)** property).

Editable Cell Type	Sample Input	Resultant (Text) Formatted Data	Resultant (Value) 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
DoubleCellType ('DoubleCellType Class' in the on-line documentation)	"10000.00"	"10000.00"	10000.00
GeneralCellType ('GeneralCellType Class' in the on-line documentation)	Any text	String of that data	Depends on whether DateTime, Boolean, or Text returns the DateTime object, the Boolean value, or the Text value
IntegerCellType ('IntegerCellType Class' in the on-line documentation)	"12345"	"12345"	12345
PercentCellType ('PercentCellType Class' in the on-line documentation)	"15%"	"15%"	0.15
RegExpCellType ('RegExpCellType Class' in the on-line documentation) (Regular Expression)	"123-45-6789"	"123-45-6789"	"123-45-6789"
TextCellType ('TextCellType Class' in the on-line documentation)	Any text	String of that text	String of that text

The DateTime cell returns data in the format of the **FormatString ('FormatString Property' in the on-line documentation)** property setting. In this example, the format string is set to "F".

If the cell's **Multiline ('Multiline Property' in the on-line documentation)** property is set to true, you can include a line feed character to include a line break when setting text.

Numbers are converted to scientific notation if they are greater than 999,999,999,999,999 or less than -999,999,999,999,999. Low fraction numbers are also converted to scientific notation.

The following table lists the graphical cell types, and how each cell type works with the data, whether formatted (**Text ('Text Property' in the on-line documentation)** property) or unformatted (**Value ('Value Property' in the on-line documentation)** property).

Graphical Cell Type	Sample Input	Resultant (Text) Formatted Data	Resultant (Value) Unformatted Data
ButtonCellType ('ButtonCellType Class' in the on-line documentation)	null	null	null
CheckBoxCellType ('CheckBoxCellType Class' in the on-line documentation)	True (checked)	"True"	1
	False (unchecked)	"False"	0
	Not set; looks false	Empty string	Null
ComboBoxCellType ('ComboBoxCellType Class' in the on-line documentation)	Any item	Text of selected item	Text of selected item or index of selected item
	No item selected	Empty string	Null
HyperLinkCellType ('HyperLinkCellType Class' in the on-line documentation)	Any text	String of that text	"True" (for visited link) or "False" (for not visited)
ImageCellType ('ImageCellType Class' in the on-line documentation)			
LabelCellType ('LabelCellType Class' in the on-line documentation)	Any text	String of that text	String of that text
ListBoxCellType ('ListBoxCellType Class' in the on-line documentation)	Any item selected	Text of the selected item	Text of or index of selected item
MultiColumnComboBoxCellType ('MultiColumnComboBoxCellType Class' in the on-line documentation)	Any item	Text of selected item	Text of selected item or index of selected item
RadioButtonListCellType ('RadioButtonListCellType Class' in the on-line documentation)	Any item selected	Text of the selected item	Text of or index of selected item
	No item selected	Empty string	Null

Understanding How Cell Type Affects Model Data

The cell type has two methods that control how data is displayed from the data model and how data is written to the data model. The formatter method takes the data from the data model and formats it for display. The parser method writes the data to the data model. Different cell types write different types of data to the data model.

The following table lists the editable cell types, and how each cell type works with the 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	Date-Time Object

documentation)	
DoubleCellType ('DoubleCellType Class' in the on-line documentation)	Double
GeneralCellType ('GeneralCellType Class' in the on-line documentation)	Depends whether Date-Time, Boolean, or String
IntegerCellType ('IntegerCellType Class' in the on-line documentation)	Integer
PercentCellType ('PercentCellType Class' in the on-line documentation)	String
RegExpCellType ('RegExpCellType 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 how each cell type works with the model.

Graphical Cell Type	Data Type Written to Model
ButtonCellType ('ButtonCellType Class' in the on-line documentation)	Boolean
CheckBoxCellType ('CheckBoxCellType Class' in the on-line documentation)	Integer
ComboBoxCellType ('ComboBoxCellType 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)	String
ImageCellType ('ImageCellType Class' in the on-line documentation)	Null
LabelCellType ('LabelCellType Class' in the on-line documentation)	String
MultiColumnComboBoxCellType ('MultiColumnComboBoxCellType Class' in the on-line documentation)	Type of the selected value
RadioButtonListCellType ('RadioButtonListCellType Class' in the on-line documentation)	Depends on value of EditorValue property. String, if EditorValue = String or item data, Integer, if EditorValue = index
TagCloudCellType ('TagCloudCellType Class' in the on-line documentation)	Null

Determining the Cell Type of a Cell

You can determine the cell type of a 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.

Using Code

You can use the **GetCellType ('GetCellType Method' in the on-line documentation)** method to get the cell type.

Example

The following code uses the **GetCellType ('GetCellType Method' in the on-line documentation)** method.

C#

```
FarPoint.Web.Spread.SheetView sv = FpSpread1.ActiveSheetView;
sv.Cells[0,0].CellType=new FarPoint.Web.Spread.ButtonCellType();
ListBox1.Items.Add(Convert.ToString(sv.GetCellType(0,0)));
```

VB

```
Dim sv As FarPoint.Web.Spread.SheetView
sv = FpSpread1.ActiveSheetView
sv.Cells(0,0).CellType=New FarPoint.Web.Spread.ButtonCellType()
ListBox1.Items.Add(Convert.ToString(sv.GetCellType(0,0)))
```

Working with Editable Cell Types

There are a set of tasks that relate to working with editable cell types in the spreadsheet. These include:

- **Setting a Currency Cell**
- **Limiting Values for a Currency Cell**
- **Setting a Date-Time Cell**
- **Displaying a Calendar in a Date-Time Cell**
- **Displaying a Number Pad in Number Cells (on-line documentation)**
- **Setting a Double Cell**
- **Setting a General Cell**
- **Setting an Integer Cell**
- **Setting a Percent Cell**
- **Setting a Regular Expression Cell**
- **Setting a Text Cell**

For information on the various cell type classes, refer to the cell type classes in the **Assembly Reference (on-line documentation)**.

Setting a Currency Cell

You can use currency cells to restrict users to entering currency values and to display data as currency values. The currency cell has a default error message that is displayed if the user types an invalid value and tries to leave the cell. This message can be changed using the properties for the currency cell class.

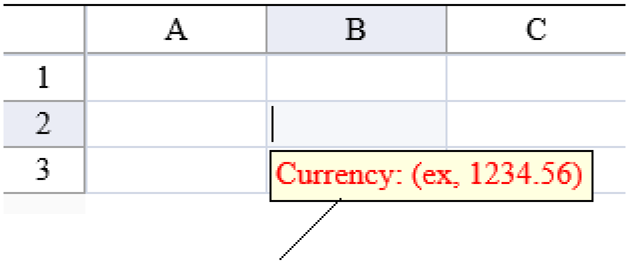
By default, Spread uses the regional Windows settings (or options) for the formatting of currency. The **CurrencyCellType ('CurrencyCellType Class' in the on-line documentation)** class does not use the **NumberFormatInfo** property inherited from the GeneralCellType. The currency cell type always uses the System.Threading.Thread.CurrentThread.CurrentCulture. These settings are:

- currency symbol (and whether to display it)
- separator character (and whether to display it)
- decimal symbol
- whether to display a leading zero
- positive currency format
- negative currency format

You can specify an edit format with the **EditMode ('EditMode Property' in the on-line documentation)** property and the **CurrencyCellType.EditModeSettings ('CurrencyCellType.EditModeSettings Class' in the on-line documentation)** class.

For details on the properties and methods for this cell type, refer to the **CurrencyCellType ('CurrencyCellType Class' in the on-line documentation)** class.

See how to define the limits for values at **Limiting Values for a Currency Cell**.



Currency cell note appears if user types invalid entry

Using Code

1. Define a currency cell type by creating an instance of the **CurrencyCellType ('CurrencyCellType Class' in the on-line documentation)** class.
2. Specify the formatting of a currency cell type.
3. Assign the currency cell type to a cell.

Example

C#

```
FarPoint.Web.Spread.CurrencyCellType currcell = new
FarPoint.Web.Spread.CurrencyCellType();
currcell.MinimumValue = 1;
FpSpread1.ActiveSheetView.Cells[1,1].CellType = currcell;
```

VB

```
Dim currcell As New FarPoint.Web.Spread.CurrencyCellType()
currcell.MinimumValue = 1
FpSpread1.ActiveSheetView.Cells(1,1).CellType = currcell
```

Using Code

1. Define a currency cell type by creating an instance of the **CurrencyCellType ('CurrencyCellType Class' in the on-line documentation)** class.
2. Specify the formatting of a currency cell type.
3. Assign the currency cell type to a cell.
4. Set the cell value.

Example

C#

```
FarPoint.Web.Spread.CurrencyCellType c = new FarPoint.Web.Spread.CurrencyCellType();
System.Globalization.NumberFormatInfo nfi = new
System.Globalization.NumberFormatInfo();
```

```
nfi.CurrencyDecimalDigits = 3;
nfi.CurrencyDecimalSeparator = ",";
nfi.CurrencySymbol = "$";
c.NumberFormat = nfi;
FpSpread1.ActiveSheetView.Cells[0, 0].CellType = c;
FpSpread1.ActiveSheetView.Cells[0, 0].Value = 234.567;
```

VB

```
Dim c As New FarPoint.Web.Spread.CurrencyCellType
Dim nfi As New System.Globalization.NumberFormatInfo
nfi.CurrencyDecimalDigits = 3
nfi.CurrencyDecimalSeparator = ","
nfi.CurrencySymbol = "$"
c.NumberFormat = nfi
FpSpread1.ActiveSheetView.Cells(0, 0).CellType = c
FpSpread1.ActiveSheetView.Cells(0, 0).Value = 234.567
```

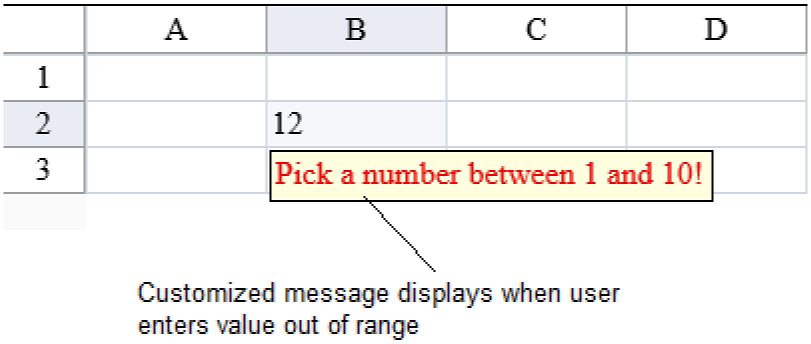
Using the Spread Designer

1. In the work area, select the cell or cells for which you want to set the cell type.
2. Select the **Home** menu.
3. Select the **SetCellType** icon under the **CellType** section.
4. Select the cell type and any other cell properties.
5. Select **OK** to close the dialog.
6. Click **Apply and Exit** to close the Spread Designer.

Limiting Values for a Currency Cell

You can set the minimum and maximum values that can be entered in a currency cell and notify the user with a message if the entry is smaller than the minimum or larger than the maximum. Use the **MinimumValue ('MinimumValue Property' in the on-line documentation)** and **MaximumValue ('MaximumValue Property' in the on-line documentation)** properties of the **CurrencyCellType ('CurrencyCellType Class' in the on-line documentation)** class to set the values.

Double, integer, and percent cells also support minimum and maximum values.



	A	B	C	D
1				
2		12		
3				

Pick a number between 1 and 10!

Customized message displays when user enters value out of range

To assign the CurrencyCellType to a cell, refer to **Setting a Currency Cell**.

Using Code

1. Create the currency cell type and set a custom error message.
2. Specify the minimum or maximum value or both of a currency cell type.

3. Assign the currency cell type to a cell.

Example

C#

```
FarPoint.Web.Spread.CurrencyCellType currcell = new
FarPoint.Web.Spread.CurrencyCellType("Pick a number between 1 and 10!");
currcell.MinimumValue = 1;
currcell.MaximumValue = 10;
FpSpread1.ActiveSheetView.Cells[1,1].CellType = currcell;
```

VB

```
Dim currcell As New FarPoint.Web.Spread.CurrencyCellType("Pick a number between 1 and
10!")
currcell.MinimumValue = 1
currcell.MaximumValue = 10
FpSpread1.ActiveSheetView.Cells(1,1).CellType = currcell
```

Using the Spread Designer

1. In the work area, select the cell or cells for which you want to set the cell type.
2. Select the **Home** menu.
3. Select the **SetCellType** icon under the **CellType** section.
4. Select the cell type and set cell properties such as **MinimumValue**, **MaximumValue**, or **ErrorMessage**.
5. Select **OK** to close the dialog.
6. Click **Apply and Exit** to close the Spread Designer.

Setting a Date-Time Cell

You can use date-time cells to restrict users to entering dates or times and to display data as date or time values.

You determine the format of the date and time to display by specifying the **DateTimeFormatInfo** object. For a complete list of date and time formats, refer to the **DateTimeCellType ('DateTimeCellType Class' in the on-line documentation)** class.

To create a time cell to display only hours and minutes, set the format string to "hh:mm".

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 ('Text Property' in the on-line documentation)** property returns "Tuesday, October 29, 2002 11:10:01 AM" as the formatted value of the cell. The **Value ('Value Property' in the on-line documentation)** property returns the date-time object.

You can specify an edit mode format with the **EditMode ('EditMode Property' in the on-line documentation)** property and the **DateTimeCellType.EditModeSettings ('DateTimeCellType.EditModeSettings Class' in the on-line documentation)** class.

For details on the properties and methods for this cell type, refer to the **DateTimeCellType ('DateTimeCellType Class' in the on-line documentation)** class.

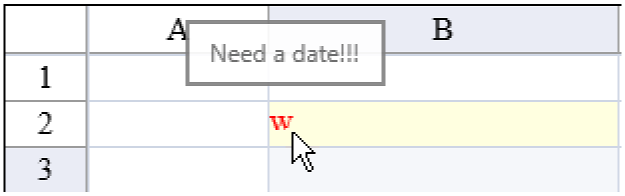
Using Code

1. Define the date-time cell type 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 using the **FormatString ('FormatString Property' in the on-line documentation)** property.
4. Assign the date-time cell type to a cell.

Example

Display the date in the format of Tuesday, March 04 (day of week, month and number of day) in the second row, second column cell. If the user inputs an invalid entry, display a note, as shown in the figure below.



C#

```
FarPoint.Web.Spread.DateTimeCellType datecell =  new
FarPoint.Web.Spread.DateTimeCellType();
datecell.ErrorMessage = "Need a date!!!";
datecell.FormatString = "dddd, MMMM d";
FpSpread1.ActiveSheetView.Columns[1].Width = 175;
FpSpread1.ActiveSheetView.Cells[1, 1].CellType = datecell;
```

VB

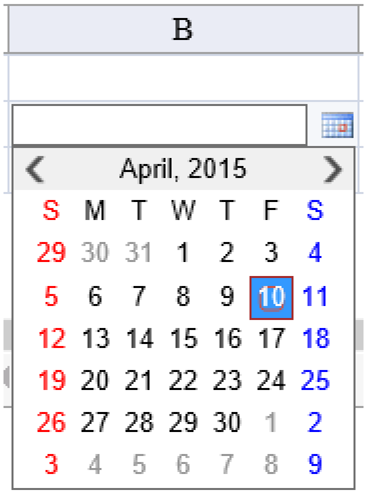
```
Dim datecell As New FarPoint.Web.Spread.DateTimeCellType()
datecell.ErrorMessage = "Need a date!!!"
datecell.FormatString = "dddd, MMMM d"
FpSpread1.ActiveSheetView.Columns(1).Width = 175
FpSpread1.ActiveSheetView.Cells(1, 1).CellType = datecell
```

Using the Spread Designer

1. In the work area, select the cell or cells for which you want to set the cell type.
2. Select the **Home** menu.
3. Select the **SetCellType** icon under the **CellType** section.
4. Select the cell type and any other cell properties.
5. Select **OK** to close the dialog.
6. Click **Apply and Exit** to close the Spread Designer.

Displaying a Calendar in a Date-Time Cell

You can display a pop-up calendar in the date-time cell as shown in the following image.



April, 2015

S	M	T	W	T	F	S
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	1	2
3	4	5	6	7	8	9

Set the **ShowPopupButton ('ShowPopupButton Property' in the on-line documentation)** property to True to display the calendar. Double-click the cell to enter edit mode and then click or tap on the drop-down button to show the calendar. Select a date and then press the Enter key to finish editing the cell.

Example

This example allows the calendar to be displayed.

C#

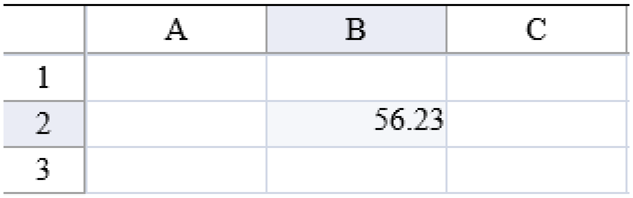
```
FarPoint.Web.Spread.DateTimeCellType datecell = new
FarPoint.Web.Spread.DateTimeCellType();
datecell.ShowPopupButton = true;
FpSpread1.ActiveSheetView.Columns[1].Width = 175;
FpSpread1.ActiveSheetView.Cells[1,1].CellType = datecell;
```

VB

```
Dim datecell As New FarPoint.Web.Spread.DateTimeCellType()
datecell.ShowPopupButton = True
FpSpread1.ActiveSheetView.Columns(1).Width = 175
FpSpread1.ActiveSheetView.Cells(1, 1).CellType = datecell
```

Setting a Double Cell

You can use double (numeric value) cells to restrict users to entering double-precision floating point numbers and to display data as these numbers. You can also specify minimum and maximum values as well as the number of decimal digits.



	A	B	C
1			
2		56.23	
3			

The double cell has a default error message that is displayed if the user types an invalid value and tries to leave the cell.

You can specify an edit mode format with the **EditMode ('EditMode Property' in the on-line documentation)** property and the **DoubleCellType.EditModeSettings ('DoubleCellType.EditModeSettings Class' in the on-**

line documentation) class.

For details on the properties and methods for this cell type, refer to the **DoubleCellType ('DoubleCellType Class' in the on-line documentation)** class.

Using Code

1. Define the double cell type by creating an instance of the **DoubleCellType ('DoubleCellType Class' in the on-line documentation)** class.
2. Assign the double cell type to a cell.

Example

This example sets a cell to a double cell type.

C#

```
FarPoint.Web.Spread.DoubleCellType dblcell = new FarPoint.Web.Spread.DoubleCellType();
FpSpread1.ActiveSheetView.Cells[1, 1].CellType = dblcell;
```

VB

```
Dim dblcell As New FarPoint.Web.Spread.DoubleCellType()
FpSpread1.ActiveSheetView.Cells(1, 1).CellType = dblcell
```

Using the Spread Designer

1. In the work area, select the cell or cells for which you want to set the cell type.
2. Select the **Home** menu.
3. Select the **SetCellType** icon under the **CellType** section.
4. Select the cell type and any other cell properties.
5. Select **OK** to close the dialog.
6. Click **Apply and Exit** to close the Spread Designer.

Setting a General Cell

Users can type text or numbers in a general cell type. The general cell type can also be used as a base from which to inherit generic cell type information.

The general cell will format the values the user types into strings, numbers, or dates. The text cell formats user typed values as strings.

For details on the properties and methods for this cell type, refer to the **GeneralCellType ('GeneralCellType Class' in the on-line documentation)** class.

Using Code

1. Define the general cell type by creating an instance of the **GeneralCellType ('GeneralCellType Class' in the on-line documentation)** class.
2. Assign the general cell type to a cell.

Example

This example sets a cell to be a general cell.

C#

```
FarPoint.Web.Spread.GeneralCellType gencell = new
FarPoint.Web.Spread.GeneralCellType();
FpSpread1.ActiveSheetView.Cells[1, 1].CellType = gencell;
```

VB

```
Dim gencell As New FarPoint.Web.Spread.GeneralCellType()
FpSpread1.ActiveSheetView.Cells(1, 1).CellType = gencell
```

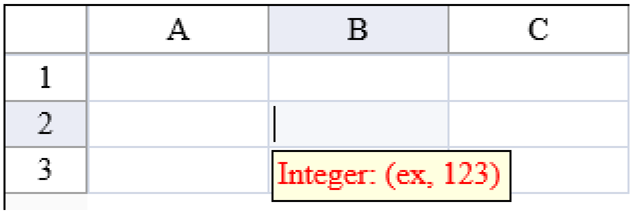
Using the Spread Designer

1. In the work area, select the cell or cells for which you want to set the cell type.
2. Select the **Home** menu.
3. Select the **SetCellType** icon under the **CellType** section.
4. Select the cell type and any other cell properties.
5. Select **OK** to close the dialog.
6. Click **Apply and Exit** to close the Spread Designer.

Setting an Integer Cell

You can use integer cells to restrict users to entering numeric values as integers only and to display data as integers. The default error message is displayed if the user types a decimal value and tries to leave the cell.

The following image displays the default error message.



Integer: (ex, 123)

You can specify an edit mode format with the **EditMode ('EditMode Property' in the on-line documentation)** property and the **IntegerCellType.EditModeSettings ('IntegerCellType.EditModeSettings Class' in the on-line documentation)** class.

For details on the properties and methods for this cell type, refer to the **IntegerCellType ('IntegerCellType Class' in the on-line documentation)** class.

Using Code

1. Define the integer cell type by creating an instance of the **IntegerCellType ('IntegerCellType Class' in the on-line documentation)** class.
2. Assign the integer cell type to a cell.

Example

This example sets a cell to be an integer cell.

C#

```
FarPoint.Web.Spread.IntegerCellType intcell = new
FarPoint.Web.Spread.IntegerCellType();
FpSpread1.ActiveSheetView.Cells[1, 1].CellType = intcell;
```

VB

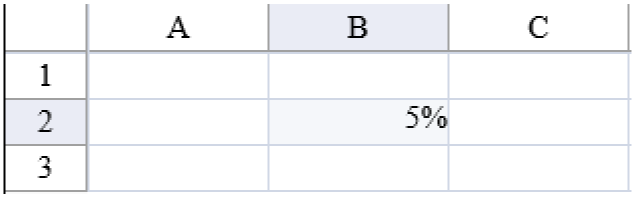
```
Dim intcell As New FarPoint.Web.Spread.IntegerCellType()
FpSpread1.ActiveSheetView.Cells(1, 1).CellType = intcell
```

Using the Spread Designer

1. In the work area, select the cell or cells for which you want to set the cell type.
2. Select the **Home** menu.
3. Select the **SetCellType** icon under the **CellType** section.
4. Select the cell type and any other cell properties.
5. Select **OK** to close the dialog.
6. Click **Apply and Exit** to close the Spread Designer.

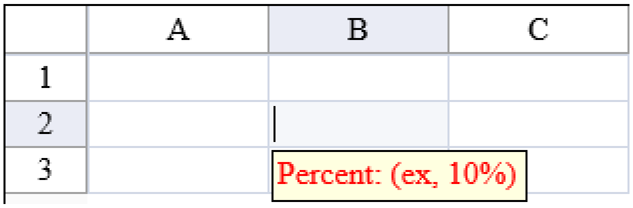
Setting a Percent Cell

You can use a percent cell to display percent values. In a percent cell type, a value of 0.35 is displayed as 35%. You can use a percent cell for displaying values as percentages and restricting inputs to percentage numeric values.



	A	B	C
1			
2		5%	
3			

The percent cell has a default error message that is displayed if the user types an invalid value and tries to leave the cell.



	A	B	C
1			
2			
3			

Percent: (ex, 10%)

For details on the properties and methods for this cell type, refer to the **PercentCellType ('PercentCellType Class' in the on-line documentation)** class.

Using Code

1. Define the percent cell type by creating an instance of the **PercentCellType ('PercentCellType Class' in the on-line documentation)** class.
2. Assign the percent cell type to a cell.

Example

This example sets a cell to be a percent cell.

C#

```
FarPoint.Web.Spread.PercentCellType pctcell = new
FarPoint.Web.Spread.PercentCellType();
FpSpread1.ActiveSheetView.Cells[1, 1].CellType = pctcell;
```

VB

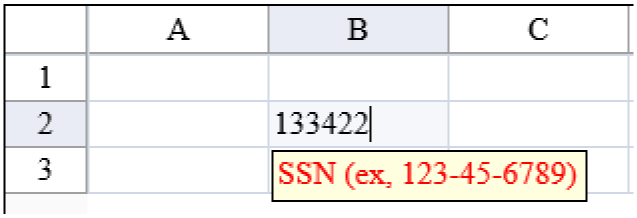
```
Dim pctcell As New FarPoint.Web.Spread.PercentCellType()
FpSpread1.ActiveSheetView.Cells(1, 1).CellType = pctcell
```

Using the Spread Designer

1. In the work area, select the cell or cells for which you want to set the cell type.
2. Select the **Home** menu.
3. Select the **SetCellType** icon under the **CellType** section.
4. Select the cell type and any other cell properties.
5. Select **OK** to close the dialog.
6. Click **Apply and Exit** to close the Spread Designer.

Setting a Regular Expression Cell

A regular expression cell is a cell that contains a text box that restricts the way data is entered in the cell to valid entries defined in a regular expression.



	A	B	C
1			
2		133422	
3			

SSN (ex, 123-45-6789)

A default error message is displayed if the user types an invalid value and tries to leave the cell.

For details on the properties and methods for this cell type, refer to the **RegExpCellType ('RegExpCellType Class' in the on-line documentation)** class.

The Regular Expression cell type expects the data in the cell to be a string. If you are working with other data types, such as date-time, then you need to get the date-time input, convert it to a string for the cell in the **Format** override, then convert the data back to a date-time in the **Parse** override.

For more information about creating regular expressions, refer to the following Microsoft web site topics:

- http://msdn.microsoft.com/en-us/library/hs600312(v=vs.110).aspx
- http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.regularexpressionvalidator.validationexpression(v=vs.110).aspx

Using Code

1. Define the regular expression cell type by creating an instance of the **RegExpCellType ('RegExpCellType Class' in the on-line documentation)** class.
2. Create a regular expression.
3. Create a message to display to the user when expression is not valid.
4. Apply the regular expression cell type to a cell or range of cells.

Example

To create a cell that restricts user input to match a regular expression, use this code.

C#

```
FarPoint.Web.Spread.RegExpCellType rgex = new FarPoint.Web.Spread.RegExpCellType();
rgex.ValidationExpression = "^\\d{3}-\\d{2}-\\d{4}$";
rgex.ErrorMessage = "SSN (ex, 123-45-6789)";
FpSpread1.ActiveSheetView.Cells[0, 0].CellType = rgex;
```

VB

```
Dim rgex As New FarPoint.Web.Spread.RegExpCellType()
rgex.ValidationExpression = "^\d{3}-\d{2}-\d{4}$"
rgex.ErrorMessage = "SSN (ex, 123-45-6789)"
FpSpread1.ActiveSheetView.Cells(0, 0).CellType = rgex
```

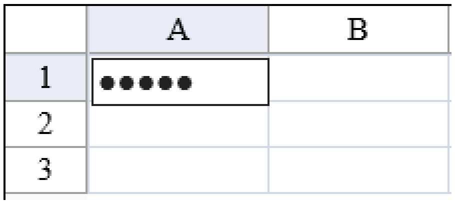
Using the Spread Designer

1. In the work area, select the cell or cells for which you want to set the cell type.
2. Select the **Home** menu.
3. Select the **SetCellType** icon under the **CellType** section.
4. Select the cell type and any other cell properties.
5. Select **OK** to close the dialog.
6. Click **Apply and Exit** to close the Spread Designer.

Setting a Text Cell

You can use text cells to restrict users to entering text values and to display data as text. You can also specify the maximum text length and a password field.

The following image displays a text cell with a password.



	A	B
1	●●●●●	
2		
3		

For details on the properties and methods for this cell type, refer to the **TextCellType ('TextCellType Class' in the on-line documentation)** class.

For information about checking a regular expression in a cell, refer to **Setting a Regular Expression Cell**.

> If you anticipate that users need to display text with multiple blank spaces between consecutive words, you should set the **AllowWrap ('AllowWrap Property' in the on-line documentation)** property to false; otherwise, the control uses spaces for formatting that Internet Explorer will trim as extra blank spaces (Internet Explorer allows one space in a TD element). If you set the **AllowWrap ('AllowWrap Property' in the on-line documentation)** property to false, the control uses the web space character for spaces, and the spaces are not trimmed in the text.

Using Code

1. Define the text cell type by creating an instance of the **TextCellType ('TextCellType Class' in the on-line documentation)** class.
2. Set properties for the text cell.
3. Apply the text cell type to a cell or range of cells.

Example

This example sets a password and a maximum length for the cell.

C#

```
FarPoint.Web.Spread.TextCellType tcell = new FarPoint.Web.Spread.TextCellType();
tcell.Password = true;
tcell.MaxLength = 5;
FpSpread1.ActiveSheetView.Cells[0, 0].CellType = tcell;
```

VB

```
Dim tcell As New FarPoint.Web.Spread.TextCellType()
tcell.Password = True
tcell.MaxLength = 5
FpSpread1.ActiveSheetView.Cells(0, 0).CellType = tcell
```

Using the Spread Designer

1. In the work area, select the cell or cells for which you want to set the cell type.
2. Select the **Home** menu.
3. Select the **SetCellType** icon under the **CellType** section.
4. Select the cell type and any other cell properties.
5. Select **OK** to close the dialog.
6. Click **Apply and Exit** to close the Spread Designer.

Working with Graphical Cell Types

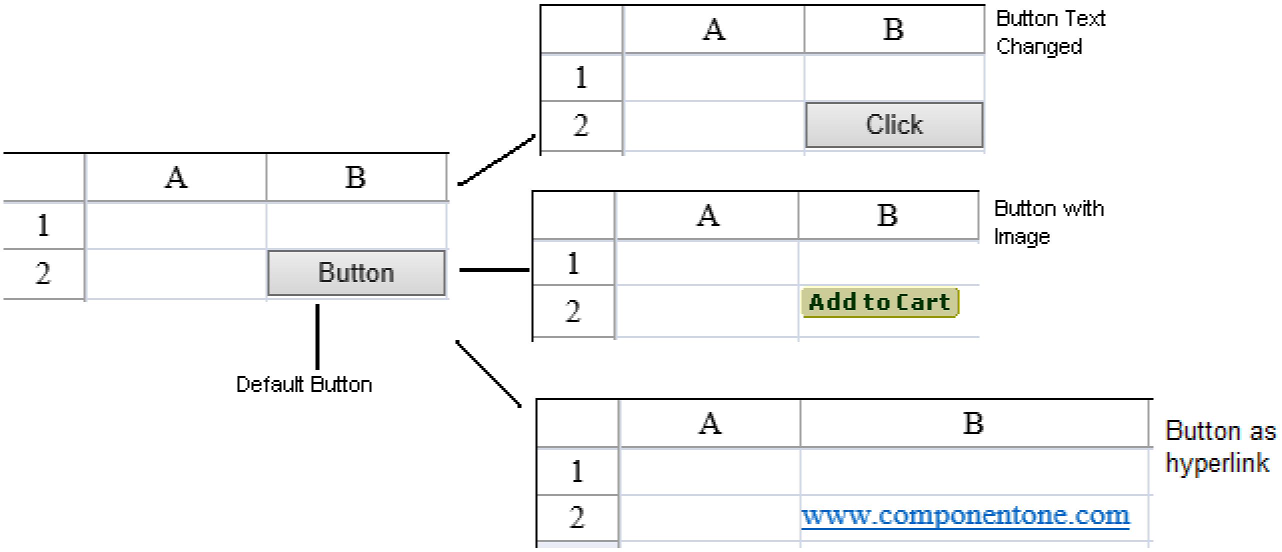
There are several cell types in the spreadsheet that have a graphical appearance. The following tasks show how to create them:

- **Setting a Button Cell**
- **Setting a Check Box Cell**
- **Setting a Combo Box Cell**
- **Setting a Hyperlink Cell**
- **Setting an Image Cell**
- **Setting a Label Cell**
- **Setting a List Box Cell**
- **Setting a Multiple-Column Combo Box Cell**
- **Setting a Radio Button List Cell**
- **Setting a Tag Cloud Cell**

For information on the various cell type classes, refer to the cell type classes in the **Assembly Reference (on-line documentation)**.

Setting a Button Cell

A button cell displays a rectangular button (with the word Button by default unless you specify otherwise) that behaves like a button control. You can specify the text for that button or you can substitute the entire graphic image. You can also specify text to appear underlined and blue, as a hypertext link would appear. The default appearance and the three possible alternatives are shown in this figure.



To create a cell that acts like a button, follow this procedure.

For details on the properties and methods for this cell type, refer to the **ButtonCellType ('ButtonCellType Class' in the on-line documentation)** class in the **Assembly Reference (on-line documentation)**.

Using Code

1. Define the button cell type by creating an instance of the ButtonCellType class and specify whether it is push button, image button, or link button.
2. Specify the properties of the button by setting the properties of that instance. Specify the text in the button or an image to use.
3. Specify the command to execute when the button is selected.
4. Assign the type to a cell (or cells).
5. Use the **ButtonCommand ('ButtonCommand Event' in the on-line documentation)** event to respond to the selected button.

Example

In this example, create a button with text in the first cell, a button with a stored image in the cell diagonally below that, and a button with link text in the one diagonally below that. These are identical to those pictured in the figure (except for placement). Use the **ButtonCommand ('ButtonCommand Event' in the on-line documentation)** event to respond to the selected button.

C#

```
FpSpread1.ActiveSheetView.Cells[0, 0].CellType = new
FarPoint.Web.Spread.ButtonCellType("OneCommand",
FarPoint.Web.Spread.ButtonType.PushButton, "Click");
FpSpread1.ActiveSheetView.Cells[1, 1].CellType = new
FarPoint.Web.Spread.ButtonCellType("OneCommand",
FarPoint.Web.Spread.ButtonType.ImageButton, "images/addtocart.gif");
FpSpread1.ActiveSheetView.Cells[2, 2].CellType = new
FarPoint.Web.Spread.ButtonCellType("OneCommand",
FarPoint.Web.Spread.ButtonType.LinkButton, "www.componentone.com");

protected void FpSpread1_ButtonCommand(object sender,
FarPoint.Web.Spread.SpreadCommandEventArgs e)
        {
            System.Drawing.Point p;
            p = (System.Drawing.Point)e.CommandArgument;
            TextBox1.Text = "You clicked the button in row " + p.X + " , column " +
p.Y;
        }
```

VB

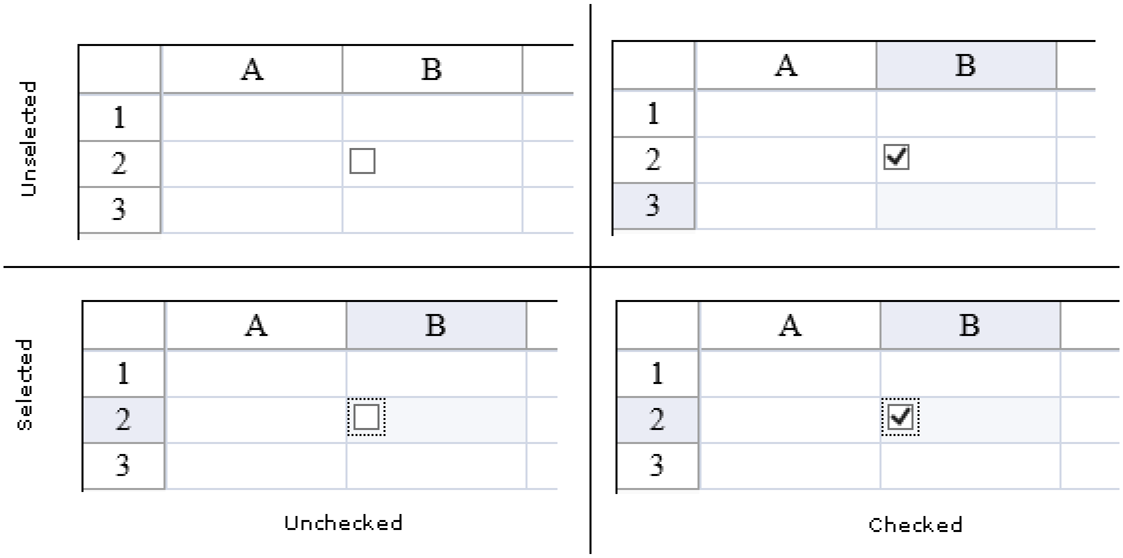
```
Dim btnc1 As New FarPoint.Web.Spread.ButtonCellType()
btnc1.CommandName = "OneCommand"
btnc1.ButtonType = FarPoint.Web.Spread.ButtonType.PushButton
btnc1.Text = "Click"
FpSpread1.ActiveSheetView.Cells(0, 0).CellType = btnc1

Dim btnc2 As New FarPoint.Web.Spread.ButtonCellType()
btnc2.CommandName = "OneCommand"
btnc2.ButtonType = FarPoint.Web.Spread.ButtonType.ImageButton
btnc2.ImageUrl = "addtocart.gif"
FpSpread1.ActiveSheetView.Cells(1, 1).CellType = btnc2

Dim btnc3 As New FarPoint.Web.Spread.ButtonCellType()
btnc3.CommandName = "OneCommand"
btnc3.ButtonType = FarPoint.Web.Spread.ButtonType.LinkButton
btnc3.Text = "www.componentone.com"
FpSpread1.ActiveSheetView.Cells(2, 2).CellType = btnc3

Private Sub FpSpread1ButtonCommand(ByVal sender As Object, ByVal e As
FarPoint.Web.Spread.SpreadCommandEventArgs) Handles FpSpread1.ButtonCommand
    TextBox1.Text = "You clicked the button in row " & e.CommandArgument.X & " , column
" & e.CommandArgument.Y
End Sub
```

Using the Spread Designer

1. In the work area, select the cell or cells for which you want to set the cell type.
2. Select the **Home** menu.
3. Select the **SetCellType** icon under the **CellType** section.
4. Select the cell type and any other cell properties.
5. Select **OK** to close the dialog.
6. Click **Apply and Exit** to close the Spread Designer.

Setting a Check Box Cell

A check box cell displays, by default, a small check box that can have one of two states, checked or unchecked. You can customize the check box by specifying the images for the two states. You can also specify whether the value is Boolean (true or false) or an integer (0 or 1). Default appearances are shown here.



To create a cell that acts like a check box, follow this procedure.

For details on the properties and methods for this cell type, refer to the **CheckBoxCellType ('CheckBoxCellType Class' in the on-line documentation)** class in the **Assembly Reference (on-line documentation)**.

Using Code

1. Define the check box cell type by creating an instance of the **CheckBoxCellType ('CheckBoxCellType Class' in the on-line documentation)** class.
2. Specify the properties of the check box cell.
3. Enter the text that goes along with each check box.
4. Specify the location of the images for the checked and unchecked boxes if you do not want to use the defaults.
5. Assign the check box cell type to a cell (or cells).

Example

This example creates a check box cell with custom images and text.

C#

```
FarPoint.Web.Spread.CheckBoxCellType chkbx = new
FarPoint.Web.Spread.CheckBoxCellType();
chkbx.Text = "Spotted";
chkbx.CheckedImageUrl = "img/checked.gif";
chkbx.UncheckedImageUrl = "img/unchecked.gif";
FpSpread1.ActiveSheetView.Cells[0, 0].CellType = chkbx;
```

VB

```
Dim chkbx As New FarPoint.Web.Spread.CheckBoxCellType()
chkbx.Text = "Spotted"
chkbx.CheckedImageUrl = "img/checked.gif"
chkbx.UncheckedImageUrl = "img/unchecked.gif"
FpSpread1.ActiveSheetView.Cells(0, 0).CellType = chkbx
```

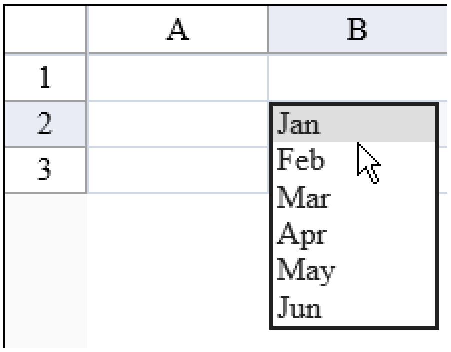
Using the Spread Designer

1. In the work area, select the cell or cells for which you want to set the cell type.

2. Select the **Home** menu.
3. Select the **SetCellType** icon under the **CellType** section.
4. Select the cell type and any other cell properties.
5. Select **OK** to close the dialog.
6. Click **Apply and Exit** to close the Spread Designer.

Setting a Combo Box Cell

A combo box cell displays an editable type box with a drop-down list of items when the cell is selected. The user may either type in the editable box (which searches for the item in the list) or select an item from the drop-down list. You can customize the appearance of the drop-down list as well as its behavior.



The combo box cell can be bound to data by setting the **DataSource ('DataSource Property' in the on-line documentation)**, **DataSourceID ('DataSourceID Property' in the on-line documentation)**, **DataMember ('DataMember Property' in the on-line documentation)**, **DataTextField ('DataTextField Property' in the on-line documentation)**, or **DataValueField ('DataValueField Property' in the on-line documentation)** property.

To create a cell that acts like a combo box, follow the procedure described here.

For details on the properties and methods for this cell type, refer to the **ComboBoxCellType ('ComboBoxCellType Class' in the on-line documentation)** class in the **Assembly Reference (on-line documentation)**.

Using Code

1. Define a combo box cell type 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.
3. Assign the list of items to a cell (or cells) defined as a combo box cell(s).
4. Specify any properties for that cell (or cells).
5. Assign the combo box cell type to a cell (or cells).

Example

This example creates a string array and adds the items to the combo cell.

C#

```
string[] cbstr;
cbstr = new String[] {"Jan", "Feb", "Mar", "Apr", "May", "Jun"};
FarPoint.Web.Spread.ComboBoxCellType cmbbx = new
FarPoint.Web.Spread.ComboBoxCellType(cbstr);
FpSpread1.ActiveSheetView.Cells[1, 1].CellType = cmbbx;
```

VB

```
Dim cbstr As string( )
cbstr = new String() {"Jan", "Feb", "Mar", "Apr", "May", "Jun"}
Dim cmbbx As New FarPoint.Web.Spread.ComboBoxCellType(cbstr)
FpSpread1.ActiveSheetView.Cells(1, 1).CellType = cmbbx
```

Using Code

1. Define a combo box cell type by creating an instance of the **ComboBoxCellType ('ComboBoxCellType Class' in the on-line documentation)** class.
2. Specify the data base table that is the source of the combo box drop-down list.
3. Populate a combo box drop-down list with values from a database table.
4. Specify any properties for that cell (or cells).
5. Assign the combo box cell type to a cell (or cells).

Example

This example creates a dataset and adds the items to the combo cell.

C#

```
System.Data.DataSet ds = new System.Data.DataSet();
System.Data.DataTable name;
System.Data.DataTable city;
name = ds.Tables.Add("Customers");
name.Columns.AddRange(new System.Data.DataColumn[] {new
System.Data.DataColumn("LastName", typeof(string)), new
System.Data.DataColumn("FirstName", typeof(string)), new System.Data.DataColumn("ID",
typeof(Int32))});
name.Rows.Add(new object[] {"Fielding", "William", 0});
name.Rows.Add(new object[] {"Williams", "Arthur", 1});
name.Rows.Add(new object[] {"Zuchini", "Theodore", 2});
city = ds.Tables.Add("City/State");
city.Columns.AddRange(new System.Data.DataColumn[] {new System.Data.DataColumn("City",
typeof(string)), new System.Data.DataColumn("Owner", typeof(Int32)), new
System.Data.DataColumn("State", typeof(string))});
city.Rows.Add(new object[] {"Atlanta", 0, "Georgia"});
city.Rows.Add(new object[] {"Boston", 1, "Mass."});
city.Rows.Add(new object[] {"Tampa", 2, "Fla."});

FarPoint.Web.Spread.ComboBoxCellType c = new FarPoint.Web.Spread.ComboBoxCellType();
System.Data.DataTable dt;
System.Collections.ArrayList al = new
System.Collections.ArrayList(ds.Tables[0].Rows.Count);

dt = ds.Tables[0];
for (int i = 0; i < (dt.Rows.Count - 1); i++)
{
    al.Add(dt.Rows[i][0]);
}
string[] s = null;
s = (string[])al.ToArray(typeof(string));
c.Items = s;
c.ShowButton = true;
FpSpread1.Sheets[0].Cells[1,1].CellType = c;
```

VB

```
Dim ds As New System.Data.DataSet
Dim name As System.Data.DataTable
Dim city As System.Data.DataTable
name = ds.Tables.Add("Customers")
name.Columns.AddRange(New System.Data.DataColumn() {New
System.Data.DataColumn("LastName", Type.GetType("System.String")), New
System.Data.DataColumn("FirstName", Type.GetType("System.String")), New
System.Data.DataColumn("ID", Type.GetType("System.Int32"))})
name.Rows.Add(New Object() {"Fielding", "William", 0})
name.Rows.Add(New Object() {"Williams", "Arthur", 1})
name.Rows.Add(New Object() {"Zuchini", "Theodore", 2})
city = ds.Tables.Add("City/State")
city.Columns.AddRange(New System.Data.DataColumn() {New System.Data.DataColumn("City",
Type.GetType("System.String")), New System.Data.DataColumn("Owner",
Type.GetType("System.Int32")), New System.Data.DataColumn("State",
Type.GetType("System.String"))})
city.Rows.Add(New Object() {"Atlanta", 0, "Georgia"})
city.Rows.Add(New Object() {"Boston", 1, "Mass."})
city.Rows.Add(New Object() {"Tampa", 2, "Fla."})
Dim c As New FarPoint.Web.Spread.ComboBoxCellType
Dim dt As System.Data.DataTable
Dim al As New ArrayList(ds.Tables(0).Rows.Count)
dt = ds.Tables(0)
Dim i As Int32
For i = 0 To dt.Rows.Count - 1
al.Add(dt.Rows(i)(0))
Next
Dim s As String()
s = al.ToArray(GetType(String))
c.Items = s
c.ShowButton = True
FpSpread1.Sheets(0).Cells(1, 1).CellType = c
```

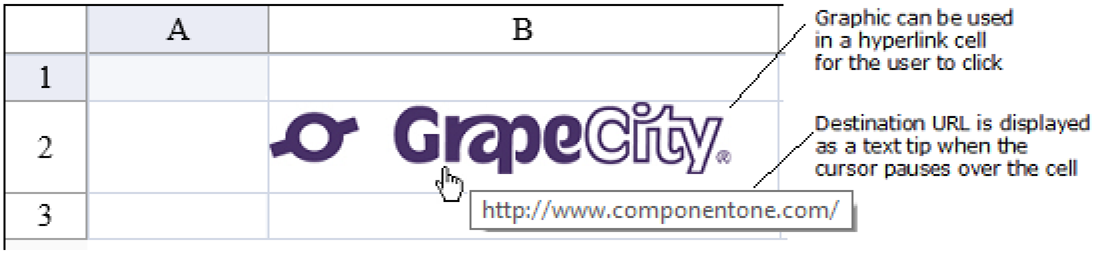
Using the Spread Designer

1. In the work area, select the cell or cells for which you want to set the cell type.
2. Select the **Home** menu.
3. Select the **SetCellType** icon under the **CellType** section.
4. Select the cell type and any other cell properties.
5. Select **OK** to close the dialog.
6. Click **Apply and Exit** to close the Spread Designer.

Setting a Hyperlink Cell

A hyperlink cell displays either text or an image that can serve as a hyperlink when clicked. The destination universal resource locator (URL), sometimes called the address, in the hyperlink can be any valid URL, such as www.yoursite.com or emailto:support@fpoint.com. For up-level browsers that support it, a text tip appears with the destination URL when the cursor pauses over the hyperlink cell. For down-level browsers, the status bar displays the URL when the cursor is over the link.

You can specify the target in which to display the destination, for example a new window (target = "_blank") or the same window (target= "_self").



Graphic can be used in a hyperlink cell for the user to click

Destination URL is displayed as a text tip when the cursor pauses over the cell

To create a cell that acts like a hyperlink, follow the procedure described here.

For details on the properties and methods for this cell type, refer to the **HyperLinkCellType ('HyperLinkCellType Class' in the on-line documentation)** class in the **Assembly Reference (on-line documentation)**.

Using Code

1. Define the hyperlink cell type by creating an instance of the **HyperLinkCellType ('HyperLinkCellType Class' in the on-line documentation)** class.
2. Choose the graphic, if you are making a hyperlink with a graphical image, and make sure the graphic file is in the correct directory and is the correct file type for the browser to display it.
3. Set the size of the cell, if you want it to match the size of a graphic to be used in the hyperlink.
4. Specify the destination URL for the hyperlink. You can optionally set the target. The destination URL is displayed as text if the cell does not use a graphic.
5. Assign the hyperlink cell to a cell (or cells).

Example

This example sets the size of the cell (by column and row) so that the graphic fits in it, then defines the location of the graphic to use as a hyperlink button, then specifies the destination URL (and has it opening is a separate window).

C#

```
FpSpread1.ActiveSheetView.Columns[1].Width = 145;
FpSpread1.ActiveSheetView.Rows[1].Height = 45;
string linkImage = @"images\fplogo.jpg";
string linkURL = "http://www.componentone.com";
string linkTarget = "_blank";
FarPoint.Web.Spread.HyperLinkCellType linkcell = new
FarPoint.Web.Spread.HyperLinkCellType();
linkcell.ImageUrl = linkImage;
linkcell.NavigateUrl = linkURL;
linkcell.Target = linkTarget;
FpSpread1.ActiveSheetView.Cells[1, 1].CellType = linkcell;
```

VB

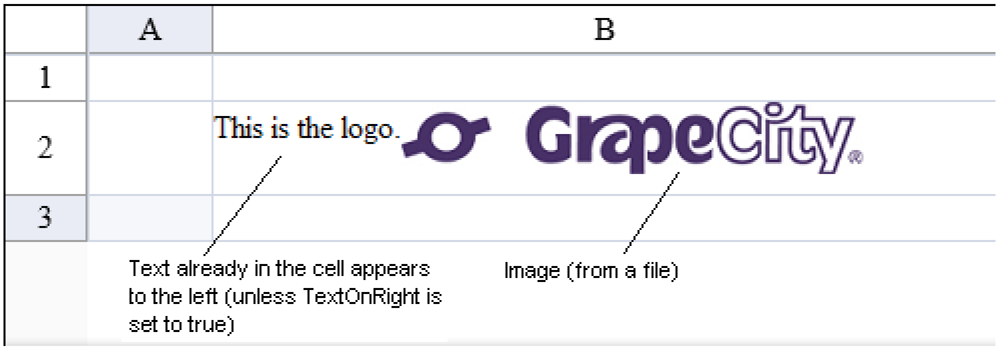
```
FpSpread1.ActiveSheetView.Columns(1).Width = 145
FpSpread1.ActiveSheetView.Rows(1).Height = 45
Dim linkimage As String = "images\fplogo.jpg"
Dim linkURL As String = "http://www.componentone.com"
Dim linkTarget As String = "_blank"
Dim linkcell As New FarPoint.Web.Spread.HyperLinkCellType()
linkcell.ImageUrl = linkImage
linkcell.NavigateUrl = linkURL
linkcell.Target = linkTarget
FpSpread1.ActiveSheetView.Cells(1, 1).CellType = linkcell
```

Using the Spread Designer

1. In the work area, select the cell or cells for which you want to set the cell type.
2. Select the **Home** menu.
3. Select the **SetCellType** icon under the **CellType** section.
4. Select the cell type and any other cell properties.
5. Select **OK** to close the dialog.
6. Click **Apply and Exit** to close the Spread Designer.

Setting an Image Cell

An image cell can contain a graphic image from a file. The cell may or may not already have text associated with it. The text can be displayed to the right or the left of the image in the cell.


A
B
1
2
3
This is the logo.
GrapeCity
Text already in the cell appears to the left (unless TextOnRight is set to true)
Image (from a file)

To create a cell that contains an image, follow this procedure.

For details on the properties and methods for this cell type, refer to the **ImageCellType ('ImageCellType Class' in the on-line documentation)** class in the **Assembly Reference (on-line documentation)**.

Using Code

1. Define the image cell type by creating an instance of the **ImageCellType ('ImageCellType Class' in the on-line documentation)** class.
2. Set the text to appear to the left or right of the image.
3. Assign the image cell to a cell (or cells).

Example

This example sets the column and width of the cell so that the image and text can appear together alongside each other, sets the cell type to image cell, and sets the defined text to appear to the left of the graphic image.

C#

```
FpSpread1.Sheets[0].Columns[1].Width = 240;
FpSpread1.Sheets[0].Rows[1].Height = 50;
FarPoint.Web.Spread.ImageCellType imagecell = new FarPoint.Web.Spread.ImageCellType();
imagecell.ImageUrl = "images\fplogo.jpg";
imagecell.TextOnRight = false; // display text to left of image
FpSpread1.ActiveSheetView.Cells[1,1].Text="This is the logo.";
FpSpread1.ActiveSheetView.Cells[1,1].CellType = imagecell;
```

VB

```
FpSpread1.Sheets(0).Columns(1).Width = 240
```

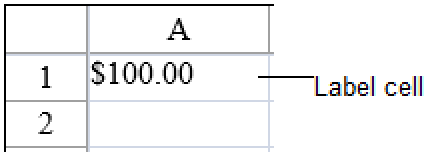
```
FpSpread1.Sheets(0).Rows(1).Height = 50
Dim imagecell As New FarPoint.Web.Spread.ImageCellType()
imagecell.ImageUrl = "images\fplogo.jpg"
imagecell.TextOnRight = false
FpSpread1.ActiveSheetView.Cells(1,1).Text="This is the logo."
FpSpread1.ActiveSheetView.Cells(1,1).CellType = imagecell
```

Using the Spread Designer

1. In the work area, select the cell or cells for which you want to set the cell type.
2. Select the **Home** menu.
3. Select the **SetCellType** icon under the **CellType** section.
4. Select the cell type and any other cell properties.
5. Select **OK** to close the dialog.
6. Click **Apply and Exit** to close the Spread Designer.

Setting a Label Cell

A label cell is a cell that cannot be edited by the end user and serves as a label for other cells.



	A
1	$100.00
2	

To create a cell that contains a label, follow this procedure.

For details on the properties and methods for this cell type, refer to the **LabelCellType ('LabelCellType Class' in the on-line documentation)** class in the **Assembly Reference (on-line documentation)**.

Using Code

1. Define the label cell type by creating an instance of the **LabelCellType ('LabelCellType Class' in the on-line documentation)** class.
2. Format and specify the text to appear as the label.
3. Assign the label cell type to a cell.

Example

This example creates a label cell that displays a currency value.

C#

```
FarPoint.Web.Spread.LabelCellType lblcell = new FarPoint.Web.Spread.LabelCellType();
int i =100;
string fstring = i.ToString("C");
lblcell.FormatString = fstring;
FpSpread1.ActiveSheetView.Cells[0, 0].CellType = lblcell;
FpSpread1.ActiveSheetView.Cells[0, 0].Text = fstring;
```

VB

```
Dim lblcell As New FarPoint.Web.Spread.LabelCellType()
Dim i As Integer = 100
Dim fstring As String = i.ToString("C")
```

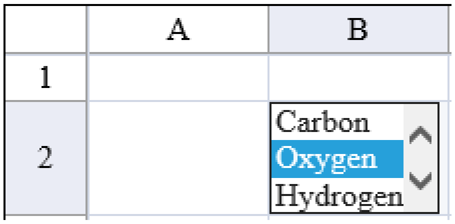
```
lblcell.FormatString = fstring
FpSpread1.ActiveSheetView.Cells(0, 0).CellType = lblcell
FpSpread1.ActiveSheetView.Cells(0, 0).Text = fstring
```

Using the Spread Designer

1. In the work area, select the cell or cells for which you want to set the cell type.
2. Select the **Home** menu.
3. Select the **SetCellType** icon under the **CellType** section.
4. Select the cell type and any other cell properties.
5. Select **OK** to close the dialog.
6. Click **Apply and Exit** to close the Spread Designer.

Setting a List Box Cell

A list box cell is a cell that displays a list of items when selected.



To create a cell that contains a list box, follow this procedure.

For details on the properties and methods for this cell type, refer to the **ListBoxCellType ('ListBoxCellType Class' in the on-line documentation)** class in the **Assembly Reference (on-line documentation)**.

Using Code

1. Define the list box cell type by creating an instance of the **ListBoxCellType ('ListBoxCellType Class' in the on-line documentation)** class.
2. Specify the list of items.
3. Assign the list box cell type to a cell.

Example

Display the list of several elements.

C#

```
FarPoint.Web.Spread.ListBoxCellType lbcell = new FarPoint.Web.Spread.ListBoxCellType();
lbcell.Items = new String[] {"Carbon", "Oxygen", "Hydrogen"};
lbcell.SelectedBackColor = System.Drawing.Color.Yellow;
lbcell.SelectedForeColor = System.Drawing.Color.DarkBlue;
FpSpread1.ActiveSheetView.Cells[1, 1].CellType = lbcell;
```

VB

```
Dim lbcell As New FarPoint.Web.Spread.ListBoxCellType()
lbcell.Items = new String() {"Carbon", "Oxygen", "Hydrogen"}
lbcell.SelectedBackColor = System.Drawing.Color.Yellow
lbcell.SelectedForeColor = System.Drawing.Color.DarkBlue
```

```
FpSpread1.ActiveSheetView.Cells(1, 1).CellType = lbcell
```

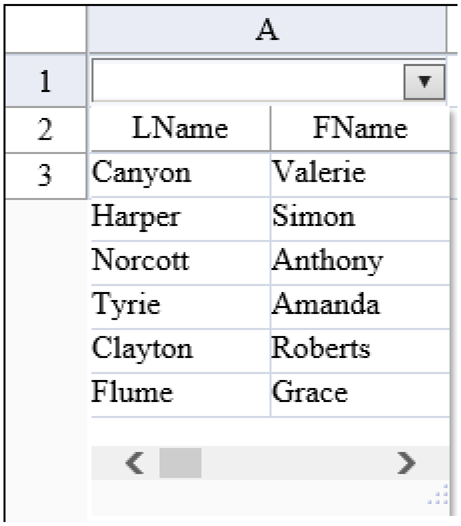
Using the Spread Designer

1. In the work area, select the cell or cells for which you want to set the cell type.
2. Select the **Home** menu.
3. Select the **SetCellType** icon under the **CellType** section.
4. Select the cell type and any other cell properties.
5. Select **OK** to close the dialog.
6. Click **Apply and Exit** to close the 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 and allow the user to choose from a displayed list.

You specify the list of items by binding the cell. You can also choose which column is displayed in the edit area of the cell with the **ColumnEditName ('ColumnEditName Property' in the on-line documentation)** property.



LName	FName
Canyon	Valerie
Harper	Simon
Norcott	Anthony
Tyrie	Amanda
Clayton	Roberts
Flume	Grace

The multi-column combo cell can be bound to data by setting the **DataSource ('DataSource Property' in the on-line documentation)**, **DataSourceID ('DataSourceID Property' in the on-line documentation)**, **DataMember ('DataMember Property' in the on-line documentation)**, **DataColumn ('DataColumn Property' in the on-line documentation)**, or **DataColumnName ('DataColumnName Property' in the on-line documentation)** property.

For details on the properties and methods for the multi-column combo box cell type, refer to the **MultiColumnComboBoxCellType ('MultiColumnComboBoxCellType Class' in the on-line documentation)** class in the **Assembly Reference (on-line documentation)**.

For details on the combo cell type, refer to the **ComboBoxCellType ('ComboBoxCellType Class' in the on-line documentation)** class in the **Assembly Reference (on-line documentation)**.

Using Code

1. Define the multi-column combo cell type by creating an instance of the **MultiColumnComboBoxCellType ('MultiColumnComboBoxCellType Class' in the on-line documentation)** class.
2. Create a dataset.
3. Specify the list of options by binding.

4. Set the display properties such as the **ShowButton ('ShowButton Property' in the on-line documentation)** property.
5. Assign the cell type to a specific cell.

Example

Display a multi-column combo cell.

C#

```
string conStr = "Provider=Microsoft.JET.OLEDB.4.0;data source=
C:\\common\\Patients2000.mdb";
string sqlStr = "SELECT * FROM Patients";
System.Data.OleDb.OleDbConnection conn = new System.Data.OleDb.OleDbConnection(conStr);
System.Data.DataSet ds = new System.Data.DataSet();
System.Data.OleDb.OleDbDataAdapter da = new System.Data.OleDb.OleDbDataAdapter(sqlStr,
conn); da.Fill(ds);
FarPoint.Web.Spread.MultiColumnComboBoxCellType mcombo = new
FarPoint.Web.Spread.MultiColumnComboBoxCellType();
mcombo.DataSource = ds;
mcombo.DataColumnName = "Patients";
mcombo.ShowButton = true;
fpSpread1.Sheets[0].Cells[0, 0].CellType = mcombo;
```

VB

```
Dim conStr As String = "Provider=Microsoft.JET.OLEDB.4.0;data source=
C:\Common\Patients2000.mdb"
Dim sqlStr As String = "SELECT * FROM Patients"
Dim conn As New System.Data.OleDb.OleDbConnection(conStr)
Dim ds As System.Data.DataSet = New System.Data.DataSet()
Dim da As New System.Data.OleDb.OleDbDataAdapter(sqlStr, conn)
da.Fill(ds)
Dim mcombo As New FarPoint.Web.Spread.MultiColumnComboBoxCellType()
mcombo.DataSource = ds
mcombo.DataColumnName = "Patients"
mcombo.ShowButton = True
FpSpread1.Sheets(0).Cells(0, 0).CellType = mcombo
```

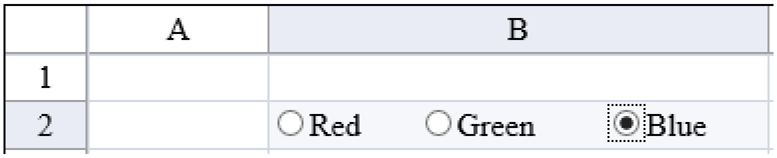
Using the Spread Designer

1. In the work area, select the cell or cells for which you want to set the cell type.
2. Select the **Home** menu.
3. Select the **SetCellType** icon under the **CellType** section.
4. Select the cell type and any other cell properties.
5. Select **OK** to close the dialog.
6. Click **Apply and Exit** to close the Spread Designer.

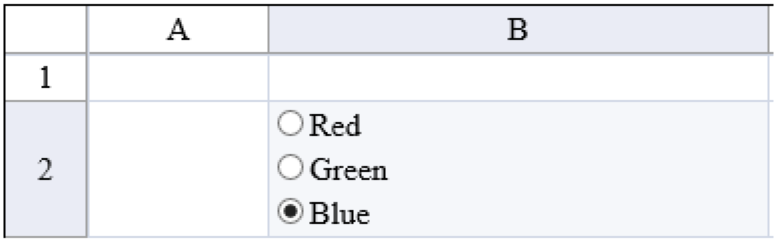
Setting a Radio Button List Cell

A radio button cell is a cell that displays a radio button (or option button) list. You can specify the list of options and specify whether to display the list horizontally or vertically. By default, the list displays horizontally in the cell. To display the list vertically, set the **RepeatDirection ('RepeatDirection Property' in the on-line documentation)** property to Vertical. Use the **Values ('Values Property' in the on-line documentation)** property to add the list of options.

The following figures display the horizontal and vertical directions.



	A	B
1		
2		○Red ○Green ●Blue



	A	B
1		
2		○Red ○Green ●Blue

Spread does not support radio buttons without labels; each radio button must have a label.

To create a cell that displays a radio button list, follow the procedure described below.

For details on the properties and methods for this cell type, refer to the **RadioButtonListCellType ('RadioButtonListCellType Class' in the on-line documentation)** class in the **Assembly Reference (on-line documentation)**.

Using Code

1. Set the size of the cell (or column) to fit the list of options.
2. Define the radio button list cell type by creating an instance of the **RadioButtonListCellType ('RadioButtonListCellType Class' in the on-line documentation)** class.
3. Specify the list of options.
4. Set the display properties of the cell.
5. Assign the radio button list cell type to a specific cell.

Example

Display the vertical list of colors as radio buttons.

C#

```
FpSpread1.ActiveSheetView.Columns[1].Width = 250;
string[] rbval;
string[] rbstr;
rbstr = new String[] {"Red", "Green", "Blue"};
rbval = new String[] {"R", "G", "B"};
FarPoint.Web.Spread.RadioButtonListCellType rblct = new
FarPoint.Web.Spread.RadioButtonListCellType(rbstr);
rblct.Values = rbval;
rblct.RepeatDirection = RepeatDirection.Vertical;
FpSpread1.ActiveSheetView.Cells[1, 1].CellType = rblct;
```

VB

```
FpSpread1.ActiveSheetView.Columns(1).Width = 250
Dim rbstr As String()
Dim rbval As String()
rbstr = New String() {"Red", "Green", "Blue"}
rbval = New String() {"R", "G", "B"}
```

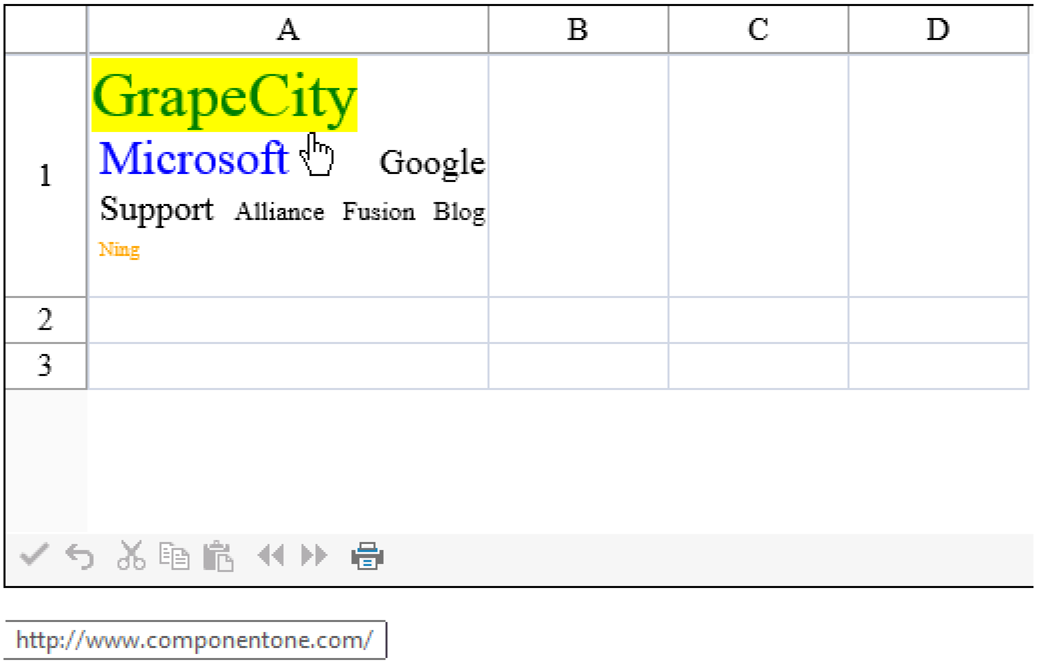
```
Dim rblct As New FarPoint.Web.Spread.RadioButtonListCellType(rbstr)
rblct.Values = rbval
rblct.RepeatDirection = RepeatDirection.Vertical
FpSpread1.ActiveSheetView.Cells(1, 1).CellType = rblct
```

Using the Spread Designer

1. In the work area, select the cell or cells for which you want to set the cell type.
2. Select the **Home** menu.
3. Select the **SetCellType** icon under the **CellType** section.
4. Select the cell type and any other cell properties.
5. Select **OK** to close the dialog.
6. Click **Apply and Exit** to close the Spread Designer.

Setting a Tag Cloud Cell

A tag cloud cell allows you to display a box or cloud of text tags, a weighted list of linked items. In the figure below several strings (Web site names) are listed in a tag cloud in the order of their weights. The color of the text is also an indication of their weights. Notice that they have corresponding link URLs.



A text tip appears with the HTML anchor for the cloud item when the cursor pauses over the cell as displayed in the above image. You can specify an additional text tip with the **Title ('Title Property' in the on-line documentation)** property.

You can customize how many tags are displayed, their order, their font and color, and other aspects of the tag cloud. The size of the displayed item depends on the weight of the item. The tag cloud cell can also be bound to a data source.

For details on the properties and methods for this cell type, refer to the **TagCloudCellType ('TagCloudCellType Class' in the on-line documentation)** class.

You can also use the TagCloudItem class to specify properties for the tag cloud cell.

Using Code

To set up a tag cloud cell type, simply define the tags and weights and assign the tag cloud cell type to a particular cell.

Example

This example puts items in the cell, sets the rank colors and weights, and creates links for the items.

C#

```
// Place this code in ASPX.Form load:
string[] text = { "GrapeCity", "Blog", "Ning", "Fusion", "Alliance", "Google",
"Microsoft", "Support" };
string[] weight = { "290", "50", "41", "56", "78", "170", "208", "140" };
string[] href = { "http://www.componentone.com", "http://www.FarPointSpread.com/Blog",
"http://www.ning.com", "http://labs.developerfusion.co.uk", "http://aspalliance.com/",
"http://www.google.com", "http://www.microsoft.com", "http://www.clubFarPoint.com" };
string[] rc = { "Red", "Orange", "Black", "Black", "Black", "Blue", "Green" };

FarPoint.Web.Spread.TagCloudCellType tagger = new
FarPoint.Web.Spread.TagCloudCellType();
tagger.BackColor = System.Drawing.Color.Empty;
tagger.RankingColors = rc;
tagger.HoverColor = "Yellow";
tagger.DisplayCount = 8;
tagger.SortOrder = FarPoint.Web.Spread.SortOrder.WeightDescending;

FpSpread1.Sheets[0].Cells[0, 0].CellType = tagger;
FpSpread1.Sheets[0].Cells[0, 0].Value =
FarPoint.Web.Spread.TagCloudCellType.ConvertToTagCloudItems(text, weight, href);
FpSpread1.Sheets[0].Columns[0, 0].Width = 200;
FpSpread1.Sheets[0].Rows[0, 0].Height = 120;
```

VB

```
' Place this code in ASPX.Form load:
Dim text As String() = { "GrapeCity", "Blog", "Ning", "Fusion", "Alliance", "Google",
"Microsoft", "Support" }
Dim weight As String() = { "290", "50", "41", "56", "78", "170", "208", "140" }
Dim href As String() = { "http://www.componentone.com",
"http://www.FarPointSpread.com/Blog", "http://www.ning.com",
"http://labs.developerfusion.co.uk", "http://aspalliance.com/",
"http://www.google.com", "http://www.microsoft.com", "http://www.clubFarPoint.com" }
Dim rc As String() = {"Red", "Orange", "Black", "Black", "Black", "Blue", "Green"}

Dim tagger As New FarPoint.Web.Spread.TagCloudCellType()
tagger.BackColor = System.Drawing.Color.Empty
tagger.RankingColors = rc
tagger.HoverColor = "Yellow"
tagger.DisplayCount = 8
tagger.SortOrder = FarPoint.Web.Spread.SortOrder.WeightDescending

FpSpread1.Cells(0, 0).CellType = tagger
FpSpread1.Cells(0, 0).Value =
FarPoint.Web.Spread.TagCloudCellType.ConvertToTagCloudItems(text, weight, href)
FpSpread1.Sheets(0).Columns(0, 0).Width = 200
FpSpread1.Sheets(0).Rows(0, 0).Height = 120
```

Using the Spread Designer

1. In the work area, select the cell or cells for which you want to set the cell type.
2. Select the **Home** menu.
3. Select the **SetCellType** icon under the **CellType** section.
4. Select the cell type and any other cell properties.
5. Select **OK** to close the dialog.
6. Click **Apply and Exit** to close the Spread Designer.

Working with ASP.NET AJAX Extender Cell Types

There are several cell types that are made available by using ASP.NET AJAX. The following tasks show how to create them:

- **Setting an Automatic-Completion Cell**
- **Setting a Calendar Cell**
- **Setting a Combo Box Cell**
- **Setting a Filtered Text Cell**
- **Setting a Masked Edit Cell**
- **Setting a Mutually Exclusive Check Box Cell**
- **Setting a Numeric Spin Cell**
- **Setting a Rating Cell**
- **Setting a Slider Cell**
- **Setting a Slide Show Cell**
- **Setting a Text Box with Watermark Cell**

These cell types are dependent on the extenders of the Microsoft ASP.NET AJAX Control Toolkit. The appearance and behavior of these cell types are controlled by the AJAX Control Toolkit. For more information, see the Microsoft AJAX site or https://ajaxcontroltoolkit.codeplex.com/.

All controls in the AJAX Control Toolkit can add or attach client behaviors to a target control. The target control is exposed as the **Editor** property of the extender cell type in Spread.

Multiple behaviors can be attached to the same target control, which is also supported in the AJAX extender implementation of Spread. For example, we can add a MaskedEditExtender to DateCalendarCellType. As a result, this cell type combines two extenders.

Unlike the other cell types in Spread, there are no restrictions and validation of any of the extenders properties, so use this cell type carefully and provide validation as needed.

Be aware of these limitations to these AJAX cell types.

- The AJAX Control Toolkit is still under development, so some of the control properties don't behave correctly as designed.
- When users export these cell types to PDF or Excel files, only the cell type value is exported.

Follow these guidelines for using AJAX extenders in a development environment.

- A ScriptManager must be placed on the Page in order to use AJAX extenders.
- If you use the .NET 2.0 framework version of the FarPoint.Web.Spread.Extender.dll assembly in the .NET 3.5 framework, and you wish to use the 3.5 version of the AJAXControlToolkit.dll assembly, you would need to add binding redirect code to the web.config file (See the code below).
  To set a binding redirect, code similar to the following would need to be added to the Web config file (the version number should match the actual version number of the assembly).

Code

```
<runtime> <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
<dependentAssembly> <assemblyIdentity name="AjaxControlToolkit"
```

```
publicKeyToken="28f01b0e84b6d53e"/> <bindingRedirect
oldVersion="1.0.0.0-1.1.0.0" newVersion="3.0.20229.20843"/>
</dependentAssembly> </assemblyBinding> </runtime>
```

For more information on AJAX support, refer to **Working with AJAX**.

For information on the various cell type classes, refer to the cell type classes in the **Assembly Reference (on-line documentation)**.

Setting an Automatic-Completion Cell

You can provide an automatic-completion edit box with the **AutoCompleteCellType ('AutoCompleteCellType Class' in the on-line documentation)** class that attempts to complete the typed entry.

This cell type is dependent on the AutoComplete extender of the Microsoft ASP.NET AJAX Control Toolkit. The appearance and behavior of this cell type is controlled by the AJAX Control Toolkit. For more information, see the Microsoft AJAX site.

All controls in the AJAX Control Toolkit can add or attach client behaviors to a target control. The target control is exposed as the **Editor** property of the extender cell type in Spread.

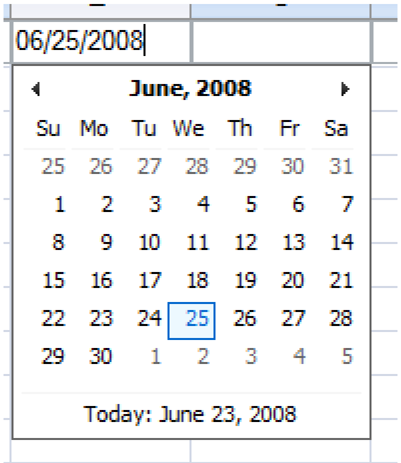
Unlike the other cell types in Spread, there are no restrictions and validation of any of the Extenders properties, so use this cell type carefully and provide validation as needed.

For a list of limitations with this AJAX cell type, refer to **Working with ASP.NET AJAX Extender Cell Types**.

For details on the properties and methods for this cell type, refer to the **AutoCompleteCellType ('AutoCompleteCellType Class' in the on-line documentation)** class.

Setting a Calendar Cell

You can provide a date-selection box that has a drop-down calendar with the **DateCalendarCellType ('DateCalendarCellType Class' in the on-line documentation)** class.



06/25/2008

Su	Mo	Tu	We	Th	Fr	Sa
25	26	27	28	29	30	31
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	1	2	3	4	5

Today: June 23, 2008

This cell type is dependent on the Calendar extender of the Microsoft ASP.NET AJAX Control Toolkit. The appearance and behavior of this cell type is controlled by the AJAX Control Toolkit. For more information, see the Microsoft AJAX site.

All controls in the AJAX Control Toolkit can add or attach client behaviors to a target control. The target control is exposed as the **Editor** property of the extender cell type in Spread.

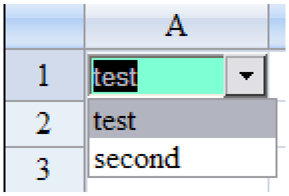
Unlike the other cell types in Spread, there are no restrictions and validation of any of the Extenders properties, so use this cell type carefully and provide validation as needed.

For a list of limitations with this AJAX cell type, refer to **Working with ASP.NET AJAX Extender Cell Types**.

For details on the properties and methods for this cell type, refer to the **DateCalendarCellType ('DateCalendarCellType Class' in the on-line documentation)** class.

Setting a Combo Box Cell

You can provide a combo box that has a drop-down list with the **AjaxComboBoxCellType ('AjaxComboBoxCellType Class' in the on-line documentation)** class.



This cell type is dependent on the combo extender of the Microsoft ASP.NET AJAX Control Toolkit. The appearance and behavior of this cell type is controlled by the AJAX Control Toolkit. For more information, see the Microsoft AJAX site. This celltype requires version 3.0.3.512 or later of the Ajax Toolkit.

All controls in the AJAX Control Toolkit can add or attach client behaviors to a target control. The target control is exposed as the **Editor** property of the extender cell type in Spread.

Unlike the other cell types in Spread, there are no restrictions and validation of any of the Extenders properties, so use this cell type carefully and provide validation as needed.

For a list of limitations with this AJAX cell type, refer to **Working with ASP.NET AJAX Extender Cell Types**.

For details on the properties and methods for this cell type, refer to the **AjaxComboBoxCellType ('AjaxComboBoxCellType Class' in the on-line documentation)** class.

Setting a Filtered Text Cell

You can provide a filtered text box with the **FilteredTextCellType ('FilteredTextCellType Class' in the on-line documentation)** class. This cell type allows you to type certain specified characters without a mask.

This cell type is dependent on the FilteredText extender of the Microsoft ASP.NET AJAX Control Toolkit. The appearance and behavior of this cell type is controlled by the AJAX Control Toolkit. For more information, see the Microsoft AJAX site.

All controls in the AJAX Control Toolkit can add or attach client behaviors to a target control. The target control is exposed as the **Editor** property of the extender cell type in Spread.

Unlike the other cell types in Spread, there are no restrictions and validation of any of the Extenders properties, so use this cell type carefully and provide validation as needed.

For a list of limitations with this AJAX cell type, refer to **Working with ASP.NET AJAX Extender Cell Types**.

For details on the properties and methods for this cell type, refer to the **FilteredTextCellType ('FilteredTextCellType Class' in the on-line documentation)** class.

Setting a Masked Edit Cell

You can provide a masked edit cell with the **MaskedEditCellType ('MaskedEditCellType Class' in the on-line documentation)** class. This cell type allows you to type certain specified characters using a mask.

This cell type is dependent on the MaskedEdit extender of the Microsoft ASP.NET AJAX Control Toolkit. The appearance and behavior of this cell type is controlled by the AJAX Control Toolkit. For more information, see the Microsoft AJAX site.

All controls in the AJAX Control Toolkit can add or attach client behaviors to a target control. The target control is exposed as the **Editor** property of the extender cell type in Spread.

Unlike the other cell types in Spread, there are no restrictions and validation of any of the Extenders properties, so use this cell type carefully and provide validation as needed.

For a list of limitations with this AJAX cell type, refer to **Working with ASP.NET AJAX Extender Cell Types**.

For details on the properties and methods for this cell type, refer to the **MaskedEditCellType ('MaskedEditCellType Class' in the on-line documentation)** class.

Setting a Mutually Exclusive Check Box Cell

You can provide a series of mutually exclusive check box cells with the **MutuallyExclusiveCheckBoxCellType ('MutuallyExclusiveCheckBoxCellType Class' in the on-line documentation)** class. This cell type allows you to have a group of check box cells where the user can only check one at a time.

This cell type is dependent on the MutuallyExclusiveCheckBox extender of the Microsoft ASP.NET AJAX Control Toolkit. The appearance and behavior of this cell type is controlled by the AJAX Control Toolkit. For more information, see the Microsoft AJAX site.

All controls in the AJAX Control Toolkit can add or attach client behaviors to a target control. The target control is exposed as the **Editor** property of the extender cell type in Spread.

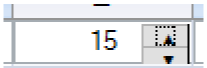
Unlike the other cell types in Spread, there are no restrictions and validation of any of the Extenders properties, so use this cell type carefully and provide validation as needed.

For a list of limitations with this AJAX cell type, refer to **Working with ASP.NET AJAX Extender Cell Types**.

For details on the properties and methods for this cell type, refer to the **MutuallyExclusiveCheckBoxCellType ('MutuallyExclusiveCheckBoxCellType Class' in the on-line documentation)** class.

Setting a Numeric Spin Cell

You can provide a numeric cell with up-down spin buttons with the **NumericUpDownCellType ('NumericUpDownCellType Class' in the on-line documentation)** class.



This cell type is dependent on the NumericUpDown extender of the Microsoft ASP.NET AJAX Control Toolkit. The appearance and behavior of this cell type is controlled by the AJAX Control Toolkit. For more information, see the Microsoft AJAX site.

All controls in the AJAX Control Toolkit can add or attach client behaviors to a target control. The target control is exposed as the **Editor** property of the extender cell type in Spread.

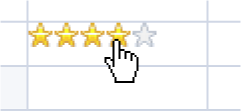
Unlike the other cell types in Spread, there are no restrictions and validation of any of the Extenders properties, so use this cell type carefully and provide validation as needed.

For a list of limitations with this AJAX cell type, refer to **Working with ASP.NET AJAX Extender Cell Types**.

For details on the properties and methods for this cell type, refer to the **NumericUpDownCellType ('NumericUpDownCellType Class' in the on-line documentation)** class.

Setting a Rating Cell

You can provide a rating cell that displays the stars (or other icon) with the **RatingCellType ('RatingCellType Class' in the on-line documentation)** class.



This cell type is dependent on the Rating extender of the Microsoft ASP.NET AJAX Control Toolkit. The appearance and behavior of this cell type is controlled by the AJAX Control Toolkit. For more information, see the Microsoft AJAX site.

All controls in the AJAX Control Toolkit can add or attach client behaviors to a target control. The target control is exposed as the **Editor** property of the extender cell type in Spread.

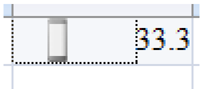
Unlike the other cell types in Spread, there are no restrictions and validation of any of the Extenders properties, so use this cell type carefully and provide validation as needed.

For a list of limitations with this AJAX cell type, refer to **Working with ASP.NET AJAX Extender Cell Types**.

For details on the properties and methods for this cell type, refer to the **RatingCellType ('RatingCellType Class' in the on-line documentation)** class.

Setting a Slider Cell

You can provide a slider control cell with the **SliderCellType ('SliderCellType Class' in the on-line documentation)** class.



This cell type is dependent on the Slider extender of the Microsoft ASP.NET AJAX Control Toolkit. The appearance and behavior of this cell type is controlled by the AJAX Control Toolkit. For more information, see the Microsoft AJAX site.

All controls in the AJAX Control Toolkit can add or attach client behaviors to a target control. The target control is exposed as the **Editor** property of the extender cell type in Spread.

Unlike the other cell types in Spread, there are no restrictions and validation of any of the Extenders properties, so use this cell type carefully and provide validation as needed.

For a list of limitations with this AJAX cell type, refer to **Working with ASP.NET AJAX Extender Cell Types**.

For details on the properties and methods for this cell type, refer to the **SliderCellType ('SliderCellType Class' in the on-line documentation)** class.

Setting a Slide Show Cell

You can provide a cell that displays a slide-show with the **SlideShowCellType ('SlideShowCellType Class' in the on-line documentation)** class.

This cell type is dependent on the SlideShow extender of the Microsoft ASP.NET AJAX Control Toolkit. The appearance and behavior of this cell type is controlled by the AJAX Control Toolkit. For more information, see the Microsoft AJAX site.

All controls in the AJAX Control Toolkit can add or attach client behaviors to a target control. The target control is exposed as the **Editor** property of the extender cell type in Spread.

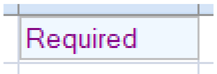
Unlike the other cell types in Spread, there are no restrictions and validation of any of the Extenders properties, so use this cell type carefully and provide validation as needed.

For a list of limitations with this AJAX cell type, refer to **Working with ASP.NET AJAX Extender Cell Types**.

For details on the properties and methods for this cell type, refer to the **SlideShowCellType ('SlideShowCellType Class' in the on-line documentation)** class.

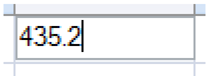
Setting a Text Box with Watermark Cell

You can provide a text box cell that displays a watermark with the **TextBoxWatermarkCellType ('TextBoxWatermarkCellType Class' in the on-line documentation)** class.



Required

Without focus (watermark displayed)



435.2

Edit mode (watermark not displayed)

This cell type is dependent on the TextBoxWatermark extender of the Microsoft ASP.NET AJAX Control Toolkit. The appearance and behavior of this cell type is controlled by the AJAX Control Toolkit. For more information, see the Microsoft AJAX site.

All controls in the AJAX Control Toolkit can add or attach client behaviors to a target control. The target control is exposed as the **Editor** property of the extender cell type in Spread.

Unlike the other cell types in Spread, there are no restrictions and validation of any of the Extenders properties, so use this cell type carefully and provide validation as needed.

For a list of limitations with this AJAX cell type, refer to **Working with ASP.NET AJAX Extender Cell Types**.

For details on the properties and methods for this cell type, refer to the **TextBoxWatermarkCellType ('TextBoxWatermarkCellType Class' in the on-line documentation)** class.

Using Validation Controls

You can prevent a user from entering invalid characters in a cell by using a validation control in the Spread cell. You can validate the data when pasting to a cell or cell range by setting the **NonEditModeValidation ('NonEditModeValidation Property' in the on-line documentation)** property to True when using a validation control. Set the **AllowServerValidators ('AllowServerValidators Property' in the on-line documentation)** property to True to support server validation using custom validator controls. You can also display an error message when the data is invalid with the **ValidationErrorMessage ('ValidationErrorMessage Property' in the on-line documentation)** property.

Initialize the validation control and then add the validation control to the Spread cell to provide validation.

You can use validation controls instead of the standard validation by setting the **Validators ('Validators Property' in the on-line documentation)** property. The following cell types provide this option:

- **CurrencyCellType ('CurrencyCellType Class' in the on-line documentation)**
- **DateTimeCellType ('DateTimeCellType Class' in the on-line documentation)**
- **DoubleCellType ('DoubleCellType Class' in the on-line documentation)**
- **GeneralCellType ('GeneralCellType Class' in the on-line documentation)**
- **IntegerCellType ('IntegerCellType Class' in the on-line documentation)**
- **PercentCellType ('PercentCellType Class' in the on-line documentation)**
- **RegExpCellType ('RegExpCellType Class' in the on-line documentation)**
- **TextCellType ('TextCellType Class' in the on-line documentation)**

If the validation fails, the **onErrorMessageShown ('ErrorMessageShown' in the on-line documentation)** event occurs and the user cannot change the active cell.

The validation is not supported if the **ShowEditor ('ShowEditor Property' in the on-line documentation)** property is true.

If you receive an error such as, "UnobtrusiveValidationMode requires a ScriptResourceMapping for 'jquery'" in Visual Studio 2012, then you may need to add the following information to web.config.

<appSettings>

```
<add key="ValidationSettings:UnobtrusiveValidationMode" value="None"/>
</appSettings>
```

Using Code

1. Add the validator code to the ASPX page.
2. Create the cell.
3. Create a validator.
4. Create validation messages.
5. Set the Validators property.

Example

This example creates a cell and assigns a basic validator to the cell.

Code

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
ErrorMessage="RequiredFieldValidator"></asp:RequiredFieldValidator>
  <asp:CompareValidator ID="CompareValidator1" runat="server"
ErrorMessage="CompareValidator"></asp:CompareValidator>
  <asp:RangeValidator ID="RangeValidator1" runat="server"
ErrorMessage="RangeValidator"></asp:RangeValidator>
```

C#

```
protected void Page_Load(object sender, System.EventArgs e)
{
if (IsPostBack) return;
FpSpread1.Sheets[0].RowCount = 10;
FpSpread1.Sheets[0].ColumnCount = 7;
// RequiredFieldValidator, from code
FarPoint.Web.Spread.TextCellType txt1 = new FarPoint.Web.Spread.TextCellType();
RequiredFieldValidator rfv = new RequiredFieldValidator();
rfv.ErrorMessage = "RequiredFieldValidator, from code: value required!";
txt1.Validators.Add(rfv);
FpSpread1.ActiveSheetView.Cells[0, 0].Text = "RequiredFieldValidator, from code";
FpSpread1.ActiveSheetView.Cells[0, 1].CellType = txt1;

// RequiredFieldValidator, from toolbox
FarPoint.Web.Spread.TextCellType txt2 = new FarPoint.Web.Spread.TextCellType();
RequiredFieldValidator1.ErrorMessage = "RequiredFieldValidator, from toolbox: value
required!";
txt2.Validators.Add(RequiredFieldValidator1);
FpSpread1.ActiveSheetView.Cells[1, 0].Text = "RequiredFieldValidator, from toolbox";
FpSpread1.ActiveSheetView.Cells[1, 1].CellType = txt2;

// CompareValidator, from toolbox
FarPoint.Web.Spread.TextCellType txt3 = new FarPoint.Web.Spread.TextCellType();
CompareValidator1.ErrorMessage = "CompareValidator, from toolbox: password does not
match! Enter \"Spread\"";
CompareValidator1.ValueToCompare = "Spread";
txt3.Validators.Add(CompareValidator1);
FpSpread1.ActiveSheetView.Cells[2, 0].Text = "CompareValidator, from toolbox";
FpSpread1.ActiveSheetView.Cells[2, 1].CellType = txt3;

// CompareValidator, from code
FarPoint.Web.Spread.TextCellType txt4 = new FarPoint.Web.Spread.TextCellType();
CompareValidator cv = new CompareValidator();
cv.ErrorMessage = "CompareValidator, from toolbox: password does not match! Enter
\"Spread\"";
cv.ValueToCompare = "Spread";
txt4.Validators.Add(cv);
FpSpread1.ActiveSheetView.Cells[3, 0].Text = "CompareValidator, from code";
FpSpread1.ActiveSheetView.Cells[3, 1].CellType = txt4;

// RangeValidator, from toolbox
FarPoint.Web.Spread.TextCellType txt5 = new FarPoint.Web.Spread.TextCellType();
RangeValidator1.ErrorMessage = "RangeValidator, from toolbox: value should in range [10
- 15]";
RangeValidator1.MinimumValue = "10"; RangeValidator1.MaximumValue = "15";
RangeValidator1.Type = ValidationDataType.Integer;
txt5.Validators.Add(RangeValidator1);
FpSpread1.ActiveSheetView.Cells[4, 0].Text = "RangeValidator, from toolbox";
FpSpread1.ActiveSheetView.Cells[4, 1].CellType = txt5;

// RangeValidator, from code
FarPoint.Web.Spread.TextCellType txt6 = new FarPoint.Web.Spread.TextCellType();
RangeValidator rv = new RangeValidator();
rv.ErrorMessage = "RangeValidator, from toolbox: value should in range [10 - 15]";
rv.MinimumValue = "10"; rv.MaximumValue = "15";
rv.Type = ValidationDataType.Integer;
txt6.Validators.Add(rv);
FpSpread1.ActiveSheetView.Cells[5, 0].Text = "RangeValidator, from toolbox";
FpSpread1.ActiveSheetView.Cells[5, 1].CellType = txt6;
}
```

VB

```
Protected Sub Page_Load(sender As Object, e As System.EventArgs)
If IsPostBack Then
 Return
End If
FpSpread1.Sheets(0).RowCount = 10
FpSpread1.Sheets(0).ColumnCount = 7
' RequiredFieldValidator, from code
Dim txt1 As New FarPoint.Web.Spread.TextCellType()
Dim rfv As New RequiredFieldValidator()
rfv.ErrorMessage = "RequiredFieldValidator, from code: value required!"
txt1.Validators.Add(rfv)
FpSpread1.ActiveSheetView.Cells(0, 0).Text = "RequiredFieldValidator, from code"
FpSpread1.ActiveSheetView.Cells(0, 1).CellType = txt1

' RequiredFieldValidator, from toolbox
Dim txt2 As New FarPoint.Web.Spread.TextCellType()
RequiredFieldValidator1.ErrorMessage = "RequiredFieldValidator, from toolbox: value
required!"
txt2.Validators.Add(RequiredFieldValidator1)
FpSpread1.ActiveSheetView.Cells(1, 0).Text = "RequiredFieldValidator, from toolbox"
FpSpread1.ActiveSheetView.Cells(1, 1).CellType = txt2

' CompareValidator, from toolbox
Dim txt3 As New FarPoint.Web.Spread.TextCellType()
CompareValidator1.ErrorMessage = "CompareValidator, from toolbox: password does not
match! Enter ""Spread"""
CompareValidator1.ValueToCompare = "Spread"
```

```
txt3.Validators.Add(CompareValidator1)
FpSpread1.ActiveSheetView.Cells(2, 0).Text = "CompareValidator, from toolbox"
FpSpread1.ActiveSheetView.Cells(2, 1).CellType = txt3

' CompareValidator, from code
Dim txt4 As New FarPoint.Web.Spread.TextCellType()
Dim cv As New CompareValidator()
cv.ErrorMessage = "CompareValidator, from toolbox: password does not match! Enter
""Spread"""
cv.ValueToCompare = "Spread"
txt4.Validators.Add(cv)
FpSpread1.ActiveSheetView.Cells(3, 0).Text = "CompareValidator, from code"
FpSpread1.ActiveSheetView.Cells(3, 1).CellType = txt4

' RangeValidator, from toolbox
Dim txt5 As New FarPoint.Web.Spread.TextCellType()
RangeValidator1.ErrorMessage = "RangeValidator, from toolbox: value should in range [10
- 15]"
RangeValidator1.MinimumValue = "10"
RangeValidator1.MaximumValue = "15"
RangeValidator1.Type = ValidationDataType.[Integer]
txt5.Validators.Add(RangeValidator1)
FpSpread1.ActiveSheetView.Cells(4, 0).Text = "RangeValidator, from toolbox"
FpSpread1.ActiveSheetView.Cells(4, 1).CellType = txt5

' RangeValidator, from code
Dim txt6 As New FarPoint.Web.Spread.TextCellType()
Dim rv As New RangeValidator()
rv.ErrorMessage = "RangeValidator, from toolbox: value should in range [10 - 15]"
rv.MinimumValue = "10"
rv.MaximumValue = "15"
rv.Type = ValidationDataType.[Integer]
txt6.Validators.Add(rv)
FpSpread1.ActiveSheetView.Cells(5, 0).Text = "RangeValidator, from toolbox"
FpSpread1.ActiveSheetView.Cells(5, 1).CellType = txt6
End Sub
```

This example supports server validation using a custom validator control.

Code

```
<FarPoint:FpSpread ID="FpSpread1" runat="server" BorderColor="Black"
BorderStyle="Solid" BorderWidth="1px" Height="200" Width="400">
   <commandbar backcolor="Control" buttonfacecolor="Control"
buttonhighlightcolor="ControlLightLight" buttonshadowcolor="ControlDark"></commandbar>
   <sheets>
      <FarPoint:SheetView SheetName="Sheet1"></FarPoint:SheetView>
   </sheets>
</FarPoint:FpSpread>

<asp:CustomValidator ID="CustomValidator2" runat="server" ErrorMessage="Error of server
side" OnServerValidate="CustomValidator2_ServerValidate"></asp:CustomValidator>
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
  if (IsPostBack) return;
  TextCellType txt = new FarPoint.Web.Spread.TextCellType();
  txt.AllowServerValidators = true;// New property to enable server validating with
validator controls
  txt.Validators.Add(CustomValidator2);
  FpSpread1.ActiveSheetView.Cells[1, 1].CellType = txt;
  FpSpread1.ActiveSheetView.Cells[1, 1].BackColor = Color.LightPink;
}
protected void CustomValidator2_ServerValidate(object source, ServerValidateEventArgs
args)
{
int value = 0;
args.IsValid = int.TryParse(args.Value, out value) && value < 10;// Accept integer
number less than 10;
}
```

VB

```
Protected Sub Page_Load(sender As Object, e As System.EventArgs)
        If IsPostBack Then
         Return

Dim txt As New FarPoint.Web.Spread.TextCellType()
  txt.AllowServerValidators = True 'New property to enable server validating with
validator controls
  txt.Validators.Add(CustomValidator2)
  FpSpread1.ActiveSheetView.Cells(1, 1).CellType = txt
  FpSpread1.ActiveSheetView.Cells(1, 1).BackColor = Color.LightPink
End Sub
Protected Sub CustomValidator2_ServerValidate(source As Object, args As
ServerValidateEventArgs)
 Dim value As Integer = 0
 args.IsValid = Integer.TryParse(args.Value, value) AndAlso value < 10  'Accept integer
number less than 10
End Sub
```

Managing Data Binding

You can connect the spreadsheet in the component to a database or other data source. You can manage how data in the spreadsheet is bound. Tasks involved include:

- **Data Binding Overview**
- **Binding to a Data Source**
- **Binding to a Range**
- **Model Data Binding in ASP.NET 4.5**
- **Setting the Cell Types for Bound Data**
- **Displaying Data as a Hierarchy**
- **Handling Row Expansion**
- **Adding an Unbound Row**
- **Limiting Postbacks When Updating Bound Data**
- **Tutorial: Binding to a Corporate Database**

Data Binding Overview

You can bind the 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.

For the basic steps used to set up data binding, see **Binding to a Data Source**. For a detailed walk-through of binding to a database see, **Tutorial: Binding to a Corporate Database**.

You can customize data binding in many ways, including:

- You might want to specify a particular data table within a data source that has multiple tables. Use the SheetView class, **DataMember ('DataMember Property' in the on-line documentation)** property to specify the particular data member.
- You can perform more specific manipulations once the data is bound. You can use the **DataField ('DataField Property' in the on-line documentation)** property in the **Column ('Column Class' in the on-line documentation)** class to bind a column to a particular data field. For instance, if you want the bound column 3 to be displayed first on the spreadsheet, or if you want to hide a column of data, or if you simply want to display the bound columns in general in a different order than they exist in the data set, you can use the **DataField ('DataField Property' in the on-line documentation)** property in the particular column to achieve this.
- You might want to specify the key number as opposed to the row number; if so, use the **DataKeyField ('DataKeyField Property' in the on-line documentation)** property of the sheet.
- You can use the SQLDataAdapter instead of the OLEDB adapter when binding to a data set. To do so, use the SQLDataAdapter to set up your data set and bind it to the spreadsheet.
- In bound mode, the data model wraps the supplied data source and if needed can supply additional data and interactivity not available from the data source, for example cell formulas and unbound rows and columns. In a few cases, you may need to create your own custom data model for performance reasons. For more information about developing custom data models, refer to **Using Sheet Models**.
- Spread offers properties to work with model data binding in ASP.NET 4.5 and later. For more information, see **Model Data Binding in ASP.NET 4.5**.

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.

- How you handle state management while bound to a database or working with large data sets can affect your application’s performance. You need to set up state management to optimize performance and account for other factors. For more information, refer to **Maintaining State**.
- If your data set is not getting updated when you click the Update button, see the information and

instructions in **Limiting Postbacks When Updating Bound Data** to make sure that you have code in your page load event so that you only re-create the bound data when you are loading for the first time and not overwriting it on each post back.

Binding to a Data Source

The following instructions provide the code necessary to bind the FpSpread component to a data set.

The basic procedure is to bind data either to the sheet directly or to a data model that the sheet uses. This means either using the **DefaultSheetDataModel ('DefaultSheetDataModel Class' in the on-line documentation)** class to construct a data model and then in the **SheetView ('SheetView Class' in the on-line documentation)** class set the **DataModel ('DataModel Property' in the on-line documentation)** property to the newly created data model or setting the **DataSource ('DataSource Property' in the on-line documentation)** member in the SheetView class.

- How you handle state management while bound to a database or working with large data sets can affect your application’s performance. You need to set up state management to optimize performance and account for other factors. For more information, refer to **Maintaining State**.
- If your data set is not getting updated when you click the Update button, see the information and instructions in **Limiting Postbacks When Updating Bound Data** to make sure that you have code in your page load event so that you only re-create the bound data when you are loading for the first time and not overwriting it on each post back.

Using Code

1. Create a data model and set it equal to a **DefaultSheetDataModel** object, specifying a data set for the new model’s *dataSource* parameter.
   Other parameter options are available for the **DefaultSheetDataModel** constructor. Refer to the **Assembly Reference (on-line documentation)** for complete information.
2. Set the SheetView object’s **DataModel ('DataModel Property' in the on-line documentation)** property equal to the new data model.

Example

This example code binds the FpSpread component to a data set named dataSet1.

C#

```
// Create a data model using a data set.
FarPoint.Web.Spread.Model.DefaultSheetDataModel model = new
FarPoint.Web.Spread.Model.DefaultSheetDataModel(dataSet1);
FpSpread1.Sheets[0].DataModel = model;
```

VB

```
' Create a data model using a data set.
Dim model As New FarPoint.Web.Spread.Model.DefaultSheetDataModel(dataSet1)
FpSpread1.Sheets(0).DataModel = model
```

Using a Shortcut

Set the FpSpread **DataSource ('DataSource Property' in the on-line documentation)** property.

Example

This example code uses the FpSpread **DataSource ('DataSource Property' in the on-line documentation)** property to bind to a data set.

C#

```
FpSpread1.DataSource = ds;
```

VB

```
FpSpread1.DataSource = ds
```

Binding to a Range

You can bind a range of cells using the **CellRange ('CellRange Property' in the on-line documentation)** property in the **SpreadDataSource ('SpreadDataSource Class' in the on-line documentation)** class. This process creates a data source of the cell range using the SpreadDataSource control. This data source can then be bound to other controls (such as a chart or list box). You can put the initial data in the Spread component by binding it to a data source or leaving the component unbound and putting the data in the cells with code or other means (such as typing in the cells). If the component is unbound, then the **DataTextField** property would be the same as the column header text.

The SpreadDataSource control can be created with code or added to the Toolbox in Visual Studio.

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. Add the SpreadDataSource control to the form (double-click the icon in the toolbox).
2. Set the **CellRange** property in the properties window.
3. Set the sheet name.
4. Set the **SpreadID** property.
5. Set the **DataSource** property of the control you want to bind to.

Using Code

1. Create a data source.
2. Set the Spread **DataSource ('DataSource Property' in the on-line documentation)** property to the data source.
3. Create a Spread data source object.
4. Set the **CellRange ('CellRange Property' in the on-line documentation)** and other properties for the **SpreadDataSource ('SpreadDataSource Class' in the on-line documentation)** class.
5. Add the Spread data source to the page.
6. Add a list box control to the page.

Example

This example binds the Spread component to a data source, creates a cell range data source, and then binds the cell range data source to a list box control.

C#

```
System.Data.DataTable dt = new System.Data.DataTable("Test");
System.Data.DataRow dr = default(System.Data.DataRow);
dt.Columns.Add("Series0");
dt.Columns.Add("Series1");
dt.Columns.Add("Series2");
dr = dt.NewRow();
dr[0] = 2;
dr[1] = 1;
dr[2] = 5;
dt.Rows.Add(dr);
dr = dt.NewRow();
dr[0] = 4;
dr[1] = 2;
dr[2] = 5;
dt.Rows.Add(dr);
dr = dt.NewRow();
dr[0] = 3;
dr[1] = 4;
dr[2] = 5;
dt.Rows.Add(dr);
dr = dt.NewRow();
dr[0] = 3;
dr[1] = 4;
dr[2] = 5;

FpSpread1.DataSource = dt;
FarPoint.Web.Spread.SpreadDataSource spreadDS = new
FarPoint.Web.Spread.SpreadDataSource();
spreadDS.SheetName = FpSpread1.ActiveSheetView.SheetName;
spreadDS.SpreadID = "FpSpread1";//FpSpread1.ID;
spreadDS.CellRange = new FarPoint.Web.Spread.Model.CellRange(0, 0, 3, 1);
this.Controls.Add(spreadDS);
this.ListBox1.DataSource = spreadDS;
this.ListBox1.DataTextField = "Series0";
this.ListBox1.DataBind();
```

VB

```
Dim dt As New System.Data.DataTable("Test")
Dim dr As System.Data.DataRow
dt.Columns.Add("Series0")
dt.Columns.Add("Series1")
dt.Columns.Add("Series2")
dr = dt.NewRow()
dr(0) = 2
dr(1) = 1
dr(2) = 5
dt.Rows.Add(dr)
dr = dt.NewRow()
dr(0) = 4
dr(1) = 2
dr(2) = 5
dt.Rows.Add(dr)
dr = dt.NewRow()
dr(0) = 3
dr(1) = 4
dr(2) = 5
dt.Rows.Add(dr)
```

```
dr = dt.NewRow()
dr(0) = 3
dr(1) = 4
dr(2) = 5
FpSpread1.DataSource = dt
Dim spreadDS As New FarPoint.Web.Spread.SpreadDataSource()
spreadDS.SheetName = FpSpread1.ActiveSheetView.SheetName
spreadDS.SpreadID = "FpSpread1"
spreadDS.CellRange = New FarPoint.Web.Spread.Model.CellRange(0, 0, 3, 1)
Controls.Add(spreadDS)
ListBox1.DataSource = spreadDS
ListBox1.DataTextField = "Series0"
ListBox1.DataBind()
```

If your data set is not updated when you click the **Update** button, make sure that you have code similar to this in your page load event so that you only re-create the bound data when you are loading for the first time and not overwriting it on each post back.

C#

```
if !(Page.IsPostBack)
{
  FpSpread1.DataBind();
}
```

VB

```
If Not Page.IsPostBack Then
  FpSpread1.DataBind()
End If
```

Model Data Binding in ASP.NET 4.5

Spread for ASP.NET supports model data binding as provided by ASP.NET 4.5. The Spread component provides the **ItemType ('ItemType Property' in the on-line documentation)** property, as well as the **SelectMethod ('SelectMethod Property' in the on-line documentation)**, **UpdateMethod ('UpdateMethod Property' in the on-line documentation)**, **InsertMethod ('InsertMethod Property' in the on-line documentation)**, and **DeleteMethod ('DeleteMethod Property' in the on-line documentation)** properties for working with model binding.

Model data binding only takes effect in .NET 4.5 or higher; if you try to use model data binding in another .NET environment, nothing happens. If you use model data binding and the **DataSourceID ('DataSourceID Property' in the on-line documentation)** property in the same project, the component throws an exception. If you use the **DataSourceID ('DataSourceID Property' in the on-line documentation)** and set one of the model data binding properties, such as **SelectMethod ('SelectMethod Property' in the on-line documentation)** or **UpdateMethod ('UpdateMethod Property' in the on-line documentation)**, the component will also throw an exception.

The **ItemType ('ItemType Property' in the on-line documentation)** property indicates the type of data item used in model data binding. By default, it is empty. Set this property to use strongly typed data binding. If you set the **ItemType ('ItemType Property' in the on-line documentation)** property and set some sheets' **SelectMethod ('SelectMethod Property' in the on-line documentation)** property, the .NET framework will try to cast the data items to the type declared by the **ItemType ('ItemType Property' in the on-line documentation)** property. Therefore, you should set the **ItemType ('ItemType Property' in the on-line documentation)** and **SelectMethod ('SelectMethod Property' in the on-line documentation)** properties to the same data types, or set the **SelectMethod ('SelectMethod Property' in the on-line documentation)** to a parent data type. If you do not, the component will throw an exception.

The **SelectMethod ('SelectMethod Property' in the on-line documentation)**, **UpdateMethod ('UpdateMethod Property' in the on-line documentation)**, **InsertMethod ('InsertMethod Property' in the on-line documentation)**, and **DeleteMethod ('DeleteMethod Property' in the on-line documentation)** properties set the name of the method to use to get, update, insert, or delete a data item in the data source. Note the following when using these properties.

- If you set some of these properties, they are merged into the ActiveSheetView model data binding properties.
- If you set the **SelectMethod ('SelectMethod Property' in the on-line documentation)** property but do not set one or more of the other model data binding properties, when the component tries to update, insert, or delete a data item, the component will display an error.
- You should put the methods to which the properties are set in the current page. If you do not, the .NET framework might throw an exception. The methods can be static methods or instant methods. Currently, the .NET Framework only accepts public methods.

Using Code

1. Set up the data source. See the example code below.
2. Bind the Spread component to the data source. For more information, see **Binding to a Data Source**.
3. If you want to do so, set the **ItemType ('ItemType Property' in the on-line documentation)** property.
4. Set the **SelectMethod ('SelectMethod Property' in the on-line documentation)**, **UpdateMethod ('UpdateMethod Property' in the on-line documentation)**, **InsertMethod ('InsertMethod Property' in the on-line documentation)**, and **DeleteMethod ('DeleteMethod Property' in the on-line documentation)** properties to the names of methods you provide in your project for handling these data binding tasks.

Example

This example code illustrates the model data binding properties.

The following code is added to the .aspx page:

```
<Sheets>
 <FarPoint:SheetView SheetName="Sheet1"
  AllowDelete="true" AllowInsert="true"
  ItemType="DeptModel.User"
  SelectMethod="GetUsers" UpdateMethod="UpdateUser" DeleteMethod="DeleteUser" InsertMethod="InsertUser">
 </FarPoint:SheetView>
</Sheets>
```

Code is added to the .cs or .vb page to create the methods referred to in the .aspx page, as shown in the following sample.

C#

```
  public IQueryable<User> GetUsers()
  {
    DeptEntities db = new DeptEntities();
    return db.Users.AsQueryable();
  }

  public bool UpdateUser(string login, string fullName, string email, string
description)
  {
    int rowsAffected = -1;
    using (DeptEntities db = new DeptEntities())
    {
      // user should exist in the database in order to be updated
      User found = db.Users.FirstOrDefault(u => u.Login == login);
      if (found == null) return false;
      // except login name, all other properties of a user can be changed
      found.FullName = fullName; found.Email = email; found.Description = description;
      if (ModelState.IsValid)
      {
        rowsAffected = db.SaveChanges();
      }
    }
    // there should only be one user updated after running this update (1 row at a
time)
    return rowsAffected == 1;
  }

  public bool InsertUser(string login, string fullName, string email, string
description)
  {
    int rowsAffected = -1;
    using (DeptEntities db = new DeptEntities())
    {
      // login name should be unique
      User found = db.Users.FirstOrDefault(u => u.Login == login);
      if (found != null)
      {
        string exceptionMessage = string.Format("Login name should be unique. There is
an existing user with the login name of {0}", login);
        throw new InvalidOperationException(exceptionMessage);
      }
      // create new User
      var user = new User()
      {
        Login = login,
        FullName = fullName,
        Email = email,
        Description = description
      };
      // add user to model, then commit changes
      if (ModelState.IsValid)
      {
        db.Users.AddObject(user);
        rowsAffected = db.SaveChanges();
      }
    }
    return rowsAffected == 1;
  }
  public bool DeleteUser(string login)
  {
    int rowsAffected = -1;
    using (DeptEntities db = new DeptEntities())
    {
      User found = db.Users.FirstOrDefault(u => u.Login == login);
      if (found != null)
      {
        db.Users.DeleteObject(found);
        rowsAffected = db.SaveChanges();
      }
    }
    return rowsAffected == 1;
  }
```

VB

```
Public Function GetUsers() As IQueryable(Of User)
 Dim db As New DeptEntities()
 Return db.Users.AsQueryable()
End Function

Public Function UpdateUser(login As String, fullName As String, email As String,
description As String) As Boolean
 Dim rowsAffected As Integer = -1
 Using db As New DeptEntities()
  ' user should exist in the database in order to be updated
  Dim found As User = db.Users.FirstOrDefault(Function(u) u.Login = login)
  If found Is Nothing Then
   Return False
  End If
  ' except login name, all other properties of a user can be changed
  found.FullName = fullName
  found.Email = email
  found.Description = description
  If ModelState.IsValid Then
   rowsAffected = db.SaveChanges()
  End If
 End Using
 ' there should only be one user updated after running this update (1 row at a time)
 Return rowsAffected = 1
End Function
Public Function InsertUser(login As String, fullName As String, email As String,
description As String) As Boolean
 Dim rowsAffected As Integer = -1
 Using db As New DeptEntities()
  ' login name should be unique
  Dim found As User = db.Users.FirstOrDefault(Function(u) u.Login = login)
  If found IsNot Nothing Then
   Dim exceptionMessage As String = String.Format("Login name should be unique. There
is an existing user with the login name of {0}", login)
   Throw New InvalidOperationException(exceptionMessage)
  End If
  ' create new User
  Dim user = New User() With { _
   .Login = login, _
   .FullName = fullName, _
   .Email = email, _
   .Description = description _
  }
  ' add user to model, then commit changes
  If ModelState.IsValid Then
   db.Users.AddObject(user)
   rowsAffected = db.SaveChanges()
  End If
 End Using
 Return rowsAffected = 1
End Function
```

```
Public Function DeleteUser(login As String) As Boolean
 Dim rowsAffected As Integer = -1
 Using db As New DeptEntities()
  Dim found As User = db.Users.FirstOrDefault(Function(u) u.Login = login)
  If found IsNot Nothing Then
   db.Users.DeleteObject(found)
   rowsAffected = db.SaveChanges()
  End If
 End Using
 Return rowsAffected = 1
End Function
```

Setting the Cell Types for Bound Data

The bound data may be any of several types and Spread provides different cell types to display that data effectively. Most often, you can use the **DataAutoCellTypes ('DataAutoCellTypes Property' in the on-line documentation)** property of the sheet to allow Spread to automatically match the best cell type with the cells of data. With this property you can turn off the automatic assignment if in certain cases you want to control the cell type. For example, if you have a column of 1's and 0's but you want to treat them as check box settings (checked and unchecked) instead of as numbers, then you can turn off automatic assignment and assign the check box cell type to that column of cells.

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

Using the Properties Window

1. At design time, in the **Properties** window, select the FpSpread component.
2. Select the **Sheets** property.
3. Click the button to display the **SheetView Collection Editor**.
4. Set the **DataAutoCellTypes** property.
5. Select **OK**.

Using a Shortcut

Set the **DataAutoCellTypes ('DataAutoCellTypes Property' in the on-line documentation)** property.

Example

This example code sets the **DataAutoCellTypes ('DataAutoCellTypes Property' in the on-line documentation)** property.

C#

```
FpSpread1.Sheets[0].DataAutoCellTypes = true;
```

VB

```
FpSpread1.Sheets(0).DataAutoCellTypes = True
```

Using Code

1. Create a data source.
2. Create a check box cell column.
3. Set the **DataAutoCellTypes ('DataAutoCellTypes Property' in the on-line documentation)** property to false.

4. Set the Spread **DataSource ('DataSource Property' in the on-line documentation)** property to the data source.

Example

This example code sets the **DataAutoCellTypes ('DataAutoCellTypes Property' in the on-line documentation)** property to false and uses a check box cell for the number data.

C#

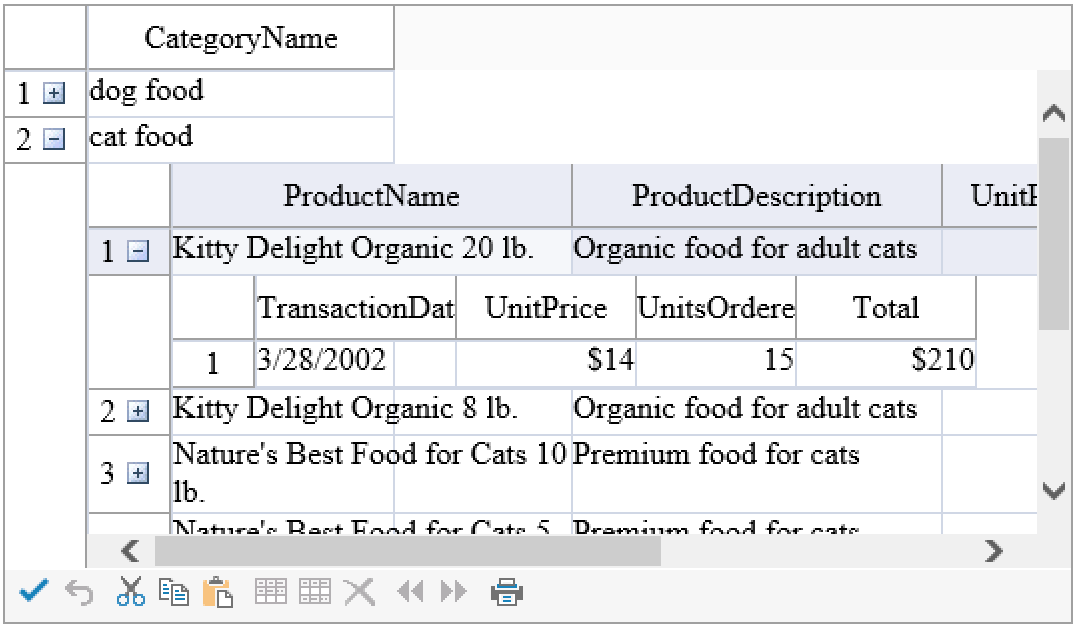
```
DataSet ds = new System.Data.DataSet();
DataTable name;
DataTable city;
name = ds.Tables.Add("Customers");
name.Columns.AddRange(new DataColumn[] {new DataColumn("LastName", typeof(string)), new
DataColumn("FirstName", typeof(string)), new DataColumn("ID", typeof(Int32))});
name.Rows.Add(new object[] { "Fielding", "William", 0 });
name.Rows.Add(new object[] { "Williams", "Arthur", 1 });
name.Rows.Add(new object[] { "Zuchini", "Theodore", 1 });
city = ds.Tables.Add("City/State");
city.Columns.AddRange(new DataColumn[] {new DataColumn("City", typeof(string)), new
DataColumn("Owner", typeof(Int32)), new DataColumn("State", typeof(string))});
city.Rows.Add(new object[] { "Atlanta", 0, "Georgia" });
city.Rows.Add(new object[] { "Boston", 1, "Mass." });
city.Rows.Add(new object[] { "Tampa", 2, "Fla." });
FpSpread1.Sheets[0].Columns[2].CellType = new FarPoint.Web.Spread.CheckBoxCellType();
FpSpread1.Sheets[0].DataAutoCellTypes = false;
FpSpread1.DataSource = ds;
```

VB

```
Dim ds As New System.Data.DataSet
Dim name As System.Data.DataTable
Dim city As System.Data.DataTable
name = ds.Tables.Add("Customers")
name.Columns.AddRange(New System.Data.DataColumn() {New
System.Data.DataColumn("LastName", Type.GetType("System.String")), New
System.Data.DataColumn("FirstName", Type.GetType("System.String")), New
System.Data.DataColumn("ID", Type.GetType("System.Int32"))})
name.Rows.Add(New Object() {"Fielding", "William", 0})
name.Rows.Add(New Object() {"Williams", "Arthur", 1})
name.Rows.Add(New Object() {"Zuchini", "Theodore", 1})
city = ds.Tables.Add("City/State")
city.Columns.AddRange(New System.Data.DataColumn() {New System.Data.DataColumn("City",
Type.GetType("System.String")), New System.Data.DataColumn("Owner",
Type.GetType("System.Int32")), New System.Data.DataColumn("State",
Type.GetType("System.String"))})
city.Rows.Add(New Object() {"Atlanta", 0, "Georgia"})
city.Rows.Add(New Object() {"Boston", 1, "Mass."})
city.Rows.Add(New Object() {"Tampa", 2, "Fla."})
FpSpread1.Sheets(0).Columns(2).CellType = New FarPoint.Web.Spread.CheckBoxCellType()
FpSpread1.Sheets(0).DataAutoCellTypes = False
FpSpread1.DataSource = ds
```

Displaying Data as a Hierarchy

You can display relational data, such as from a relational database, on a sheet in hierarchies. The following figure shows an example of how you can display the data from the database provided for the tutorials.



	CategoryName
1 ⊞	dog food
2 ⊟	cat food

	ProductName	ProductDescription	UnitI
1 ⊟	Kitty Delight Organic 20 lb.	Organic food for adult cats	

	TransactionDat	UnitPrice	UnitsOrdere	Total
1	3/28/2002	$14	15	$210

	ProductName	ProductDescription	UnitI
2 ⊞	Kitty Delight Organic 8 lb.	Organic food for adult cats	
3 ⊞	Nature's Best Food for Cats 10 lb.	Premium food for cats	
	Nature's Best Food for Cats 5	Premium food for cats	

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 component to display the data as you want. This is the procedure described in the examples that follow.

Properties such as **EditModePermanent ('EditModePermanent Property' in the on-line documentation)** and **EditModeReplace ('EditModeReplace Property' in the on-line documentation)** only apply to the parent Spread and do not apply to the child sheets unless you set them in the **ChildViewCreated ('ChildViewCreated Event' in the on-line documentation)** event.

- The hierarchical display of data, of displaying sheets within cells of a spread sheet, can fill up the visible part of the component quickly. In order to let you minimize the amount of hierarchical overhead, that is the amount of space taken by the higher levels of the hierarchy, you can collapse the hierarchy using the display of the hierarchy bar. For more information about the hierarchy bar, refer to **Customizing the Hierarchy Bar**.
- For more information on Outlook-style grouping for a hierarchical display of data, refer to **Customizing Grouping of Rows of User Data**.
- For more information on row expansion, refer to **Handling Row Expansion**.

The **SpreadImage ('SpreadImage Class' in the on-line documentation)** class is not supported in the **ChildViewCreated ('ChildViewCreated Event' in the on-line documentation)** event.

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 **DataSource ('DataSource Property' in the on-line documentation)** or the sheet **DataSource ('DataSource Property' in the on-line documentation)** property equal to the data set.
4. Provide code in the FpSpread component's **ChildViewCreated ('ChildViewCreated Event' in the on-line documentation)** event for displaying the parent and child views of the data.

Example

This example binds the component to a data set that contains multiple related tables from a database and sets up the component to display the data in hierarchies. This 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.NET 11\docs\Windows Forms\TutorialFiles. This assumes that before this code you have an include statement:

Visual Basic

```
Imports System.Data.OleDb
```

And here is the code:

Visual Basic

```
Protected Sub Page_Load(sender As Object, e As EventArgs) Handles Me.Load
      If IsPostBack Then Return
      End If
      ' Call subroutines to set up data and format the Spread component
      InitData()
      FormatSpread()
End Sub
Private Sub InitData()
        Dim con As New OleDbConnection()
        Dim cmd As New OleDbCommand()
        Dim da As New OleDbDataAdapter()
        Dim ds As New System.Data.DataSet()
        Dim dt As System.Data.DataTable
con.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=C:\\Program
Files\GrapeCity\Spread.NET 11\Windows Forms\TutorialFiles\databind.mdb"
con.Open()
With cmd
.Connection = con
.CommandType = System.Data.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
.Columns(0).Width = 200
End With
End Sub

Protected Sub FpSpread1_ChildViewCreated(sender As Object, e As
FarPoint.Web.Spread.CreateChildViewEventArgs) Handles FpSpread1.ChildViewCreated
Dim dateType As New FarPoint.Web.Spread.DateTimeCellType()

If e.SheetView.RelationName = "Root" Then
With e.SheetView
.DataAutoCellTypes = 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.Web.Spread.CurrencyCellType()
.Columns(7).CellType = New FarPoint.Web.Spread.CheckBoxCellType()
End With
Else
With e.SheetView
.DataAutoCellTypes = 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.Web.Spread.CurrencyCellType()
.Columns(1).CellType = dateType
'Add a total column
.ColumnCount = .ColumnCount + 1
.ColumnHeader.Cells(0, .ColumnCount - 1).Value = "Total"
.Columns(.ColumnCount - 1).CellType = New FarPoint.Web.Spread.CurrencyCellType()
.Columns(.ColumnCount - 1).Formula = "F1*G1"
End With
End If
End Sub
```

Handling Row Expansion

With a hierarchical display of data, as discussed in **Displaying Data as a Hierarchy**, users can be allowed to expand the rows that have more data or they can be prevented from expanding those rows. Use the **GetRowExpandable ('GetRowExpandable Method' in the on-line documentation)** and **SetRowExpandable ('SetRowExpandable Method' in the on-line documentation)** properties of the sheet (or in the sheet models) to control the ability of users to expand the rows.

You can customize the icons for expanding and collapsing hierarchies. For more information, refer to **Customizing the Graphical Interface**.

Using a Shortcut

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

Example

This example code sets the **SetRowExpandable ('SetRowExpandable Method' in the on-line documentation)** method.

C#

```
FpSpread1.Sheets[0].DataSource = ds;
FpSpread1.Sheets[0].SetRowExpandable(0, false);
```

VB

```
FpSpread1.Sheets(0).DataSource = ds
FpSpread1.Sheets(0).SetRowExpandable(0, False)
```

Adding an Unbound Row

You can add an unbound row of cells to a sheet in a component that is bound to a data source.

Using a ShortCut

1. Bind the data.
2. Use the **AddUnboundRows ('AddUnboundRows Method' in the on-line documentation)** method after adding any data to the data set.
3. Use the **AddRowToDataSource ('AddRowToDataSource Method' in the on-line documentation)** method to add the row to the data source.

Example

This example code adds an unbound row, adds data to the row, and then adds the row to the data source.

C#

```
private void FpSpread1ChildViewCreated(object sender,
FarPoint.Web.Spread.CreateChildViewEventArgs e)
{
FarPoint.Web.Spread.SheetView sv;
sv = e.SheetView;
sv.AddUnboundRows(0, 1);
sv.Cells[0, 0].Value = "Dallas";
sv.Cells[0, 1].Value = "0";
sv.Cells[0, 2].Value = "Texas";
sv.AddRowToDataSource(0, true);
}
```

VB

```
Private Sub FpSpread1ChildViewCreated(ByVal sender As Object, ByVal e As
FarPoint.Web.Spread.CreateChildViewEventArgs) Handles FpSpread1.ChildViewCreated
Dim sv As FarPoint.Web.Spread.SheetView
sv = e.SheetView
sv.AddUnboundRows(0, 1)
sv.Cells(0, 0).Value = "Dallas"
sv.Cells(0, 1).Value = "0"
sv.Cells(0, 2).Value = "Texas"
sv.AddRowToDataSource(0, True)
```

```
End Sub
```

Limiting Postbacks When Updating Bound Data

If your data set is getting lost when you click the **Update** button, make sure that you have code in your page load event so that you only re-create the bound data when you are loading for the first time and not overwriting it on each post back.

Using Code

Use the **IsPostBack** property.

Example

In this example, the if-endif structure surrounding the **DataBind ('DataBind Method' in the on-line documentation)** method restricts the method from being run on each post back.

C#

```
if !(Page.IsPostBack)
    {
        ... other code ...
        FpSpread1.DataBind();
     }
```

VB

```
If Not Page.IsPostBack Then
        ... other code ...
        FpSpread1.DataBind()
End If
```

Tutorial: Binding to a Corporate Database

The following tutorials walk you through creating an ASP.NET project in Visual Studio .NET using Spread for ASP.NET and binding to a database.

- **Using Spread with Visual Studio 2012 and the SQL Data Source**
- **Using Spread with the AccessDataSource Control**

The following tutorial walks you through creating an ASP.NET project in Visual Studio .NET using Spread for ASP.NET. By binding to a corporate database, you will learn how to set up a database connection and bind the spreadsheet to a data source.

The Microsoft Jet 4.0 driver is not supported on 64-bit processes.

This tutorial uses an earlier version of Visual Studio. In this tutorial, the major steps are:

- **Adding Spread to a DataBind Project**
- **Setting up the Database Connection**
- **Specifying the Data to Use**
- **Creating the Data Set**
- **Binding Spread to the Database**
- **Improving the Display by Changing the Cell Type**

Using Spread with Visual Studio 2012 and the SQL Data Source

Later versions of Visual Studio have data source controls that require fewer steps. This list of steps uses the SQLDataSource control to bind the Spread control.

1. Start a new Visual Studio .NET project.
2. Name the project databind.
3. Name the form in the project binding.aspx.
4. Add the FpSpread component to your project.
5. Place the component on the form.
6. If the **Toolbox** is not displayed, choose **Toolbox** from the **View** menu.
7. Double-click the SqlDataSource control (under the **Data** section in the toolbox) to place it on the form.
8. Select **Configure Data Source** in the pop-up menu. Select **New Connection**.
9. Choose **Microsoft Access Database File** in the **Choose Data Source** dialog.
10. Browse to the data file. Select the fpnorthwinds.mdb file installed in the Spread.NET\Common folder. Click **Next**.
11. Choose whether to save the connection string in the application configuration file.
12. Select **Specify columns from a table or view** (or use a stored procedure).
13. Select **Customers** under the **Name:** drop-down.
14. Select **ContactName** and **Phone** in the **Columns:** section.
15. Select **Next** and **Finish**. The **Test Query** button can be used to test the connection before selecting **Finish**.
16. Bind the data source to Spread by adding the following code to the form:

    C#

    ```
    FpSpread1.DataSource = SqlDataSource1;
    ```

    Visual Basic

    ```
    FpSpread1.DataSource = SqlDataSource1
    ```

17. Run the project to see the results.

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

Using Spread with the AccessDataSource Control

Later versions of Visual Studio have data source controls that require fewer steps. This list of steps uses the AccessDataSource control to bind the Spread control.

1. Start a new Visual Studio .NET project.
2. Name the project databind.
3. Name the form in the project binding.aspx.
4. Add the FpSpread component to your project.
5. Place the component on the form.
6. If the **Toolbox** is not displayed, choose **Toolbox** from the **View** menu.
7. Double-click the AccessDataSource control (under the **Data** section in the toolbox) to place it on the form.
8. Select **Configure Data Source** in the pop-up menu and browse to the data file (mdb file). You may need to add the mdb file to the project to see it in the browse dialog.
9. Click **Next**.
10. Select **Specify columns from a table or view**.
11. Select **Products** under the **Name:** drop-down.

12. Select **ProductName**, **ProductDescription**, **UnitPrice**, and **LeadTime** in the **Columns:** section.
13. Select **Next** and **Finish**. The **Test Query** button can be used to test the connection before selecting **Finish**.
14. Bind the data source to Spread by adding the following code to the form:

    C#

    ```
    FpSpread1.Sheets[0].DataSource = AccessDataSource1;
    ```

    Visual Basic

    ```
    FpSpread1.Sheets(0).DataSource = AccessDataSource1
    ```

15. Run the project to see the results.

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

Adding Spread to a DataBind Project

Use the following steps to create a project and add the Spread control to the form.

1. Start a new Visual Studio .NET project.
2. Name the project databind.
3. Name the form in the project binding.aspx.
4. Add the FpSpread component to your project.
5. 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 Spread to a Project**

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.

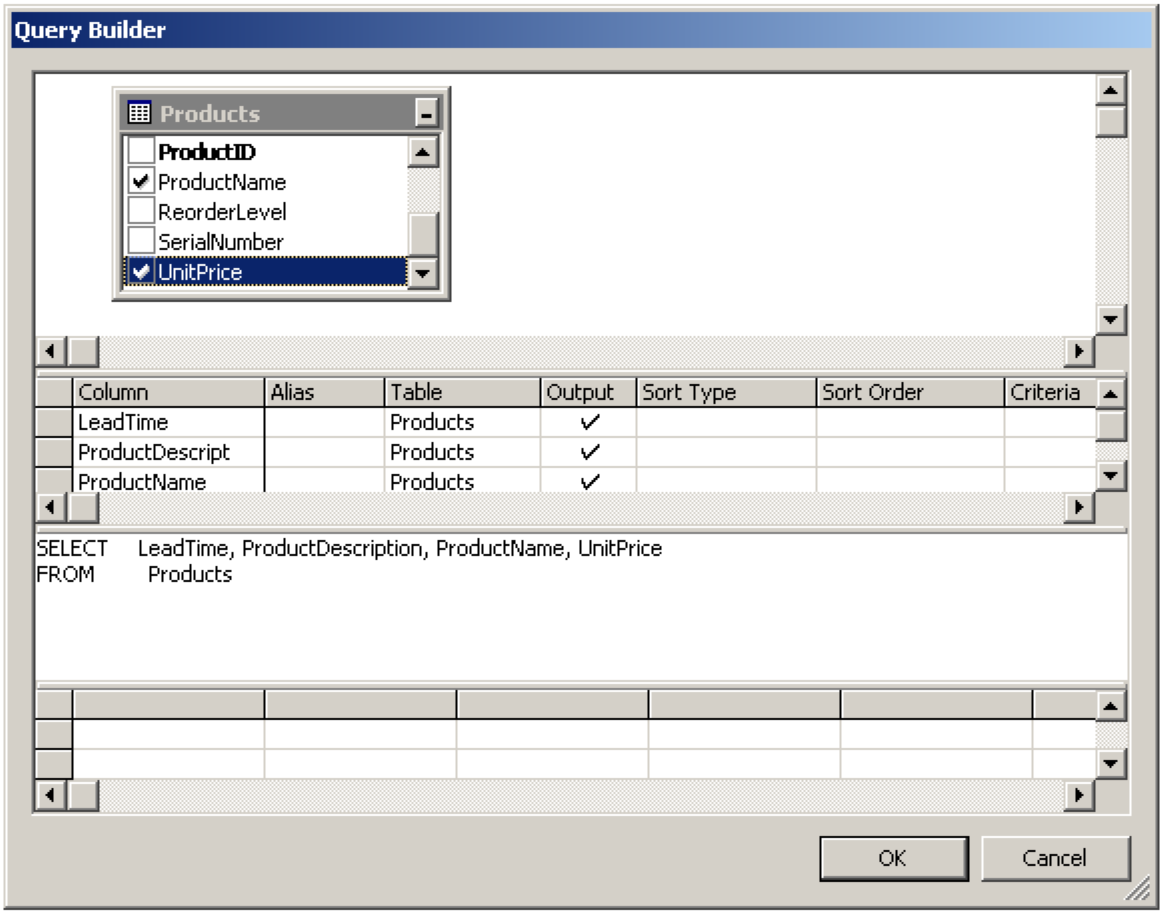
1. If the **Toolbox** is not displayed, from the **View** menu choose **Toolbox**.
2. Click the **Data** tab to display the available data controls.
3. Double-click the OleDbConnection control to add it to your form.
   The OleDbConnection control is added to your form, in a new area created below the 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 \Program Files\GrapeCity\Spread.NET 11\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.

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 component. 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
10. The Query Builder creates your SQL query in the status box. Your dialog should look like this:



□ **ProductID**
☑ ProductName
□ ReorderLevel
□ SerialNumber
☑ UnitPrice

Column	Alias	Table	Output	Sort Type	Sort Order	Criteria
LeadTime		Products	✓			
ProductDescript		Products	✓			
ProductName		Products	✓			

```
SELECT    LeadTime, ProductDescription, ProductName, UnitPrice
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.

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 component.

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 DataSet control to add it to your form.
4. In the **Add Dataset** dialog, select **Untyped** dataset and choose **OK**.
5. The DataSet control is added in the area below the visible area of the form.
6. Press F4 to display the **Properties** window for the DataSet control.
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. Select the line

C#

```
// Put user code to initialize the page here.
```

Visual Basic

```
' Put user code to initialize the page here.
```

and type the following code to replace it:

C#

```
DataSet ds;
ds = dbDataSet;
dbAdapt.Fill(ds);
```

Visual Basic

```
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.

Binding Spread to the Database

Your data set is ready, now you need to add the Spread component to display the data, and provide code to bind the Spread component to the data set.

1. Double-click on the form to open the code window.
2. Type the following code below the code you added to create the data set:

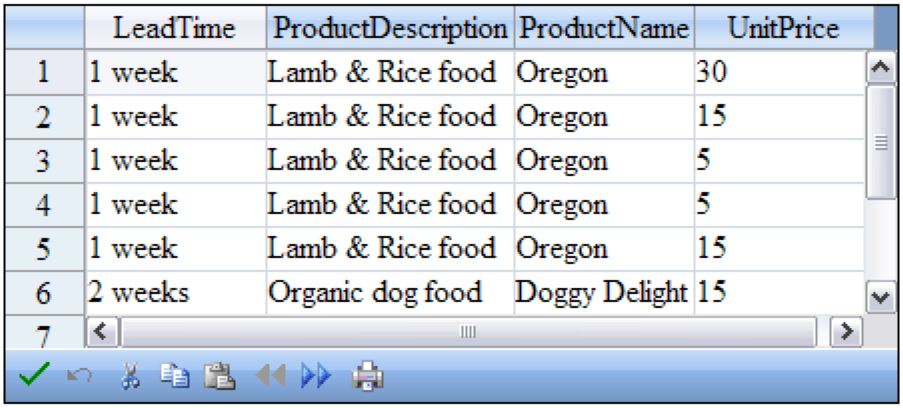
C#

```
FarPoint.Web.Spread.Model.DefaultSheetDataModel model = new
FarPoint.Web.Spread.Model.DefaultSheetDataModel(dbDataSet);
FpSpread1.Sheets[0].DataModel = model;
```

Visual Basic

```
Dim model As FarPoint.Web.Spread.Model.DefaultSheetDataModel = New
FarPoint.Web.Spread.Model.DefaultSheetDataModel(dbDataSet)
FpSpread1.Sheets(0).DataModel = model
```

3. Save your project.
4. Run your project and you should see a form that looks similar to the following:



	LeadTime	ProductDescription	ProductName	UnitPrice
1	1 week	Lamb & Rice food	Oregon	30
2	1 week	Lamb & Rice food	Oregon	15
3	1 week	Lamb & Rice food	Oregon	5
4	1 week	Lamb & Rice food	Oregon	5
5	1 week	Lamb & Rice food	Oregon	15
6	2 weeks	Organic dog food	Doggy Delight	15
7				

5. If your form does not look similar to this form, adjust the size of your Spread component, and re-check the steps

you have performed so far.

6. Stop the project.

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.

1. Double-click on the form to open the code window.
2. Set the cell type for the UnitPrice column by adding the following code below the code you have already added:

C#

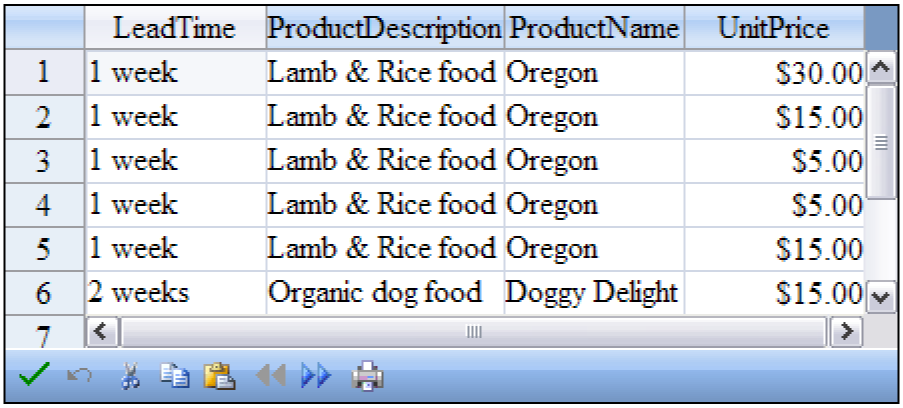
```
FarPoint.Web.Spread.StyleInfo style = new FarPoint.Web.Spread.StyleInfo();
FarPoint.Web.Spread.CurrencyCellType curPrice = new
FarPoint.Web.Spread.CurrencyCellType();
curPrice.FixedPoint = true;
style.CellType = curPrice;
style.HorizontalAlign = HorizontalAlign.Right;
style.VerticalAlign = VerticalAlign.Middle;
FpSpread1.ActiveSheetView.SetStyleInfo(-1, 3, style);
```

Visual Basic

```
Dim style As FarPoint.Web.Spread.StyleInfo
style = New FarPoint.Web.Spread.StyleInfo()
Dim curPrice As New FarPoint.Web.Spread.CurrencyCellType()
curPrice.FixedPoint = True
style.CellType = curPrice
style.HorizontalAlign = HorizontalAlign.Right
style.VerticalAlign = VerticalAlign.Middle
FpSpread1.ActiveSheetView.SetStyleInfo(-1, 3, style)
```

3. Save your project.

Run your project and you should see a form that looks similar to the following:



	LeadTime	ProductDescription	ProductName	UnitPrice
1	1 week	Lamb & Rice food	Oregon	$30.00
2	1 week	Lamb & Rice food	Oregon	$15.00
3	1 week	Lamb & Rice food	Oregon	$5.00
4	1 week	Lamb & Rice food	Oregon	$5.00
5	1 week	Lamb & Rice food	Oregon	$15.00
6	2 weeks	Organic dog food	Doggy Delight	$15.00
7				

Your bound Spread component is complete! You have completed this tutorial.

Managing Data in the Component

The tasks involved with handling data include these:

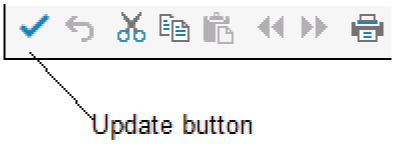
- **Saving Data to the Server**
- **Placing and Retrieving Data**
- **Server-Side Scripting**

For information on loading data from files and saving data to files, refer to **Managing File Operations**.

For information about working with bound data, refer to **Managing Data Binding**.

Saving Data to the Server

You must save changes from the client in order to update the data on the server. Changes from the client can be saved either by using the **SaveChanges ('SaveChanges Method' in the on-line documentation)** method in code or by the user clicking the **Update** button on the command bar.



Be sure to display the command bar to allow the **Update** button to be displayed. For more information about the command bar, refer to **Customizing the Command Buttons**.

Placing and Retrieving Data

You can add and return data for the component. How you add data or return it depends on whether you want to work with formatted data, which might include formatting characters, or unformatted data, and whether you are adding or returning data for a range of cells or an individual cell. You can use sheet methods and cell properties to work with formatted or unformatted data.

Read the following sections for more information and instructions:

- **Handling Data Using Sheet Methods**
- **Handling Data Using Cell Properties**

You can place (set) data in cells and retrieve (get) the data. For more information, refer to **Understanding How Cell Types Display Data**.

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, for example, in a text box, you will probably want to add the data as string data that is parsed by the FpSpread component. If you are adding several values and want to add them directly to the data model, you can add them as objects.

Formatted data usually includes information that denotes the context of the data. Unformatted data does not include additional information and might require a specific format to convey meaning. For example, formatted currency data might include currency and separator characters to indicate monetary value, as in $1,025.34. Unformatted data would not include the currency and separator characters, only the numeric value, as in 1025.34.

The following table summarizes the ways you can add data using methods at the sheet level.

Data Description	How Many Cells	Method
As a string with formatting (for example "$1,234.56")	Individual cell	**GetText ('GetText Method' in the on-line documentation)** **SetText ('SetText Method' in the on-line documentation)**
	Range of cells	**GetClip ('GetClip Method' in the on-line documentation)** **SetClip ('SetClip Method' in the on-line documentation)**
As a string without formatting (for example "1234.45")	Individual cell	**GetValue ('GetValue Method' in the on-line documentation)** **SetValue ('SetValue Method' in the on-line documentation)**
	Range of cells	**GetClipValue ('GetClipValue Method' in the on-line documentation)** **SetClipValue ('SetClipValue Method' in the on-line documentation)**
As a data object with formatting	Range of cells	**GetArray ('GetArray Method' in the on-line documentation)** **SetArray ('SetArray Method' in the on-line documentation)**

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 ('Cell Class' in the on-line documentation)** 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 model SetArray method.

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 Data**.

You can write HTML tags in cells using the Text property. The following code allows you to add HTML code into a cell by setting the **EncodeValue ('EncodeValue Property' in the on-line documentation)** property of Spread:

VB

```
FpSpread1.EncodeValue = False
FpSpread1.Cells(0, 0).Text = "<a href='http://www.componentone.com'>GrapeCity</a>"
```

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 a Shortcut

Place formatted string data using the Sheet **SetClip ('SetClip 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, 2,
2,"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, 2, 2, "Sunday" + vbTab + "Monday" + vbTab + "Tuesday"
+ vbCrLf + "Wednesday" + vbTab + "Thursday" + vbTab + "Friday" + vbCrLf + "Saturday" +
vbTab + "Sunday" + vbTab + "Monday")
```

Using Code

Add formatted string data by calling the SheetView object **SetClip ('SetClip 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.Web.Spread.SheetView newsheet=new FarPoint.Web.Spread.SheetView();
// Add data to cells A1 through C3.
newsheet.SetClip(0, 0, 2, 2, "Sunday\tMonday\tTuesday\r\nWednesday\tThursday\tFriday
\r\nSaturday\tSunday\tMonday");
// Assign the SheetView object to a sheet in the component.
fpSpread1.Sheets[0] = newsheet;
```

VB

```
' Create a new SheetView object.
Dim newsheet As New FarPoint.Web.Spread.SheetView()
' Add data to cells A1 through C3.
newsheet.SetClip(0, 0, 2, 2, "Sunday" + vbTab + "Monday" + vbTab + "Tuesday" + vbCrLf +
"Wednesday" + vbTab + "Thursday" + vbTab + "Friday" + vbCrLf + "Saturday" + vbTab +
"Sunday" + vbTab + "Monday")
' Assign the SheetView object to a sheet in the component.
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 Class 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)**

Using Code

Set the **Text ('Text Property' in the on-line documentation)** property.

Example

This example code sets the **Text ('Text Property' in the on-line documentation)** property for the cell.

C#

```
FpSpread1.Sheets[0].Cells[0,0].Text = "test";
```

VB

```
FpSpread1.Sheets(0).Cells(0,0).Text = "test"
```

Server-Side Scripting

You can handle server-side scripting by performing these tasks:

- **Understanding Effects of Client-Side Validation**
- **Understanding Postback and Page Load Events**
- **Understanding the Effect of Mode on Events**

Understanding Effects of Client-Side Validation

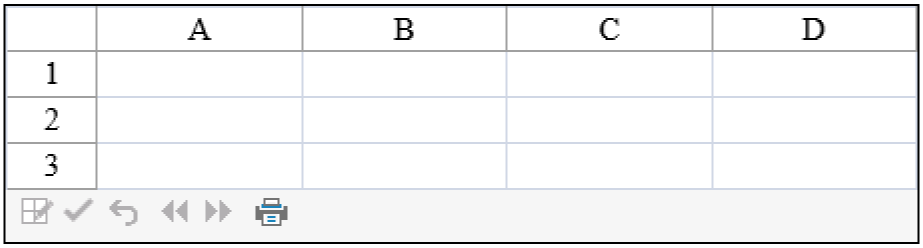
To enable client-side validation, you use the **EnableClientScript ('EnableClientScript Property' in the on-line documentation)** property. But setting this property affects many aspects of the FpSpread component, including its appearance and what events occur. By setting the **EnableClientScript ('EnableClientScript Property' in the on-line documentation)** property to false, the component produces pages that are similar to the pages the component produces for down-level browsers.

The following table describes the effect the setting of the **EnableClientScript ('EnableClientScript Property' in the on-line documentation)** property has on the component.

Feature	EnableClientScript Property Setting and Effect
Command bar	When it is set to false, the command bar buttons change to display an Edit button and the Cut, Copy, and Paste buttons are not displayed.

buttons	
Control size	When it is set to false, the component automatically sizes to the size of the page. For example, if the sheet has only two columns and two rows, the component displays only those two columns and rows, and no gray area.
Events	The events that occur for the **FpSpread ('FpSpread Class' in the on-line documentation)** class are affected by the setting of this property, the **AutoPostBack ('AutoPostBack Property' in the on-line documentation)** property, and the SheetView class’s **OperationMode ('OperationMode Property' in the on-line documentation)** property as listed in the table in **Understanding the Effect of Mode on Events**.
Message row display	If it is set to true, the component displays error messages as pop-ups. If it is set to false, the component adds an extra row below the row with the error, in which to display the error message. The additional row is not numbered.
Selection display	If it is set to true, the selection text color is not displayed, but the row or column header displays the selection background color. If it is set to false, the selection text color is displayed, but the row or column header do not display the selection background color.
Styles (and cell editor)	If it is set to false when using a custom style (a **NamedStyle ('NamedStyle Class' in the on-line documentation)** object), you should set the **Parent ('Parent Property' in the on-line documentation)** property of the **NamedStyle ('NamedStyle Class' in the on-line documentation)** to DataAreaDefault because the style has no editor and so no cells are editable. Set the **Parent ('Parent Property' in the on-line documentation)** property to use DataAreaDefault as the parent style so that the GeneralCellType editor is inherited from the DataAreaDefault and the cells are thus editable.

The following image displays the control with the **EnableClientScript ('EnableClientScript Property' in the on-line documentation)** property set to false.



	A	B	C	D
1				
2				
3				

Understanding Postback and Page Load Events

When you run an ASP.NET application, the FpSpread component on the server outputs its data and settings in HTML, which is sent to the client along with scripting if the **EnableClientScript ('EnableClientScript Property' in the on-line documentation)** property is set to true and the client’s browser can handle or is set to allow scripting.

Code you have added to the **PageLoad** event in your application occurs at the time the HTML page is sent from the server, and every time the page is sent from the server. Therefore, it is important to understand that every action that you provide or that users perform that initiates a postback to the server loads a new page into the browser. Any code you have in the **PageLoad** event is re-executed at the time the new page is sent.

For example, if you set data in a cell in the **PageLoad** event, such as setting cell A1 to the value 100, then the user changes the value in the cell, then clicks a button, the page posts back to the server, and the **PageLoad** event resets the value of the cell to 100. The user’s data is lost.

To deal with this chain of events in the page loads, you should add code to the **PageLoad** event that checks if the page is a postback or the initial load of the page.

Use a single line in the **PageLoad** event.

Example

Use the following code in the **PageLoad** event to prevent inadvertent changes to pages due to post backs:

C#

```
if (this.IsPostBack) return;
```

VB

```
If (IsPostBack) Then
  Return
End If
```

Understanding the Effect of Mode on Events

The setting of the **EnableClientScript ('EnableClientScript Property' in the on-line documentation)** property affects many aspects of the FpSpread component, including its appearance and what events occur, depending on the operation mode.

Events that are not listed in the following table are not affected by the operation mode.

The events that occur for the FpSpread class are affected by the **EnableClientScript ('EnableClientScript Property' in the on-line documentation)** setting and the SheetView class **OperationMode ('OperationMode Property' in the on-line documentation)** property as listed in the following table.

Event	Normal	ReadOnly	RowMode	SingleSelect
ActiveRowChanged ('ActiveRowChanged Event' in the on-line documentation)	Occurs if EnableClient-Script is false	Does not occur	Occurs if EnableClient-Script is false	Occurs if EnableClient-Script is false
ActiveSheetChanged ('ActiveSheetChanged Event' in the on-line documentation)	Occurs	Occurs	Occurs	Occurs
ButtonCommand ('ButtonCommand Event' in the on-line documentation)	Occurs	Occurs	Occurs	Occurs
CancelCommand ('CancelCommand Event' in the on-line documentation)	Occurs	Does not occur	Occurs	Does not occur
CellClick ('CellClick Event' in the on-line documentation)	Occurs if AutoPostBack is true	Occurs if AutoPostBack is true	Occurs if AutoPostBack is true	Occurs if AutoPostBack is true
ChildViewCreated ('ChildViewCreated Event' in the on-line documentation)	Occurs	Occurs	Occurs	Occurs
ColumnHeaderClick ('ColumnHeaderClick Event' in the on-line documentation)	Occurs if AutoPostBack is true	Occurs if AutoPostBack is true	Occurs if AutoPostBack is true	Occurs if AutoPostBack is true
CreateButton ('CreateButton Event' in the on-line documentation)	Occurs	Occurs	Occurs	Occurs
DeleteCommand ('DeleteCommand Event' in the on-line documentation) (AllowDelete set to true)	Occurs	Does not occur	Occurs	Does not occur
EditCommand ('EditCommand Event' in the on-line documentation)	Occurs if EnableClient-Script is false	Does not occur	Occurs if EnableClient-Script is false	Does not occur

ErrorCommand ('ErrorCommand Event' in the on-line documentation)	Occurs if EnableClient-Script is false	Does not occur	Occurs if EnableClient-Script is false	Does not occur
InsertCommand ('InsertCommand Event' in the on-line documentation) (AllowInsert set to true)	Occurs	Does not occur	Occurs	Does not occur
RowHeaderClick ('RowHeaderClick Event' in the on-line documentation)	Occurs if AutoPostBack is true	Occurs if AutoPostBack is true	Occurs if AutoPostBack is true	Occurs if AutoPostBack is true
SaveOrLoadSheetState ('SaveOrLoadSheetState Event' in the on-line documentation)	Occurs	Occurs	Occurs	Occurs
SortColumnCommand ('SortColumnCommand Event' in the on-line documentation) (AllowSort set to true)	Occurs	Occurs	Occurs	Occurs
TopRowChanged ('TopRowChanged Event' in the on-line documentation) (does not occur if scroll bar is used to change the top row)	Occurs	Occurs	Occurs	Occurs
UpdateCommand ('UpdateCommand Event' in the on-line documentation)	Occurs	Does not occur	Occurs	Does not occur

Managing Formulas

The following topics provide information about creating and using formulas.

- **Placing a Formula in Cells**
- **Specifying a Cell Reference Style in a Formula**
- **Using a Circular Reference in a Formula**
- **Nesting Functions in a Formula**
- **Finding a Value with Goal Seeking**
- **Recalculating and Updating Formulas Automatically**
- **Creating a Custom Function**
- **Creating a Custom Name**
- **Using the Formula Extender Control (on-line documentation)**

For more information about formulas, refer to the Formula Reference.

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. The formula can use cell and cross-sheet references.

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 addressing. For examples of formulas that use cell references, refer to **Specifying a Cell Reference Style in a Formula**.

You can add a formula by specifying the **Formula ('Formula Property' in the on-line documentation)** 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 **Adding Formulas to Cells**.

Be careful of the type of cell in which the data is found, and whether you use the **Text ('Text Property' in the on-line documentation)** or **Value ('Value Property' in the on-line documentation)** 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 NumberCellType parses a string into a double data type. When you assign the cell data using the **Value ('Value Property' in the on-line documentation)** 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.

If the **AllowUserFormulas ('AllowUserFormulas Property' in the on-line documentation)** property is true, the user can copy formulas to other cells (type = and select the formula and use Control-C to copy it).

For more information about formulas, refer to the Formula Reference.

Using a Shortcut

Add a formula to a cell, row, or column by specifying the **Formula ('Formula Property' in the on-line documentation)** 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 third column.

C#

```
FpSpread2.ActiveSheetView.Cells[2, 0].Formula = "PRODUCT(A1,5)";
FpSpread2.ActiveSheetView.Columns[3].Formula = "SUM(A1:A4)";
```

VB

```
FpSpread2.ActiveSheetView.Cells(2, 0).Formula = "PRODUCT(A1,5)"
FpSpread2.ActiveSheetView.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 puts the sum of two cells in a third cell.

C#

```
FarPoint.Web.Spread.Cell mycell;
mycell = FpSpread1.Cells[2, 0];
mycell.Formula = "SUM(A1:A2)";
```

VB

```
Dim mycell as FarPoint.Web.Spread.Cell
mycell = FpSpread1.Cells(2, 0)
mycell.Formula = "SUM(A1:A2)"
```

Example

This example shows how to use a cross-sheet reference in a formula.

C#

```
FpSpread1.Sheets.Count = 3;
FpSpread1.Sheets[0].Cells[0, 0].Formula = "sheet1!A2+sheet2!A1";
```

VB

```
FpSpread1.Sheets.Count = 3
FpSpread1.Sheets(0).Cells(0, 0).Formula = "sheet1!A2+sheet2!A1"
```

Specifying a Cell Reference Style in a Formula

Besides values, operators, and functions, a formula can contain references to values in other cells or sheets. 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 the type of cell reference for the sheet by using the **ReferenceStyle ('ReferenceStyle Property' in the on-line documentation)** property. 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). For details on the way to specify the reference style, refer to the **ReferenceStyle ('ReferenceStyle Enumeration' in the on-line documentation)** enumeration and the **SheetView ('SheetView Class' in the on-line documentation) ReferenceStyle ('ReferenceStyle Property' in the on-line documentation)** property in the **Assembly Reference (on-line documentation)**.

If you have changed the cell reference style to a style that cannot represent the formula, the 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(A2:A10)	Sums rows 2 through 10 in the first column
PI( )*C6	Pi times the value in cell C6
(A3 + B3) * C3	Adds the values in the first two cells of row 3 and multiplies the result by the value in the third cell
IF(A4>5, A4*2, A4*3)	If the contents of cell A4 are greater than 5, then multiply the contents of cell A4 by 2, or else multiply the contents of cell A4 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:** Remember that 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).

For more information on cell reference styles, refer to the Formula Reference, and the topic Cell References in a Formula.

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.Web.Spread.Model.ReferenceStyle.A1;
```

VB

```
FpSpread1.Sheets(0).ReferenceStyle = FarPoint.Web.Spread.Model.ReferenceStyle.A1
```

Using the Spread Designer

1. Select the **Settings** menu.
2. Select the **Calculation** icon under the **Sheet Settings** section.
3. Set the various formula related properties.
4. Select **OK** to close the dialog.
5. Click **Apply and Exit** to close the Spread Designer.

Using a Circular Reference in a Formula

You can refer to a formula in the cell that contains that formula. This is a circular reference. This is done typically to recursively perform a function to approach an optimum value. You can select how many times a function iterates on itself (recurses) by setting the **MaximumIterations ('MaximumIterations Property' in the on-line documentation)** property. You can also set the maximum amount of change. If the amount of change (difference between the current and previous formula result) is greater than the maximum change value, the formula continues until it reaches the maximum number of iterations or the formula result change is less than the maximum change value.

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.

For more information about formulas, refer to the Formula Reference.

Using Code

1. Set the **Iteration ('Iteration Property' in the on-line documentation)** property to true to calculate the circular reference.
2. Set the cell types for the formula.
3. Set the recalculation iteration count with the **MaximumIterations ('MaximumIterations Property' in the on-line documentation)** property for the sheet.
4. Set the reference style for the sheet.
5. Use the circular reference in a formula in a cell.

Example

This example uses a circular reference in a cell and sets the iterations.

C#

```
FpSpread1.ActiveSheetView.Iteration = true;
FpSpread1.ActiveSheetView.SetValue(0, 1, 20);
FpSpread1.ActiveSheetView.MaximumChange = 5;
FpSpread1.ActiveSheetView.MaximumIterations = 5;
FpSpread1.ActiveSheetView.SetFormula(0, 0, "B1+C1");
FpSpread1.ActiveSheetView.SetFormula(0, 2, "A1*3");
```

VB

```
FpSpread1.ActiveSheetView.Iteration = True
FpSpread1.ActiveSheetView.SetValue(0, 1, 20)
FpSpread1.ActiveSheetView.MaximumChange = 5
FpSpread1.ActiveSheetView.MaximumIterations = 5
FpSpread1.ActiveSheetView.SetFormula(0, 0, "B1+C1")
FpSpread1.ActiveSheetView.SetFormula(0, 2, "A1*3")
```

Using the Spread Designer

1. Select the **Settings** menu.
2. Select the **Calculation** icon under the **Sheet Settings** section.
3. Check the **Iteration** check box.
4. Set **Maximum Change** and **Maximum Iterations**.
5. Select **OK** to close the dialog.

6. Click **Apply and Exit** to close the Spread Designer.

Nesting Functions in a Formula

You can nest a function within another function in a formula.

For more information about formulas, refer to the Formula Reference.

Using Code

1. Specify the cell type.
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#

```
FarPoint.Web.Spread.DoubleCellType ncell = new FarPoint.Web.Spread.DoubleCellType();
FpSpread1.Sheets[0].Cells[0, 0, 2, 0].CellType = ncell;
FpSpread1.Sheets[0].Cells[0, 0, 2, 0].Value = 2;
FpSpread1.Sheets[0].Cells[3, 1].Formula = "PRODUCT(A1, SUM(A2,A3))";
```

VB

```
Dim ncell As New FarPoint.Web.Spread.DoubleCellType
FpSpread1.Sheets(0).Cells(0, 0, 2, 0).CellType = ncell
FpSpread1.Sheets(0).Cells(0, 0, 2, 0).Value = 2
FpSpread1.Sheets(0).Cells(3, 1).Formula = "PRODUCT(A1, SUM(A2,A3))"
```

Finding a Value with Goal Seeking

You can find the closest value for a cell that produces a desired formula result in another cell using the goal seeking capability.

Use the **GoalSeek ('GoalSeek Method' in the on-line documentation)** method in the **FpSpread ('FpSpread Class' in the on-line documentation)** class to find an input value that produces the desired formula result.

For more information about formulas, refer to the Formula Reference.

Using Code

Use the **GoalSeek ('GoalSeek Method' in the on-line documentation)** method to get the required value.

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)
```

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.

Also by default, the spreadsheet updates formulas when you insert or delete columns or rows or when you move or swap blocks of cells. You can turn off these automatic formula updates. However, generally, you probably want the spreadsheet to update formulas when you insert or delete columns or rows or when you move or swap blocks of cells. 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.

The **ClientAutoCalculation ('ClientAutoCalculation Property' in the on-line documentation)** property is used for automatic calculation of client-side data. The formula updates when the cell goes out of edit mode instead of waiting until the user clicks on the save changes icon. **ClientAutoCalculation ('ClientAutoCalculation Property' in the on-line documentation)** will have no effect on a hierarchy. You can set either **AutoCalculation ('AutoCalculation Property' in the on-line documentation)** or **ClientAutoCalculation ('ClientAutoCalculation Property' in the on-line documentation)** or both properties at the same time.

For more information about formulas, refer to the Formula Reference.

Using a Shortcut

Set the **AutoCalculation ('AutoCalculation Property' in the on-line documentation)** property and the **ClientAutoCalculation ('ClientAutoCalculation Property' in the on-line documentation)** property.

Example

This example sets the **AutoCalculation ('AutoCalculation Property' in the on-line documentation)** and **ClientAutoCalculation ('ClientAutoCalculation Property' in the on-line documentation)** properties.

C#

```
FpSpread1.Sheets[0].AutoCalculation = true;
```

```
FpSpread1.ClientAutoCalculation = true;
```

VB

```
FpSpread1.Sheets(0).AutoCalculation = True
FpSpread1.ClientAutoCalculation = True
```

Using the Spread Designer

1. Select the **Settings** menu.
2. Select the **Calculation** icon under the **Sheet Settings** section.
3. Check the **Automatic Calculation** check box.
4. Select the **Edit** icon under **Spread Settings** to set Client Auto Calculation.
5. Select **OK** to close the dialog.
6. Click **Apply and Exit** to close the Spread Designer.

Creating 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 custom function.

For more information about formulas, refer to the Formula Reference.

Using Code

1. Define the custom function(s).
2. Register the function(s) in the sheet using the **AddCustomFunction ('AddCustomFunction Method' in the on-line documentation)** method.
3. Use the custom function(s).

Example: Creating, Registering, and Using Three Custom Functions

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 : 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 = 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 [i.e. similar to how 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 second step is to register the custom functions as this code does.

C#

```
DefaultSheetDataModel dataModel = FpSpread1.ActiveSheetView.DataModel as
FarPoint.Web.Spread.Model.DefaultSheetDataModel;
if( dataModel != null ) {
    dataModel.AddCustomFunction(new CubeFunctionInfo());
    dataModel.AddCustomFunction(new XorFunctionInfo());
    dataModel.AddCustomFunction(new NullFunctionInfo());
}
```

The third step is to use the custom functions in formulas, as shown in this code.

C#

```
FpSpread1.ActiveSheetView.SetFormula(0, 0, "CUBE(5)");
FpSpread1.ActiveSheetView.SetFormula(1, 0, "XOR(FALSE,FALSE)");
FpSpread1.ActiveSheetView.SetFormula(1, 1, "XOR(TRUE,FALSE)");
FpSpread1.ActiveSheetView.SetFormula(1, 2, "XOR(FALSE,TRUE)");
FpSpread1.ActiveSheetView.SetFormula(1, 3, "XOR(TRUE,TRUE)");
FpSpread1.ActiveSheetView.SetFormula(2, 0, "CHOOSE(1,100,NULL(),300)");
FpSpread1.ActiveSheetView.SetFormula(2, 1, "CHOOSE(2,100,NULL(),300)");
FpSpread1.ActiveSheetView.SetFormula(2, 2, "CHOOSE(3,100,NULL(),300)");
```

Creating 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.

For more information about formulas, refer to the Formula Reference.

Using Code

Define the custom name using the **AddCustomName ('AddCustomName Method' in the on-line documentation)** method.

Example

To add a custom name, use the **AddCustomName ('AddCustomName Method' in the on-line documentation)** method as shown in this code:

C#

```
FarPoint.Web.Spread.Model.DefaultSheetDataModel d = new
FarPoint.Web.Spread.Model.DefaultSheetDataModel();
d.AddCustomName("test", "$B$1", 0, 0);
```

VB

```
Dim d FarPoint.Web.Spread.Model.DefaultSheetDataModel = New
FarPoint.Web.Spread.Model.DefaultSheetDataModel()
d.AddCustomName("test", "$B$1", 0, 0)
```

Example

To add a custom name for a value, use the **AddCustomName ('AddCustomName Method' in the on-line documentation)** method as shown in this code:

C#

```
FarPoint.Web.Spread.Model.DefaultSheetDataModel d;
d = (FarPoint.Web.Spread.Model.DefaultSheetDataModel)FpSpread1.Sheets[0].DataModel;
d.AddCustomName("alpha", "101", 0, 0);
```

VB

```
Dim d As New FarPoint.Web.Spread.Model.DefaultSheetDataModel
d = (FarPoint.Web.Spread.Model.DefaultSheetDataModel)FpSpread1.Sheets(0).DataModel
d.AddCustomName("alpha", "101", 0, 0)
```

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**

For information on saving skins, which can be saved as files, refer to **Creating a Skin for Sheets**.

Saving Data to a File

You can save the data, and for some types of files both the data and formatting, in the component to a file or stream. Spread provides methods for saving from a Spread file to several industry accepted file types including Microsoft Excel and plain text files.

Consult the following sections 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 File**
- **Saving to a PDF File**

Saving to a Spread XML File

You can save the data or the data and formatting in an FpSpread component to a Spread XML file or to a stream. When you save, all sheets in the component are saved to the file or stream or a specific sheet can be saved. If you choose to save the formatting, the data saved includes formatting characters, such as currency symbols, and other information such as cell types are also saved.

You can also save a file from inside Spread Designer.

Refer to the SheetView class **Save ('Save Method' in the on-line documentation)** methods.

For instructions for opening Spread-compatible XML files, see **Opening a Spread XML File**.

Using Code

Use the FpSpread component's **Save ('Save Method' in the on-line documentation)** method, specifying the path and file name of the Spread XML file to save or the Stream object to save, and whether to save data only or data and formatting.

Example

This example code saves the data and formatting in a component to an XML file.

C#

```
// Save the data and formatting to an XML file.
FpSpread1.Save("C:\\savefile.xml", False);
```

VB

```
' Save the data and formatting to an XML file.
FpSpread1.Save("C:\savefile.xml", False)
```

Using the Spread Designer

1. Select the **File** menu.
2. Choose the **Save** option.
   The **Save As** dialog appears.
3. For the **Save As** type, select Spread files (*.xml).
4. 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.
5. Click **OK** to close the Spread Designer.

Saving to an Excel File

You can save data to an Excel-formatted (BIFF8 format or XLSX) file or stream. There are multiple **SaveExcel ('SaveExcel Method' in the on-line documentation)** methods each with several options. For instance, you can specify whether headers are saved with the data using the setting of the **IncludeHeaders ('IncludeHeaders Enumeration' in the on-line documentation)** enumeration. Use the ExcelSaveFlags.UseOOXMLFormat with the ExcelSaveFlags enumeration to save to an XLSX format.

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. Advanced content includes items such as macros, ActiveX controls, data connections, and so on. Consider the following when using the document caching option:

- Advanced document content is preserved (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.
- To keep any document caching settings (changes would be lost during a postback), open the original file with the document caching only setting and then save the file using the document caching setting.

You can also save a file from inside Spread Designer.

The SaveExcel button on the CommandBar allows users to export a spreadsheet into an Excel file. To display this button on the CommandBar, users need to set the **ShowExcelButton ('ShowExcelButton Property' in the on-line documentation)** property in the **CommandBarInfo ('CommandBarInfo Class' in the on-line documentation)** class to true.

For more information on exporting spreadsheets to excel files, see **Working with the SaveExcel button on the CommandBar (on-line documentation)**.

For instructions for opening Excel-compatible files, see **Opening an Excel-Formatted File**.

For more information about how the data and formatting is exported to the Excel file format, see the **Import and Export Reference (on-line documentation)**.

Using Code

Use the FpSpread object's **SaveExcel ('SaveExcel Method' in the on-line documentation)** method, providing the path and file name for the file to save, or providing additional information using one of the overloaded methods.

Example

The first example saves the data in a FpSpread component to an Excel-formatted file and specifies that both row and column headers are included in the output. The second example saves to a stream.

C#

```
// Save data to Excel-formatted file, including headers.
FpSpread1.SaveExcel("C:\\excelfile.xls",
FarPoint.Web.Spread.Model.IncludeHeaders.BothCustomOnly);
// Save data to memory stream and then load in second component.
System.IO.MemoryStream s = new System.IO.MemoryStream();
FpSpread1.SaveExcel(s);
s.Position = 0;
FpSpread2.OpenExcel(s);
s.Close();
```

VB

```
' Save data to an Excel-formatted file, including headers.
FpSpread1.SaveExcel("C:\excelfile.xls",
FarPoint.Web.Spread.Model.IncludeHeaders.BothCustomOnly)
' Save data to memory stream and then load in second component.
Dim s As New System.IO.MemoryStream()
FpSpread1.SaveExcel(s)
s.Position = 0
FpSpread2.OpenExcel(s)
s.Close()
```

Using the Spread Designer

1. Select the **File** menu.
2. Choose the **Save** option.
   The **Save As** dialog appears.
3. For the **Save As** type, select Excel files (.xls or .xlsx).
4. Specify the path and file name to which to save the file, and then click **Save**.
5. 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.

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.

There are multiple **SaveTextFile ('SaveTextFile Method' in the on-line documentation)** methods each with several options. For instance, you can specify whether headers are saved with the data using the setting of the **IncludeHeaders ('IncludeHeaders Enumeration' in the on-line documentation)** enumeration.

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 spreadsheet or a portion of the spreadsheet data from the component to tab-delimited and delimited files.

You can also save a file from inside Spread Designer. The Spread Designer saves the current sheet.

For instructions for opening text files, see **Opening a Text File**.

Using a Shortcut

1. To save the entire sheet, call one of the SheetView object **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 particular method you choose.
2. To save a portion of a sheet, call one of the SheetView object **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 particular 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", false,
FarPoint.Web.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", False,
FarPoint.Web.Spread.Model.IncludeHeaders.BothCustomOnly, "#", "%", "^")
```

Using the Spread Designer

1. Select the **File** menu.
2. Choose the **Save** option.
   The **Save As** dialog appears.
3. For the **Save As** type, select Text files (.txt).
4. Specify the path and file name to which to save the file, and then click **Save**.
5. Click **OK** to close the Spread Designer.

Saving to an HTML File

You can save the data in a sheet to an HTML file or stream.

Using Code

Use the SheetView's **SaveHtml ('SaveHtml Method' in the on-line documentation)** method, specifying the path and file name of the file to save or the Stream object to save.

Example

This example code saves the data to an HTML file.

C#

```
FpSpread1.Sheets[0].Cells[0, 0].Value = 1;
FpSpread1.Sheets[0].SaveHtml("C:\\SpreadASP\\samples\\test.html");
```

VB

```
FpSpread1.Sheets(0).Cells(0,0).Value = 1
FpSpread1.Sheets(0).SaveHtml("C:\SpreadASP\samples\test.html")
```

Saving to a PDF File

You can save the spreadsheet to a Portable Document File (PDF, version 1.4) file. Each sheet is saved to a new page in the PDF file.

The following topics contain more information about saving to PDF.

- **Saving to PDF Methods**
- **Setting PDF Security Options (on-line documentation)**
- **Setting PrintInfo Class Properties**
- **Setting Smart Print Options**
- **Setting Headers and Footers**

Saving to PDF Methods

You can use one of the **SavePdf ('SavePdf Method' in the on-line documentation)** methods or one of the **SavePdfToResponse ('SavePdfToResponse Method' in the on-line documentation)** methods to save to a PDF file. The former saves the Spread control to the specified PDF file. The latter saves Spread to the specified response object in PDF format.

You can customize the appearance of the PDF file using the **PrintInfo ('PrintInfo Class' in the on-line documentation)** class settings for the component, including whether the pages in the PDF file are portrait or landscape. The **PrintInfo ('PrintInfo Class' in the on-line documentation)** class is only used when saving to a PDF file.

You can also save a file from inside Spread Designer (**File** menu, **Print**, **SaveToPDF**).

You can specify page breaks with the column **PageBreak ('PageBreak Property' in the on-line documentation)** property or the row **PageBreak ('PageBreak Property' in the on-line documentation)** property.

Using Code

Use the **SavePdf ('SavePdf Method' in the on-line documentation)** method.

Example

This example saves to PDF with the **SavePdf ('SavePdf Method' in the on-line documentation)** method.

C#

```
FpSpread1.SavePdf("c:\\test.pdf");
```

VB

```
FpSpread1.SavePdf("c:\test.pdf")
```

Using the Spread Designer

1. Select the **File** menu.
2. Choose the **Print** option.

3. Choose the **SaveToPDF** option.
4. For the **Save As** type, select PDF files (.PDF).
5. Specify the path and file name to which to save the file, and then click **Save**.
6. Click **OK** to close the Spread Designer.

Setting PrintInfo Class Properties

You can set properties for the **PrintInfo ('PrintInfo Class' in the on-line documentation)** class that are used when saving to a PDF file. You can specify titles, headers, footers, and many other options with the **PrintInfo ('PrintInfo Class' in the on-line documentation)** class.

The properties in this class are only available when saving to a PDF file.

> The ShowColumnHeader and ShowRowHeader properties in the PrintInfo class apply when printing or saving to PDF.

You can also specify smart print options. See **Setting Smart Print Options** for more information.

Using Code

Set the PrintInfo class properties and then use the **SavePdf ('SavePdf Method' in the on-line documentation)** method.

Example

This example code sets the orientation before saving to PDF.

C#

```
FarPoint.Web.Spread.PrintInfo pi = new FarPoint.Web.Spread.PrintInfo();
pi.Orientation = FarPoint.Web.Spread.PrintOrientation.Landscape;
FpSpread1.Sheets[0].PrintInfo = pi;
FpSpread1.SavePdf("c:\\test.pdf");
```

VB

```
Dim pi As New FarPoint.Web.Spread.PrintInfo()
pi.Orientation = FarPoint.Web.Spread.PrintOrientation.Landscape
FpSpread1.Sheets(0).PrintInfo = pi
FpSpread1.SavePdf("c:\test.pdf")
```

Setting Smart Print Options

Spread can 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 properties that you use to configure smart printing are part of the **PrintInfo ('PrintInfo Class' in the on-line documentation)** class. These properties only have an effect when content is saved to PDF.

The printing optimization rules, which you can turn on or off, can be customized by setting the properties of these rule objects:

Rule Object	Description
LandscapeRule	Determines whether to print the sheet in landscape or portrait orientation.

('LandscapeRule Class' in the on-line documentation)	
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 and **SmartPrintRule ('SmartPrintRule Class' in the on-line documentation)** object.

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 **UseSmartPrint ('UseSmartPrint Property' in the on-line documentation)** property to true.
4. Set the **SheetView ('SheetView Class' in the on-line documentation)** object PrintInfo 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.Web.Spread.SmartPrintRulesCollection printrules = new
FarPoint.Web.Spread.SmartPrintRulesCollection();
printrules.Add(new
FarPoint.Web.Spread.BestFitColumnRule(FarPoint.Web.Spread.ResetOption.None));
printrules.Add(new
FarPoint.Web.Spread.LandscapeRule(FarPoint.Web.Spread.ResetOption.None));
printrules.Add(new FarPoint.Web.Spread.ScaleRule(FarPoint.Web.Spread.ResetOption.All,
1.0f, .4f, .2f));
// Create a PrintInfo object and set the properties.
FarPoint.Web.Spread.PrintInfo printset = new FarPoint.Web.Spread.PrintInfo();
printset.SmartPrintRules = printrules;
printset.UseSmartPrint = true;
fpSpread1.Sheets[0].PrintInfo = printset;
// Print the sheet.
fpSpread1.SavePdf("c:\\test.pdf");
```

VB

```
' Create the print rules.
Dim printrules As New FarPoint.Web.Spread.SmartPrintRulesCollection()
printrules.Add(New
FarPoint.Web.Spread.BestFitColumnRule(FarPoint.Web.Spread.ResetOption.None))
printrules.Add(New
FarPoint.Web.Spread.LandscapeRule(FarPoint.Web.Spread.ResetOption.None))
printrules.Add(New FarPoint.Web.Spread.ScaleRule(FarPoint.Web.Spread.ResetOption.All,
1.0F, 0.4F, 0.2F))
' Create a PrintInfo object and set the properties.
Dim printset As New FarPoint.Web.Spread.PrintInfo()
printset.SmartPrintRules = printrules
printset.UseSmartPrint = True
FpSpread1.Sheets(0).PrintInfo = printset
' Print the sheet.
FpSpread1.SavePdf("C:\test.pdf")
```

Setting Headers and Footers

You can provide headers and footers that appear on the printed pages when saving to a PDF file. 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".

These settings are only available when saving to a PDF file.

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)
/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*"	Sets 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)
/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 saving to PDF.

The following list provides additional information about headers and footers:

- You can specify a color for the text from a list of colors if the color is previously defined in the **Colors ('Colors Property' in the on-line documentation)** property.
- You can specify an image if the image is previously defined in the **Images ('Images Property' in the on-line documentation)** 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.

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 Class' 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.Web.Spread.PrintInfo printset = new FarPoint.Web.Spread.PrintInfo();
printset.Colors = new Drawing.Color[] {Drawing.Color.Green, Drawing.Color.Yellow,
Drawing.Color.Gold, Drawing.Color.Indigo, Drawing.Color.Brown};
printset.Images = new System.Drawing.Image[]
{System.Drawing.Image.FromFile("C:\\images\\point.jpg"),
System.Drawing.Image.FromFile("C:\\images\\logo.gif"),
System.Drawing.Image.FromFile("C:\\images\\icon.jpg")};
printset.Header = ""/fn\"Book Antiqua\" /fz\"14\" Print job for GrapeCity 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.SavePdf("c:\\test.pdf");
```

VB

```
' Create PrintInfo object and set properties.
Dim printset As New FarPoint.Web.Spread.PrintInfo()
printset.Colors = New Drawing.Color() {Drawing.Color.Green, Drawing.Color.Yellow,
Drawing.Color.Gold, Drawing.Color.Indigo, Drawing.Color.Brown}
printset.Images = New System.Drawing.Image()
{System.Drawing.Image.FromFile("D:\images\point.jpg"),
System.Drawing.Image.FromFile("D:\images\logo.gif"),
System.Drawing.Image.FromFile("C:\images\icon.jpg")}
printset.Header = "/fn""Book Antiqua"" /fz""14"" Print job for GrapeCity 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.SavePdf("c:\test.pdf")
```

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 Class' 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.Web.Spread.PrintInfo pi = new FarPoint.Web.Spread.PrintInfo();
pi.Footer = "This is Page /p/nof /pc Pages";
pi.Header = "Print Job For /nFPT Inc.";
pi.Colors = new Drawing.Color[] {Drawing.Color.Red, Drawing.Color.Blue};
pi.Images = new System.Drawing.Image[]
{System.Drawing.Image.FromFile("D:\Corporate.jpg"),
System.Drawing.Image.FromFile("D:\Building.jpg")};
```

```
pi.RepeatColEnd = 25;
pi.RepeatColStart = 1;
pi.RepeatRowEnd = 25;
pi.RepeatRowStart = 1;
fpSpread1.Sheets[0].PrintInfo = pi;
fpSpread1.SavePdf("c:\\test.pdf");
```

VB

```
' Create PrintInfo object and set properties.
Dim pi As New FarPoint.Web.Spread.PrintInfo
pi.Footer = "This is Page /p/nof /pc Pages"
pi.Header = "Print Job For /nFPT Inc."
pi.Colors = New Drawing.Color() {Drawing.Color.Red, Drawing.Color.Blue}
pi.Images = New System.Drawing.Image()
{System.Drawing.Image.FromFile("D:\Corporate.jpg"),
System.Drawing.Image.FromFile("D:\Building.jpg")}
pi.RepeatColEnd = 25
pi.RepeatColStart = 1
pi.RepeatRowEnd = 25
pi.RepeatRowStart = 1
FpSpread1.Sheets(0).PrintInfo = pi
FpSpread1.SavePdf("c:\test.pdf")
```

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 Class' 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.Web.Spread.PrintInfo printset = new FarPoint.Web.Spread.PrintInfo();
printset.Header = "/lJobName";
printset.Footer = "/r/p of /pc";
// Create SheetView object and assign it to the first sheet.
FarPoint.Web.Spread.SheetView SheetToPrint = new FarPoint.Web.Spread.SheetView();
SheetToPrint.PrintInfo = printset;
// Set the PrintInfo property for the first sheet.
fpSpread1.Sheets[0] = SheetToPrint;
// Print the sheet.
fpSpread1.SavePdf("c:\\test.pdf");
```

VB

```
' Create PrintInfo object and set properties.
Dim printset As New FarPoint.Web.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.Web.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.SavePdf("c:\test.pdf")
```

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. You can open a specific sheet in an Excel file and load it to a specific sheet in Spread.

If you open or load a file or Stream object that contains more columns or rows than the sheet or sheets into which you are opening the file or stream, the component adds columns or rows as needed to the sheet or sheets. If you open or load a file or Stream object with fewer columns or rows 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 delete the additional columns or rows 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.

Read the following sections for more information and instructions:

- **Opening a Spread XML File**
- **Opening an Excel-Formatted File**
- **Opening a Text File**

Opening a Spread XML File

Spread can save data or data and formatting to an XML file or a stream, which you can then open back into the FpSpread component. You can open a Spread XML file or stream of an entire component or a specific sheet.

You can also open a file from inside Spread Designer.

Refer to the SheetView class **Open ('Open Method' in the on-line documentation)** method to open a specific sheet. Use the FpSpread **Open ('Open Method' in the on-line documentation)** method to open the entire control.

For instructions for saving Spread XML files, see **Saving to a Spread XML File**.

Using Code

Use the FpSpread component’s **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.

C#

```
// Open a Spread-compatible XML file.
FpSpread1.Open("c:\spreadfile");
```

VB

```
' Open a Spread-compatible XML file.
FpSpread1.Open("c:\spreadfile")
```

Using the Spread Designer

1. Select the **File** menu.
2. Choose the **Open** option.
3. The **Open** dialog appears.
4. Change the Files of type box to Spread files (*.XML).
5. Specify the path and file name of the file to open, and then click **Open**.
6. Click **OK** to close the Spread Designer.

Opening an Excel-Formatted File

You can open an existing Excel-formatted (BIFF8 format or XLSX) file or stream. With the SheetView object, you can open a specific Excel sheet to a specific sheet, the currently active sheet, in Spread. You can specify which sheet in Excel to open, either by the sheet index or by the sheet name. There are multiple **OpenExcel ('OpenExcel Method' in the on-line documentation)** methods each with several options.

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, etc. In order to keep any document caching settings (changes would be lost during a postback), open the original file with the document caching only setting and then save the file using the document caching setting.

You can also open a file from inside Spread Designer.

For more information about how the data is imported from the Excel format, see the **Import and Export Reference (on-line documentation)**.

Using Code

Use the FpSpread object's **OpenExcel ('OpenExcel Method' in the on-line documentation)** method, providing the path and file name for the file to open, or providing additional information using one of the overloaded methods.

Example

This example code opens an Excel-formatted file, loads the data from the specified Excel sheet into this sheet, and sets the number of rows displayed on a single page to match the number of rows in the Excel file.

C#

```
FpSpread1.OpenExcel("c:\\excelfile.xls", 2);
FpSpread1.ActiveSheetView.PageSize = FpSpread1.Rows.Count;
```

VB

```
FpSpread1.OpenExcel("c:\excelfile.xls", 2)
FpSpread1.ActiveSheetView.PageSize = FpSpread1.Rows.Count
```

Using the Spread Designer

1. Select the **File** menu.
2. Choose the **Open** option.
3. The **Open** dialog appears.
4. Change the Files of type box to Excel files (*.XLS).
5. Specify the path and file name of the file to open, and then click **Open**.
6. Set the **PageSize** property to the total number of rows if you wish to see all the rows on one page. Use the **Property Grid** to set the **PageSize** property.
7. Click **OK** to close the Spread Designer.

Opening a 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 you specify. There are multiple **LoadTextFile ('LoadTextFile Method' in the on-line documentation)** methods each with several options.

If the file uses custom delimiters, 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.

You can also open a file from inside Spread Designer.

For instructions for saving to text files, see **Saving to a Text File**.

Using a Shortcut

Use one of the SheetView object's **LoadTextFile ('LoadTextFile Method' in the on-line documentation)** methods, specifying the path and file name or stream, whether data or data and formatting was saved, whether headers are included, and the custom delimiters, depending on the particular method you choose.

Example

This example code loads a text file that contains formatted data, headers, and custom delimiters.

C#

```
// Load a text file with headers and custom delimiters.
FpSpread1.Sheets[0].LoadTextFile("c:\textfile.txt", False,
FarPoint.Web.Spread.Model.IncludeHeaders.BothCustomOnly, "#", "%", "^");
```

VB

```
' Load a text file with headers and custom delimiters.
FpSpread1.Sheets(0).LoadTextFile("c:\textfile.txt", False,
FarPoint.Web.Spread.Model.IncludeHeaders.BothCustomOnly, "#", "%", "^")
```

Using the Spread Designer

1. Select the **File** menu.
2. Choose the **Open** option.
3. The **Open** dialog appears.
4. Change the Files of type box to Text files (*.txt).
5. Specify the path and file name of the file to open, and then click **Open**.
6. Click **OK** to close the Spread Designer.

Using Sheet Models

You can use models to customize the user experience with the spreadsheet and extend the functionality for your particular application. As a set, 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 FpSpread component, then each sheet has its own set of models.

You can do many tasks without ever using the models. Through the Spread Designer or through properties of the shortcut objects (such as Cells, Columns, and Rows), you can affect many of the changes that define your spreadsheet. But to understand fully how Spread works and to make use of many of the features and the customizations available to you as a developer, you might want to understand how to use the underlying models.

The sheet models are the basis for all the shortcut objects, so using models is generally faster than using shortcut objects (less processing time is required).

For example, in code using the shortcut object to set a value:

VB

```
FpSpread1.Sheets(0).Cells(0,0).Value = "Test"
```

would be equivalent to using the underlying data model method:

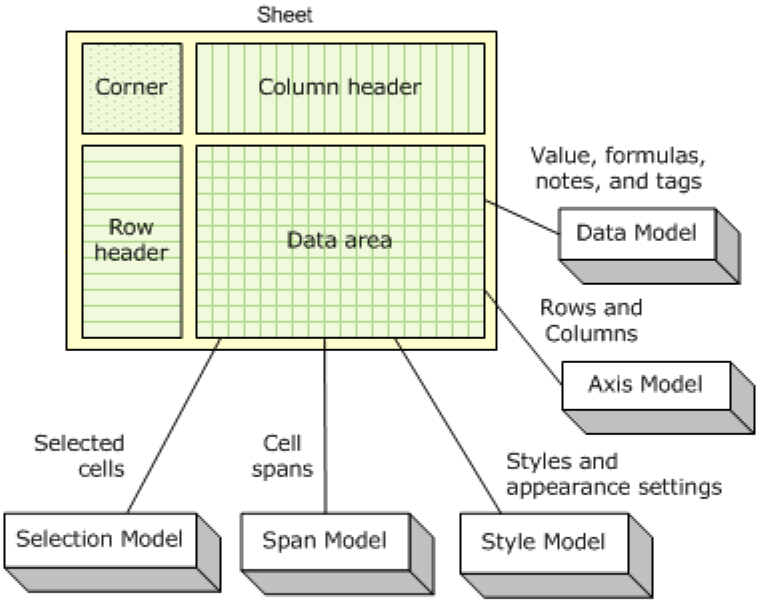
VB

```
FpSpread1.Sheets(0).DataModel.SetValue(FpSpread1.Sheets(0).GetModelRowFromViewRow(0),
FpSpread1.Sheets(0).GetModelColumnFromViewColumn(0), "Test")
```

Overview

As described in the **Product Overview**, several aspects of the sheet in the component are governed by underlying models.

Each quadrant of the sheet (corner, column header, row header, or data area) has its own set of models. The models are shown conceptually in this diagram.



These topics can help you customize the component using models:

- **Understanding How the Models Work**

- **Customizing Models**
- **Understanding the Optional Interfaces**
- **Creating a Custom Sheet Model**

Interfaces

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).

These topics provide an introduction to the use of sheet models. For complete lists of these interfaces, look at the overview for the default model classes in the **Assembly Reference (on-line documentation)**. You can find a list of the models, their classes and interfaces, and links to the Assembly Reference in **Understanding How the Models Work**.

Understanding the Models

The component provides models that provide a basis for much of the customization that is possible with the component. The following topics provide more information about models and for each of the models of a sheet.

- **Understanding How the Models Work**
- **Customizing Models**
- **Understanding the Axis Model**
- **Understanding the Data Model**
- **Understanding the Selection Model**
- **Understanding the Span Model**
- **Understanding the Style Model**

Understanding How the Models Work

To understand how models work, 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).

Sheet Model	Classes and Interface	Description
Axis model	**BaseSheetAxisModel ('BaseSheetAxisModel Class' in the on-line documentation)** **DefaultSheetAxisModel ('DefaultSheetAxisModel Class' in the**	Basis for how the sheet's rows and columns are structured. For more information, see **Understanding the Axis Model**.

	on-line documentation) **ISheetAxisModel ('ISheetAxisModel Interface' in the on-line documentation)**	
Data model	**BaseSheetDataModel ('BaseSheetDataModel Class' in the on-line documentation)** **DefaultSheetDataModel ('DefaultSheetDataModel Class' in the on-line documentation)** **ISheetDataModel ('ISheetDataModel Interface' in the on-line documentation)**	Basis for the 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)** **DefaultSheetSelectionModel ('DefaultSheetSelectionModel Class' in the on-line documentation)** **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)** **DefaultSheetSpanModel ('DefaultSheetSpanModel Class' in the on-line documentation)** **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)** **DefaultSheetStyleModel ('DefaultSheetStyleModel Class' in the on-line documentation)** **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**.

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. Most of the aspects changed with these objects automatically change the setting in the corresponding sheet model and vice versa; for example, if you add columns to the data model, then they are added to the sheet. This is true even for 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.

As shown in the figure in **Using Sheet Models**, the component is considered to have four quadrants. The data area of the spreadsheet is considered one sheet with its own set of models, and the row headers, column headers, and corner are considered as separate sheets, each with their own models.

Not everything in the Spread namespace is in the models. For example, there are aspects of the overall component, such as the sheet tabs, the sheet background color, and the grid lines, that are not in the models. But the relevant information about a given cell, both about the data in the cells and about the appearance of the cells, is in the models.

Customizing Models

Each model has a base model class and a default model class and an interface. The default model is given as the model with which you will most likely develop; this provides the default features that the component offers and is used for small customizations to the models. The base model is the base on which the default model is created and is for creating custom models from scratch. 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 can 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, and so on) is optional in the model class, and is implemented in separate interfaces from the main model interfaces (such as ISheetDataModel) so if you want to implement your own model class, you can pick and choose which pieces of functionality you have in your model.

For more information on creating a custom model for a sheet, refer to **Creating a Custom Sheet Model**.

Understanding the Axis Model

The axis model includes the methods that manage row- and column-related settings of the spreadsheet, that is, how the rows and columns of cells are oriented on the sheet.

Overview

Many of the axis-related settings are included 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 **Assembly Reference (on-line documentation)** for more information on the axis model in general and to the **DefaultSheetAxisModel ('DefaultSheetAxisModel Class' in the on-line documentation)** 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 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.
- Override the **GetSize ('GetSize Method' in the on-line documentation)** method for the row index.
- Optionally, you can override the **GetResizable ('GetResizable Method' in the on-line documentation)** method to prevent the user from trying to change the row heights manually, which will not work since **GetSize ('GetSize Method' in the on-line documentation)** is always returning the preferred height.

Example

The following example code makes each row three times taller than the default height.

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
```

Understanding the Data Model

The data model includes the contents of the cells, which could be the value or the formula in a cell, or the cell notes or cell tags. This includes the unformatted data 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.

Overview

The data model is usually the model most users who create a custom model will want to replace. The data model implements more interfaces, and more optional functionality through them, than any of the other models.

> Users who want to implement the equivalent to the unbound virtual model feature of the ActiveX Spread control must create a custom data model.

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 can be used in unbound or bound modes. In unbound mode, the data model acts much like a two-dimensional array of cell values. In bound mode, the data model wraps the supplied DataSource and if needed can supply additional settings not available from the DataSource, for example, cell formulas, and unbound rows or columns.

The **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 not much memory is used since the sparsity of the data is high. In fact, those objects do not allocate any memory for data until it is actually needed. As long as there are no notes, formulas, or tags set into the model, memory usage remains low. For more information, see the Performance section below.

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 the SheetView.**GetModelRowFromViewRow ('GetModelRowFromViewRow Method' in the on-line documentation)** and SheetView.**GetModelColumnFromViewColumn ('GetModelColumnFromViewColumn Method' in the on-line documentation)** methods.

> 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.

Setting Unformatted Data

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 (using these methods is the same as calling SheetView.Models.Data.GetValue and SheetView.Models.Data.SetValue).

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 SheetView. That value is not updated to the data model until the cell leaves edit mode. But you can manually update the value to the data model from your code, as shown in the following example.

C#

```
SheetView.SetValue(row, column, SheetView.Cells(row, column).Value);
```

VB

```
SheetView.SetValue(row, column, SheetView.Cells(row, column).Value)
```

Data Binding

When the data model implements IDataSourceSupport 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 the **AddColumns ('AddColumns Method' in the on-line documentation)** method after it is bound, and the values in those unbound columns are stored in the data model rather than the data source.

> If you add unbound columns using the **AddColumns** method, IDataSourceSupport.**IsColumnBound** returns false for those model column indexes.

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. 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). At that point, the unbound row becomes a bound row.

Performance

If you derive from **DefaultSheetDataModel ('DefaultSheetDataModel Class' in the on-line documentation)** and use that implementation of GetValue and SetValue to store the data, then it will use the Spread component's implementation of sparse arrays and matrices to balance the memory usage with the access speed. This feature 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 probably start to run out of memory).

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. In other words, putting very large amounts of data in the FpSpread component can result in a very large view state and significant delays as the view state is saved and loaded during the page life cycle. To reduce these delays, turn off the SaveViewState property and load the data each time the page loads from the Page_Load event.

You can run some simple tests by creating a test project with a Spread on a form, and setting the ColumnCount and RowCount 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 a lot 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 significant amount of data for that to happen).

Summary

The default data model class, **DefaultSheetDataModel**, implements all of the interfaces discussed in this topic, plus many others related to calculation, hierarchy, and serialization. For more detail, refer to the topics for the classes in the data model provided in the **Assembly Reference (on-line documentation)**.

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

C#

```
FpSpread1.Sheets[0].DataSource = MyData.Tables(0);
```

VB

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

and

C#

```
FarPoint.Web.Spread.Model.DefaultSheetDataModel model = new
FarPoint.Web.Spread.Model.DefaultSheetDataModel(MyData, strTable);
FpSpread1.Sheets[0].DataModel = model;
```

VB

```
Dim model As FarPoint.Web.Spread.Model.DefaultSheetDataModel = New
FarPoint.Web.Spread.Model.DefaultSheetDataModel(MyData, strTable)
FpSpread1.Sheets(0).DataModel = 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 for both code snippets, but the first example results in the old data model being garbage collected. Generally you may 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.

Understanding the Selection Model

The selection model includes any of the settings related to ranges of selected cells. This may include 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 method, for setting cells as selected, and the AddSelection, ClearSelection, and RemoveSelection methods for adding, clearing, and removing selected ranges from the sheet. Refer to the **Assembly Reference (on-line documentation)** for more information on the selection model in general and to the **DefaultSheetSelectionModel ('DefaultSheetSelectionModel Class' in the on-line documentation)** 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 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 pointer.

Some events cause the anchor cell in the selection model to get 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 Esc key, or if you use the keyboard to move the active cell instead of the mouse, then the component might clear the selection model.

The selection model is saved to the view state only if it contains at least one selection.

Understanding the Span Model

The span model includes the objects needed to handle cell spans and automatic merging of cells. Refer to the Cell 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 **Assembly Reference (on-line documentation)** for more information on the span model in general and to the **DefaultSheetSpanModel ('DefaultSheetSpanModel Class' in the on-line documentation)** methods in particular.

The default implementation of the span model (**DefaultSheetSpanModel ('DefaultSheetSpanModel Class' in the on-line documentation)**) uses an array to store the cell spans. If there are a modest number of spans in the sheet (for example, a thousand or less) then the default implementation is responsive. 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.

Understanding the Style Model

The style model includes appearance settings, whether determined in the Spread Designer, or set as properties in the Properties List or as inherited by a custom skin for a whole sheet or by a custom style for individual cells. For more information on appearance settings for a sheet, refer to **Creating a Skin for Sheets** and **Applying a Skin to a Sheet**. For more information on appearance settings for a cell, refer to **Creating and Applying a Custom Style for**

Cells.

Overview

The appearance settings may be set from any of the following classes in the 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 also may be set from any of these classes in the Spread namespace that affect style:

- **Appearance ('Appearance Class' in the on-line documentation)**
- **Border ('Border 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)**

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
- Border
- CellType
- Editor
- Font
- ForeColor
- Formatter
- HorizontalAlignment
- Locked
- Renderer
- VerticalAlignment

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 with Cell Types**.

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. Refer to the **Assembly Reference (on-line documentation)** for more information on the style model in general and to the **DefaultSheetStyleModel ('DefaultSheetStyleModel Class' in the on-line documentation)** methods in particular.

Direct Versus Composite

“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.

Styles

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 set to -1), column (row index set to -1), or the entire model (column and row index set to -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.**DefaultStyle ('DefaultStyle Property' in the on-line documentation)**, ColumnHeader.**DefaultStyle ('DefaultStyle Property' in the on-line documentation)**, and RowHeader.**DefaultStyle ('DefaultStyle Property' in the on-line documentation)**).

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 a row index of -1 and a column index of -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.

Setting **StyleName** replaces the style in the style model with the **NamedStyle ('NamedStyle Class' in the on-line documentation)** having the specified name. This has the effect of changing the settings of all of the style-related properties, including **ParentStyleName** (which wraps StyleInfo.Parent). Any previous setting for style-related properties like **BackColor**, **Font**, **Border**, or **ParentStyleName** are overwritten when you set **StyleName**. The component expects that the named style is set up with all the properties as you want them to be set. 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** object, and any changes that you make to one of those cells will also be changing all of the other cells sharing the same named style.

Example

The style properties for a cell can be composited or merged from the **Cell**, **Row**, **Column**, **Sheet**, and parent **NamedStyle** objects. This code snippet illustrates the order of inheritance of a composited style:

CS

```
NamedStyle test_parent = new NamedStyle("test_parent");
NamedStyle test = new NamedStyle("test", "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";
```

This code creates two **NamedStyle** objects with a parent-child relationship, then sets some properties on the styles and adds them to the **NamedStyleCollection** in the FpSpread component. Then 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 no 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**.

Formatting Information

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. 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 Spread, and no FormatInfo is stored in the style model, but the data entered into the cells is always treated as text.

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.
IDataKeySupport ('IDataKeySupport Interface' in the on-line documentation), **IDataSourceSupport ('IDataSourceSupport Interface' in the on-line documentation)**	Allow customization of data binding on a sheet; used in conjunction with each other.
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)**, **IIterationSupport ('IIterationSupport Interface' in the on-line documentation)**	Allow 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)**.
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)	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)**	Allow 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.
ISerializeSupport ('ISerializeSupport Interface' in the on-line documentation)	Allows customization of XML serialization for the contents of a sheet.
IUnboundRowSupport ('IUnboundRowSupport Interface' in the on-line documentation)	Allows customization of unbound rows with data binding on a sheet; used in conjunction with **IDataSourceSupport ('IDataSourceSupport Interface' in the on-line documentation)**.

None of these optional interfaces are required for saving Excel or text files, or for printing. For more detailed information on these interfaces, refer to the **Assembly Reference (on-line documentation)**.

Creating a Custom Sheet Model

You can use a sheet model as a template for a new custom model. For example, you might want to make a custom data model. Using a custom data model requires creating a class which implements **ISheetDataModel ('ISheetDataModel Interface' in the on-line documentation)**, then setting an instance of the class into the SheetView.Models.Data 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 optional interfaces, refer to **Understanding the Optional Interfaces**.

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

In **BaseSheetDataModel ('BaseSheetDataModel Class' in the on-line documentation)**, the **Changed ('Changed Event' in the on-line documentation)** event is also implemented for you, along with the overloaded **FireChanged** method, so you do not need to provide an implementation of that event either. The event itself is the only thing in **BaseSheetDataModel ('BaseSheetDataModel Class' in the on-line documentation)** that is not virtual.

The **FireChanged** and **OnChanged** methods are protected members of **BaseSheetDataModel ('BaseSheetDataModel Class' in the on-line documentation)** and are not part of any interface. They are virtual helper methods for the **Changed** event. The only difference between the **FireChanged** and **OnChanged** method is that **OnChanged** takes the EventArgs argument, and **FireChanged** takes the arguments used to create the EventArgs. The **FireChanged** method will not create the EventArgs object unless there is actually a handler attached to the delegate, so it is better to use that in subclasses for firing the event.

In a few cases, you may need to create your own custom data model for performance reasons. For example, suppose you want to display a large table of computed values (such as an addition or multiplication table) that consists of a million rows by ten columns. If you used the default sheet data model, you would need to compute and store all ten million values which would consume a lot of time and memory. Here is example code that creates a custom data model that enhances performance.

Example

This example creates a custom data model.

C#

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

Maintaining State

To provide a seamless and coherent user experience, Web-based applications must include a way of maintaining state. ASP.NET pages and Web Forms components can automatically maintain state and some Web Forms components offer properties and methods that let you manage this. Take into account the various advantages and disadvantages of each solution when choosing the method of maintaining state that will optimize the performance of your application.

As an ASP.NET developer, you can optimize the way your application maintains state by understanding the impact of several factors. You will want to handle state management differently depending on the amount of data, availability of the server, and the resources for implementation. You can have your application save data locally on the client (either to the View State, in a cookie, or in hidden form fields), or on the server (either to the Session State or Application State or to an SQL database). Depending on the amount of data, some approaches to saving state have more impact on the performance of the server and some have more on the client side.

The following table provides an overview of the state management options and features specific to each option.

	Less Coding	**Large Data Sets**	**Server Performance**	**Client Performance**	**Data Durability**	**Scalability**	**Security**	**Data Integrity**	**Data Accessible by other software**	**Configuration options**
View State	X									
Session State		X		X	X	X				
SQL Database		X	X		X	X	X	X	X	X
Page Request			X	X						

If you are unfamiliar with state management, first read the overview in **State Overview**.

In Spread for ASP.NET, you can manage state in different ways, including the following options:

- **Saving Data to the View State**, which is client based
- **Saving Data to the Session State**, which is server based
- **Saving Data to an SQL Database**, which is server based
- **Loading Data for Each Page Request**

These topics discuss the advantages and disadvantages of these options. Review this information to determine the best approach for your application.

State Overview

The HTML page state is a snapshot of the state of all the page’s data and property settings. This information needs to round-trip from the server to the client to maintain a seamless and coherent user experience. The state maintains the information in the HTML page when the page is refreshed. As an ASP.NET developer, you can and should maintain the state when the page is refreshed, so that user data remains in the page. You have probably experienced pages that do not maintain state; when the page is refreshed, such as to remind you to complete part of a form, your information is lost, and you must complete the entire page again. Understandably, users prefer pages that maintain the state.

State management can be done in many ways. The method you choose can affect performance; therefore, you should understand the various ways you can manage state and choose the appropriate option for your application. You need to set up your application’s state management to optimize performance while maintaining the state.

ASP.NET pages and Web Forms components, including Spread for ASP.NET, automatically maintain state. In addition, the Spread component offers properties and methods that let you manage the state, including:

- FarPoint.Web.Spread.FpSpread.EnableViewState property
- FarPoint.Web.Spread.SheetView.IsTrackingViewState property
- FarPoint.Web.Spread.SheetView.LoadViewState method
- FarPoint.Web.Spread.SheetView.SaveViewState method

Saving Data to the View State

View state management provides client-based state management, where data is written into pages in hidden fields.

To save the data to the view state, set the **IsTrackingViewState ('IsTrackingViewState Property' in the on-line documentation)** property for the active sheet to true, which is the default setting.

Advantages and Disadvantages

The advantages of using the view state are:

- No server resources are required.
- It requires less coding.
- It requires less database access.
- Spread manages all its data. The user changes are saved automatically to the component's data model.

The default control settings use the view state to save data. This default setting is best for small data models. If you are using larger data sets, you will probably want to use one of the other state management options

The disadvantage of using view state is the impact on performance. Because the view state is stored in the page itself, storing large values can cause the page to slow down when users display it and when they post it.

Using Code

Use the view state to save data.

Example

The following sample illustrates using the view state to save data.

C#

```
protected void Page_Load(object sender, System.EventArgs e)
{
if (this.IsPostBack) return;
// Connect to NWIND MS Access example with OLE DB.
OleDbConnection thisConnection = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=D:\\NWIND.MDB");
// Open connection.
thisConnection.Open();
// Create DataSet to contain related data tables, rows, and columns.
DataSet thisDataSet = new DataSet();
OleDbDataAdapter orderAdapter = new OleDbDataAdapter("SELECT EmployeeID, LastName,
FirstName, Title FROM Employees", thisConnection);
orderAdapter.Fill(thisDataSet, "Employees");
FpSpread1.ActiveSheetView.IsTrackingViewState = true;
FpSpread1.ActiveSheetView.DataSource = thisDataSet;
FpSpread1.ActiveSheetView.DataMember = "Employees";
thisConnection.Close();
thisConnection.Dispose();
}
```

VB

```
 Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    If (Me.IsPostBack) Then Return

    ' Connect to NWIND MS Access example with OLE DB.
    Dim thisConnection As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0; Data
Source=D:\NWIND.MDB")
    ' Open connection.
    thisConnection.Open()
    ' Create DataSet to contain related data tables, rows, and columns.
    Dim thisDataSet As New DataSet()
    Dim orderAdapter As New OleDbDataAdapter("SELECT EmployeeID, LastName, FirstName,
Title FROM Employees", thisConnection)

    orderAdapter.Fill(thisDataSet, "Employees")

    FpSpread1.ActiveSheetView.IsTrackingViewState = True
    FpSpread1.ActiveSheetView.DataSource = thisDataSet
    FpSpread1.ActiveSheetView.DataMember = "Employees"

    thisConnection.Close()
    thisConnection.Dispose()
 End Sub
```

Saving Data to the Session State

Session state management provides server-based state management, where data is saved to separate browser sessions for each user.

To save the data to the session state, set the **IsTrackingViewState ('IsTrackingViewState Property' in the on-line documentation)** property for the active sheet to True, as with saving data to the view state. Then handle the session state in the **SaveOrLoadSheetState ('SaveOrLoadSheetState Event' in the on-line documentation)** event.

Advantages and Disadvantages

The advantages of using the session state are:

- It offers easier implementation.
- It requires less database access.
- It can handle larger data models.
- Because the view state is small, pages load quickly.
- It enhances data durability. Data placed in session-state variables can survive Internet Information Services (IIS) restarts and worker-process restarts without losing session data because the data is stored in another process space.
- It provides platform scalability. Session state can be used in both multiple-computer and multiple-process configurations.

Using the session state is best if the data is too big to save to the view state.

The disadvantage of using the session state is the impact on performance. Session state variables stay in server memory until they are either removed or replaced, and therefore can degrade server performance. Session state variables containing blocks of information like large data sets can adversely affect Web server performance as server load increases.

Using Code

Use the session state to save data.

Example

The following sample illustrates using the session state to save data.

C#

```
protected void Page_Load(object sender, System.EventArgs e)
{
if (this.IsPostBack) return;
// Connect to NWIND MS Access example with OLE DB.
OleDbConnection thisConnection = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=D:\\NWIND.MDB");
// Open connection.
thisConnection.Open();
// Create DataSet to contain related data tables, rows, and columns.
DataSet thisDataSet = new DataSet();
OleDbDataAdapter orderAdapter = new OleDbDataAdapter("SELECT * FROM Orders",
thisConnection);
orderAdapter.Fill(thisDataSet, "Orders");
FpSpread1.ActiveSheetView.IsTrackingViewState = true;
FpSpread1.ActiveSheetView.DataSource = thisDataSet;
FpSpread1.ActiveSheetView.DataMember = "Orders";
}

protected void FpSpread1_SaveOrLoadSheetState(object sender,
FarPoint.Web.Spread.SheetViewStateEventArgs e)
{
if (e.IsSave)
{
Session[e.SheetView.SheetName] = e.SheetView.SaveViewState();
}
else
{
e.SheetView.LoadViewState(Session[e.SheetView.SheetName]);
}
e.Handled = true;
}
```

VB

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    If (Me.IsPostBack) Then Return
    ' Connect to NWIND MS Access example with OLE DB.
     Dim thisConnection As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=D:\NWIND.MDB")
    ' Open connection.
    thisConnection.Open()
    ' Create DataSet to contain related data tables, rows, and columns.
    Dim thisDataSet As New DataSet()
    Dim orderAdapter As New OleDbDataAdapter("SELECT * FROM Orders", thisConnection)
    orderAdapter.Fill(thisDataSet, "Orders")
    FpSpread1.ActiveSheetView.DataSource = thisDataSet
    FpSpread1.ActiveSheetView.DataMember = "Orders"
    FpSpread1.ActiveSheetView.IsTrackingViewState = True
End Sub

Private Sub FpSpread1_SaveOrLoadSheetState(ByVal sender As Object, ByVal e As
FarPoint.Web.Spread.SheetViewStateEventArgs) Handles FpSpread1.SaveOrLoadSheetState
    If (e.IsSave) Then  Session(e.SheetView.SheetName) = e.SheetView.SaveViewState()
    Else
        e.SheetView.LoadViewState(Session(e.SheetView.SheetName))
    End If
    e.Handled = True
End Sub
```

Saving Data to an SQL Database

Database state management saves data to a specified database. Using an SQL database for state management is best if you are working with large amounts of data, particularly data that needs to be secure and maintain integrity.

To save data to an SQL database, you must install SQL State Management, as explained in the example.

Advantages and Disadvantages

The advantages of using a database to maintain state are:

- Databases are typically very secure, requiring rigorous authentication and authorization.
- Databases offer large capacity.
- Database information can be stored as long as you like, and it is not subject to the availability of the Web server.
- Databases include various facilities for maintaining data integrity.
- Data stored in your database is accessible to a wide variety of information-processing tools.
- There are numerous database tools available, offering wide support and custom configurations.

The disadvantages of using a database to maintain state are:

- Using a database to support state management requires more complex hardware and software configurations.
- Using a database can affect performance, for example due to poor construction of the relational data model. Also, large numbers of queries to the database can adversely affect server performance.

Using Code

This example describes how to install SQL state management. To install SQL State Management complete the following instructions.

1. Do one of the following:
    - If you are using the MSDE version of SQL Server that ships with Visual Studio.NET you need to open a command prompt window and navigate to the Windows\Microsoft.Net\Framework\(Version) directory. Once there, issue the following command to run the InstallSqlState.sql script:
      ```
      OSQL -S localhost -U sa -P <InstallSqlState.sql
      ```
    - OSQL.exe is a tool that ships with MSDE and SQL Server. It allows you to apply a T-SQL script to a SQL Server.
    - If you are using SQL Server 7 or SQL Server 2000 you can follow the directions above or you can open the Enterprise Manager, open the InstallSqlState.sql script from the Windows\Microsoft.Net\(Framework Version) directory, and execute the script from there.
    - Whichever method you choose, the script will set up an ASPState database in your SQL Server Group Databases.
2. After you have run the script to set up SQL state management, you need to make a change to your project's web.config file. Under the session State section, change the Mode setting from its current setting (most likely InProc) to SQL Server.
3. Then configure the sqlConnectionString to point to the SQL Server where you installed the T-SQL script InstallSqlState.sql as follows:
   ```
   sqlConnectionString="data source=127.0.0.1;user id=sa;password=;"
   ```

Loading Data for Each Page Request

When you load data for each page request, you are not maintaining state, rather, you are re-creating each page as it is requested. Load data for every page request when there is a large data set and you must minimize the use of server resources.

To load data for every page request, set the **IsTrackingViewState ('IsTrackingViewState Property' in the on-line documentation)** property for the active sheet to False.

Advantages and Disadvantages

The advantages of loading data for every page request are:

- No server resources are required.
- Because the view state is small, pages load quickly.

The disadvantages of loading data for every page request are:

- The programmers must code more to handle the **UpdateCommand ('UpdateCommand Event' in the on-line documentation)**, **InsertCommand ('InsertCommand Event' in the on-line documentation)**, and **DeleteCommand ('DeleteCommand Event' in the on-line documentation)** events to update the database. In addition, the programmers might need to set up row, column, or cell styles for each page request.
- This method requires more database access if the FpSpread component is bound.

Using Code

Load data each time the page is loaded.

Example 1

The following sample code illustrates loading data for every page request.

VB

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
  ' Put user code to initialize the page here.
  OleDbDataAdapter1.Fill(DataSet11, "Orders")
  FpSpread1.ActiveSheetView.DataKeyField = "OrderID"
  FpSpread1.ActiveSheetView.IsTrackingViewState = False
  Me.DataBind()
End Sub

Private Sub FpSpread1_UpdateCommand(ByVal sender As Object, ByVal e As
FarPoint.Web.Spread.SpreadCommandEventArgs) Handles FpSpread1.UpdateCommand
  Dim conn As New OleDb.OleDbConnection()
  conn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Password="""";User
ID=Admin;Data Source=C:\test\NW" & _
  "ind.mdb;Mode=Share Deny None;Extended Properties="""";Jet OLEDB:System
database="""""" & _
  ";Jet OLEDB:Registry Path="""";Jet OLEDB:Database Password="""";Jet OLEDB:Engine
Type" & _
  "=4;Jet OLEDB:Database Locking Mode=0;Jet OLEDB:Global Partial Bulk Ops=2;Jet OLE" &
_
  "DB:Global Bulk Transactions=1;Jet OLEDB:New Database Password="""";Jet OLEDB:Creat"
& _
  "e System Database=False;Jet OLEDB:Encrypt Database=False;Jet OLEDB:Don't Copy Lo" &
_
  "cale on Compact=False;Jet OLEDB:Compact Without Replica Repair=False;Jet OLEDB:S" &
_
```

```
  "FP=False"
  Dim cmdText As String = "UPDATE Orders SET CustomerID = ?, EmployeeID = ?, Freight =
?, OrderDate = ?, Req" & _
"uiredDate = ?, ShipAddress = ?, ShipCity = ?, ShipCountry = ?, ShipName = ?, Shi" & _
"ppedDate = ?, ShipPostalCode = ?, ShipRegion = ?, ShipVia = ? WHERE (OrderID = ?)"
  Dim updateCmd As OleDb.OleDbCommand = New OleDb.OleDbCommand(cmdText, conn)
  updateCmd.Parameters.Add(New System.Data.OleDb.OleDbParameter("CustomerID",
System.Data.OleDb.OleDbType.VarWChar, 5, "CustomerID"))
  updateCmd.Parameters.Add(New System.Data.OleDb.OleDbParameter("EmployeeID",
System.Data.OleDb.OleDbType.Integer, 0, System.Data.ParameterDirection.Input, False,
CType(10, Byte), CType(0, Byte), "EmployeeID", System.Data.DataRowVersion.Current,
Nothing))
  updateCmd.Parameters.Add(New System.Data.OleDb.OleDbParameter("Freight",
System.Data.OleDb.OleDbType.Currency, 0, System.Data.ParameterDirection.Input, False,
CType(19, Byte), CType(0, Byte), "Freight", System.Data.DataRowVersion.Current,
Nothing))
  updateCmd.Parameters.Add(New System.Data.OleDb.OleDbParameter("OrderDate",
System.Data.OleDb.OleDbType.DBDate, 0, "OrderDate"))
  updateCmd.Parameters.Add(New System.Data.OleDb.OleDbParameter("RequiredDate",
System.Data.OleDb.OleDbType.DBDate, 0, "RequiredDate"))
  updateCmd.Parameters.Add(New System.Data.OleDb.OleDbParameter("ShipAddress",
System.Data.OleDb.OleDbType.VarWChar, 60, "ShipAddress"))
  updateCmd.Parameters.Add(New System.Data.OleDb.OleDbParameter("ShipCity",
System.Data.OleDb.OleDbType.VarWChar, 15, "ShipCity"))
  updateCmd.Parameters.Add(New System.Data.OleDb.OleDbParameter("ShipCountry",
System.Data.OleDb.OleDbType.VarWChar, 15, "ShipCountry"))
  updateCmd.Parameters.Add(New System.Data.OleDb.OleDbParameter("ShipName",
System.Data.OleDb.OleDbType.VarWChar, 40, "ShipName"))
  updateCmd.Parameters.Add(New System.Data.OleDb.OleDbParameter("ShippedDate",
System.Data.OleDb.OleDbType.DBDate, 0, "ShippedDate"))
  updateCmd.Parameters.Add(New System.Data.OleDb.OleDbParameter("ShipPostalCode",
System.Data.OleDb.OleDbType.VarWChar, 10, "ShipPostalCode"))
  updateCmd.Parameters.Add(New System.Data.OleDb.OleDbParameter("ShipRegion",
System.Data.OleDb.OleDbType.VarWChar, 15, "ShipRegion"))
  updateCmd.Parameters.Add(New System.Data.OleDb.OleDbParameter("ShipVia",
System.Data.OleDb.OleDbType.Integer, 0, System.Data.ParameterDirection.Input, False,
CType(10, Byte), CType(0, Byte), "ShipVia", System.Data.DataRowVersion.Current,
Nothing))
  updateCmd.Parameters.Add(New System.Data.OleDb.OleDbParameter("OrderID",
System.Data.OleDb.OleDbType.Integer, 0, System.Data.ParameterDirection.Input, False,
CType(10, Byte), CType(0, Byte), "OrderID", System.Data.DataRowVersion.Original,
Nothing))
  Dim sv As FarPoint.Web.Spread.SheetView = e.SheetView
  Dim keyValue As String = sv.GetDataKey(e.CommandArgument)
  ' Find the row.
  Dim rowFlag As Boolean = False
  Dim keyCol As Integer = 4 ' order id
  Dim r As Integer
  For r = 0 To sv.RowCount - 1
    Dim tmp As String = sv.GetValue(r, 4)
    If (tmp = keyValue) Then
      rowFlag = True
      Exit For
    End If
  Next
  If Not rowFlag Then
    Return
  End If

Dim i As Integer
For i = 0 To sv.ColumnCount - 1
  Dim colName As String = sv.GetColumnLabel(0, i)
  If (Not e.EditValues.Item(i) Is FarPoint.Web.Spread.FpSpread.Unchanged) Then
    updateCmd.Parameters(colName).Value = e.EditValues.Item(i) ElseIf
(OleDbUpdateCommand1.Parameters.Contains(colName)) Then
    updateCmd.Parameters(colName).Value = sv.GetValue(r, i)
  End If
Next

Try
  conn.Open()
  i = updateCmd.ExecuteNonQuery()
  conn.Close()
  conn.Dispose()
Catch ex As Exception
  ' Update database failed.
  conn.Close()
  conn.Dispose()
End Try
End Sub
```

Example 2

The following sample code illustrates loading data for every page request. This example requires more coding, but provides a more efficient application. To further speed up page loading, you can use the "where" clause in the SQL statements to retrieve one page of records so that database server does not have to return a large data set.

VB

```
Private topRow As Integer
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
  topRow = FpSpread1.Sheets(0).TopRow
  SetDataModel(topRow, topRow)
End Sub

Private Sub FpSpread1_TopRowChanged(ByVal sender As Object, ByVal e As
FarPoint.Web.Spread.SpreadCommandEventArgs) Handles FpSpread1.TopRowChanged
  SetDataModel(topRow, e.SheetView.TopRow)
End Sub

Public Sub SetDataModel(ByVal oldTopRow As Integer, ByVal newTopRow As Integer)
  Dim firstOrderID As Integer = -1
  Dim lastOrderID As Integer = -1
  If Not ViewState("lastOrderID") Is Nothing Then
    lastOrderID = ViewState("lastOrderID")
  End If
  If Not ViewState("firstOrderID") Is Nothing Then
    firstOrderID = ViewState("firstOrderID")
  End If
  Dim ps As Integer = FpSpread1.Sheets(0).PageSize
  If newTopRow > oldTopRow Then
    Me.OleDbSelectCommand1.CommandText = "SELECT Top " & ps & " CustomerID, EmployeeID,
Freight, OrderDate, OrderID, RequiredDate, ShipAdd" & _
```

```
"ress, ShipCity, ShipCountry, ShipName, ShippedDate, ShipPostalCode, ShipRegion, " & _
"ShipVia FROM Orders" & " Where OrderID >" & lastOrderID & " Order by OrderID"
    ElseIf newTopRow = oldTopRow Then
    Me.OleDbSelectCommand1.CommandText = "SELECT Top " & ps & " CustomerID, EmployeeID,
Freight, OrderDate, OrderID, RequiredDate, ShipAdd" & _
"ress, ShipCity, ShipCountry, ShipName, ShippedDate, ShipPostalCode, ShipRegion, " & _
"ShipVia FROM Orders" & " Where OrderID >=" & firstOrderID & " Order by OrderID"
  Else
    Me.OleDbSelectCommand1.CommandText = "SELECT Top " & ps & " CustomerID, EmployeeID,
Freight, OrderDate, OrderID, RequiredDate, ShipAdd" & _
"ress, ShipCity, ShipCountry, ShipName, ShippedDate, ShipPostalCode, ShipRegion, " & _
"ShipVia FROM Orders" & " Where OrderID <" & firstOrderID & " Order by OrderID DESC"
  End If
FpSpread1.Sheets(0).IsTrackingViewState = False
DataSet31.Tables(0).Clear()

If newTopRow < oldTopRow Then
  ' Reverse the order.
  Dim tmpTable As Data.DataTable = DataSet31.Tables(0).Clone()
  OleDbDataAdapter1.Fill(tmpTable)

  Dim dr As Data.DataRow
  Dim i As Integer
  For i = 0 To tmpTable.Rows.Count - 1
    dr = tmpTable.Rows(tmpTable.Rows.Count - 1 - i)
    DataSet31.Tables(0).ImportRow(dr)
  Next
Else
  OleDbDataAdapter1.Fill(DataSet31)
End If

Dim model As MyModel = New MyModel(DataSet31, String.Empty)
model.TopRow = newTopRow

Dim dbCmd As Data.OleDb.OleDbCommand = New Data.OleDb.OleDbCommand("select count(*)
from orders", OleDbConnection1)
OleDbConnection1.Open()
model.RowCount = CType(dbCmd.ExecuteScalar(), Integer)
OleDbConnection1.Close()

FpSpread1.Sheets(0).DataModel = model
ViewState("firstOrderID") = DataSet31.Tables(0).Rows(0).Item("OrderID")
Dim dtcount As Integer = DataSet31.Tables(0).Rows.Count
ViewState("lastOrderID") = DataSet31.Tables(0).Rows(dtcount - 1).Item("OrderID")
End Sub
'PageLoad
```

Example 3

The following sample code illustrates loading data for every page request. This example requires more coding, but provides a more efficient application. To further speed up page loading, you can use the "where" clause in the SQL statements to retrieve one page of records so that database server does not have to return a large data set.

VB

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
```

```
  'Put user code to initialize the page here.
  Dim topRow As Integer = FpSpread1.Sheets(0).TopRow
  SetDataModel(topRow)
End Sub

Public Sub SetDataModel(ByVal topRow As Integer)
  Dim ps As Integer = FpSpread1.Sheets(0).PageSize

  Me.OleDbSelectCommand1.CommandText = "SELECT CategoryID, Discontinued, ProductID,
ProductName, QuantityPerUnit, Reorder" & _
"Level, SupplierID, UnitPrice, UnitsInStock, UnitsOnOrder FROM Products" & " Where
ProductID >=" & topRow & " and ProductID <= " & (topRow + ps) & " Order by ProductID"

FpSpread1.Sheets(0).IsTrackingViewState = False
DataSet41.Tables(0).Clear()
OleDbDataAdapter1.Fill(DataSet41)

Dim model As MyModel = New MyModel(DataSet41, String.Empty)
model.TopRow = topRow
Dim dbCmd As Data.OleDb.OleDbCommand = New Data.OleDb.OleDbCommand("select count(*)
from Products", OleDbConnection1)
OleDbConnection1.Open()
model.RowCount = CType(dbCmd.ExecuteScalar(), Integer)
OleDbConnection1.Close()
FpSpread1.Sheets(0).DataModel = model
End Sub

Private Sub FpSpread1_TopRowChanged(ByVal sender As Object, ByVal e As
FarPoint.Web.Spread.SpreadCommandEventArgs) Handles FpSpread1.TopRowChanged
SetDataModel(e.SheetView.TopRow)
End Sub

Public Class MyModel
Inherits FarPoint.Web.Spread.Model.BaseSheetDataModel

Private dataset As Data.DataSet = Nothing
Private datamember As String = String.Empty
Private trow As Integer = 0
Private rCount As Integer = 0

Public Sub New(ByVal ds As Data.DataSet, ByVal dm As String)
  dataset = ds
  datamember = dm
End Sub

Public Overrides Function GetValue(ByVal row As Integer, ByVal col As Integer) As
Object
Dim dt As Data.DataTable = Me.GetDataTable()
If dt Is Nothing Then
  Return Nothing
Else
If row < TopRow Or row >= TopRow + dt.Rows.Count Then
  Return Nothing
Else
  Dim r As Integer = row - TopRow
  Return dt.Rows(r).Item(col)
End If
End If
End Function

Public Overrides Function IsEditable(ByVal row As Integer, ByVal col As Integer) As
Boolean
Return True
End Function

Public Function GetDataTable() As Data.DataTable
If dataset Is Nothing Then
  Return Nothing
Else
If datamember Is Nothing Or datamember = String.Empty Then
  Return dataset.Tables(0)
Else
  Return dataset.Tables(datamember)
End If
End If
End Function

Public Property TopRow() As Integer
Get
  Return trow
End Get
Set(ByVal Value As Integer)
  trow = Value
End Set
End Property

Public Overrides Property RowCount() As Integer
Get
  Return rCount
End Get
Set(ByVal Value As Integer)
  rCount = Value
End Set
End Property

Public Overrides Property ColumnCount() As Integer
Get
Dim dt As Data.DataTable = GetDataTable()

If (dt Is Nothing) Then
  Return 0
Else
  Return dt.Columns.Count
End If
End Get
Set(ByVal Value As Integer)
End Set
End Property

End Class
```

Working with the Chart Control

The Chart control can be used with the Spread control to display your data in professional looking charts. The Chart control can be added using the Spread designer and Spread properties. You can also use Chart classes to create a Chart control. The following topics explain general features of the control as well as how to use the control.

The following topics include:

- **Understanding and Customizing Charts**
- **Creating Charts**

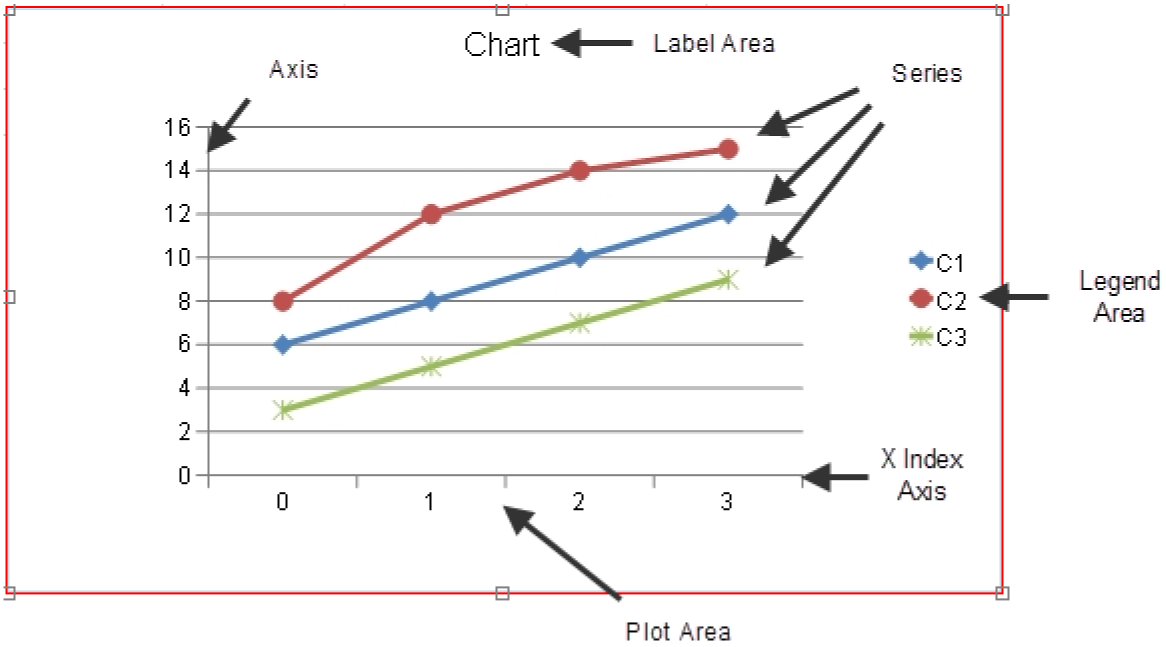
Understanding and Customizing Charts

You can create different plot types and customize charts with borders, labels, legends, and other effects. Consult the following sections for more information about formatting charts:

- **Chart User Interface Elements**
- **Chart Types and Views**
- **Plot Types**
- **Series**
- **Walls**
- **Axis and Other Lines**
- **Fill Effects**
- **Elevation and Rotation**
- **Lighting, Shapes, and Borders**
- **Size - Height, Width, and Depth**
- **Labels**
- **Legends**

Chart User Interface Elements

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.



For more information about the chart elements, refer to the following topics:

- **Plot Types**
- **Series**
- **Axis and Other Lines**
- **Labels**
- **Legends**

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.

Plot Types

A plot area is the area in which data points (bars, points, lines, and so on) are drawn. A plot area contains 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 Plot Types**
- **XY Plot Types**
- **XYZ Plot Types**
- **Pie Plot Types**
- **Polar Plot Types**
- **Radar Plot Types**
- **Data Plot Types**

Each plot type has different chart types associated with it, as explained in the plot type topics.

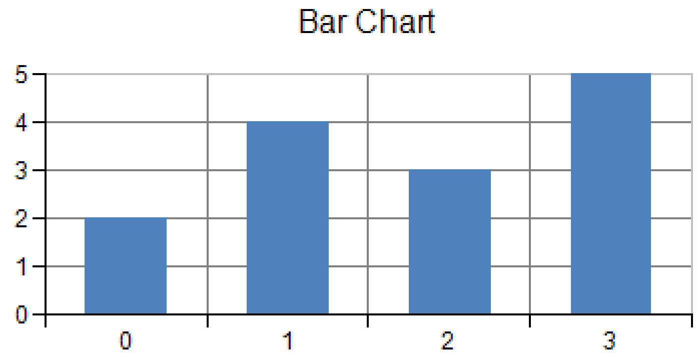
Y Plot Types

The Y plot area contains series that have values in one dimension.

When visualized in two dimensions, 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 three dimensions, a Y plot area takes the form of a cube with the *x*-axis representing categories, the *y*-axis representing values, and the *z*-axis (depth) representing series.

A Y plot area can be oriented vertically or horizontally. When oriented vertically, the *x*-axis is horizontal and the *y*-axis is vertical. When oriented horizontally, the *x*-axis is vertical and the *y*-axis is horizontal. The following image shows a bar chart:

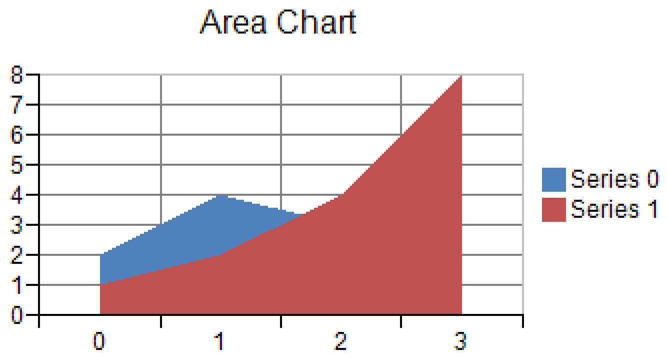


You can have any of these types of Y plots:

- **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.

- Area
- Stacked Area
- Stacked 100% (normalized) Area

Area Charts

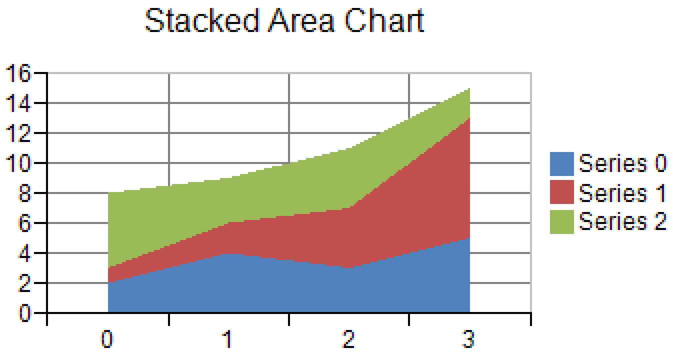
Each point in an area series contains a single data value. The data value is visualized as a point on an area.

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.



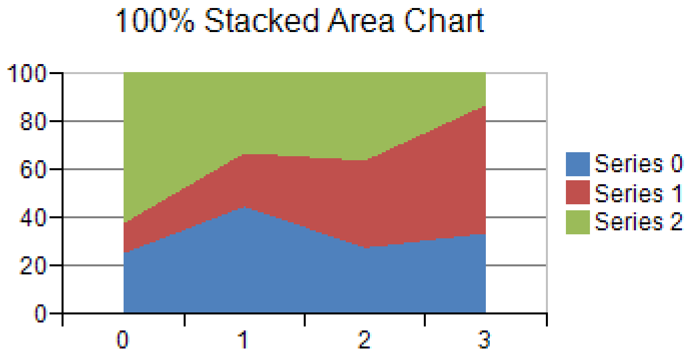
A stacked area series is a composite series that groups together two or more area series.

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

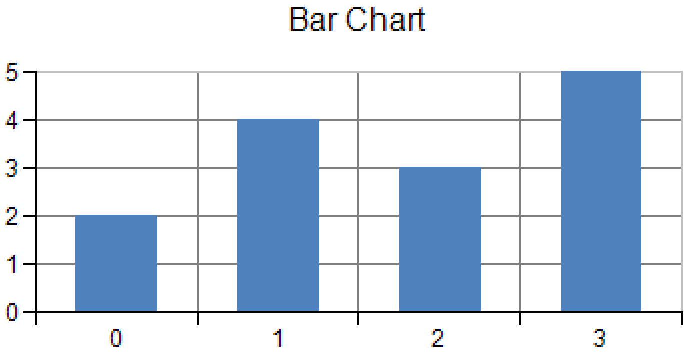
The stacked 100% area chart shows the points vertically stacked and spread all the way to the top (100%) but spread proportionately; otherwise, it is similar to a stacked area chart.



For more information on the area series object in the API, refer to the AreaSeries 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.

- Bar
- Multiple Bar
- Cluster Bar
- Stacked Bar
- Stacked 100% (normalized) Bar

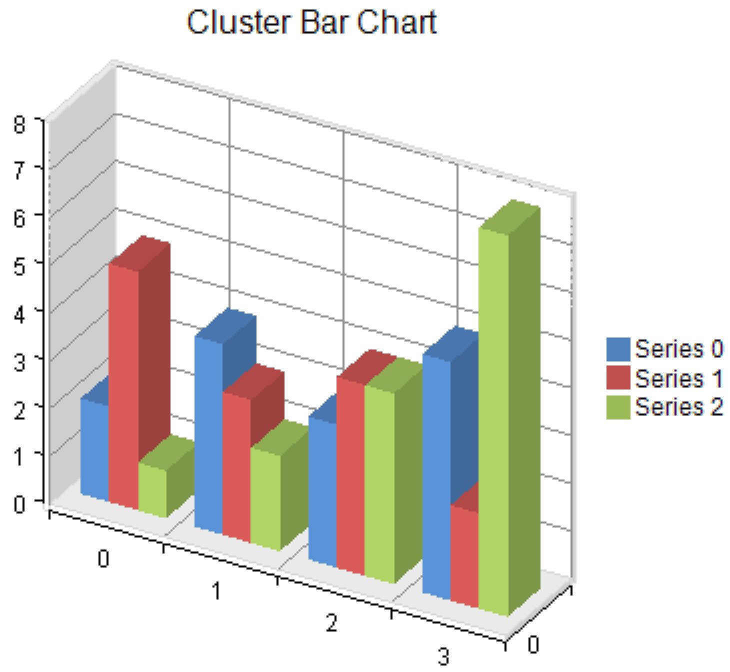
Bar Charts

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 class to True.

Clustered Bar Charts

The cluster bar shows the bars alongside each other horizontally, clustered by series.



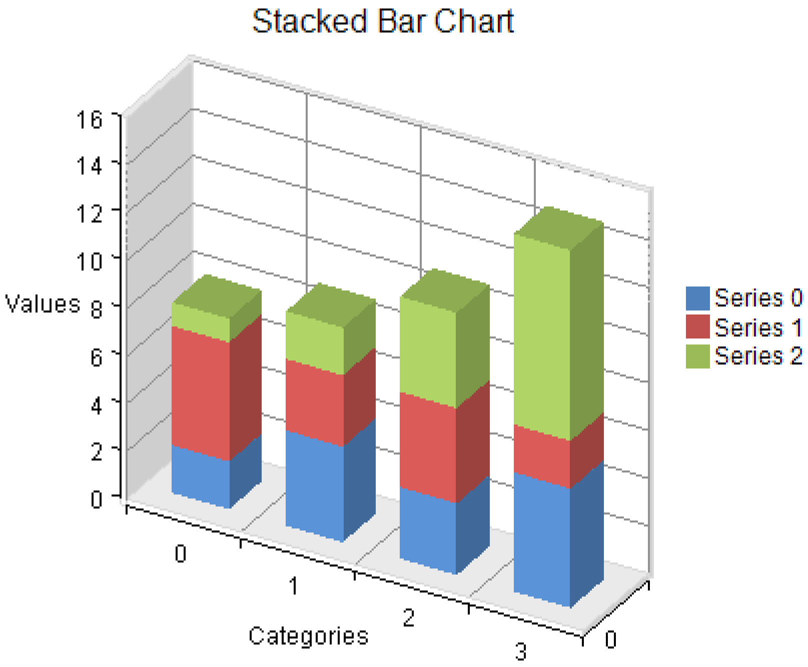
A cluster bar series is a composite series that groups together two or more bar series. A cluster bar series can be assigned a border, fill effect, width, depth, and an origin for the bars as well as a width for the group. Assigning null for a border or fill effect indicates that the property is unset. 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.



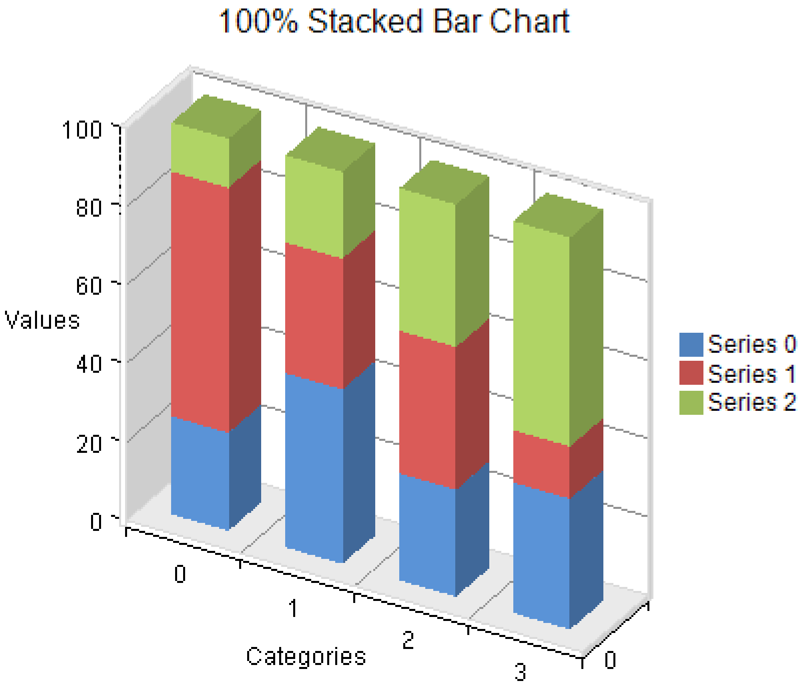
A stacked bar series is a composite series that groups together two or more bar series. A stacked bar series can be assigned a border, fill effect, width, and a depth for the bars. You can also specify whether the group should be displayed 100% stacked. Assigning null for a border or fill effect indicates 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

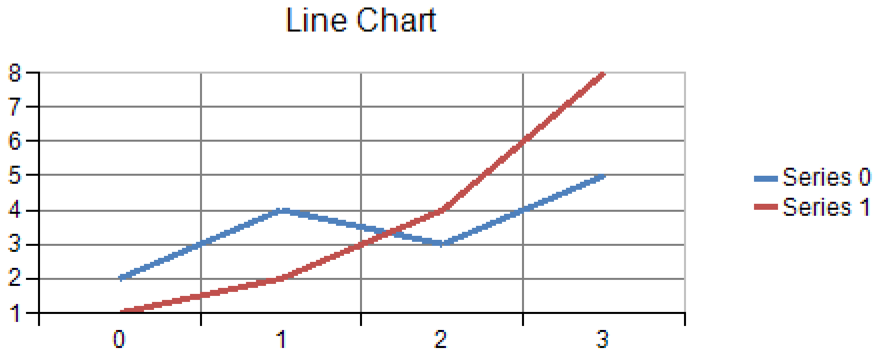
The stacked 100% bar shows the bars vertically stacked and spread all the way to the top (100%) but spread proportionately; otherwise, they are similar to stacked bar charts.



For more information on the bar series object in the API, refer to the BarSeries 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.

- Line
- Stacked Line
- Stacked 100% (normalized) Line

Line Charts

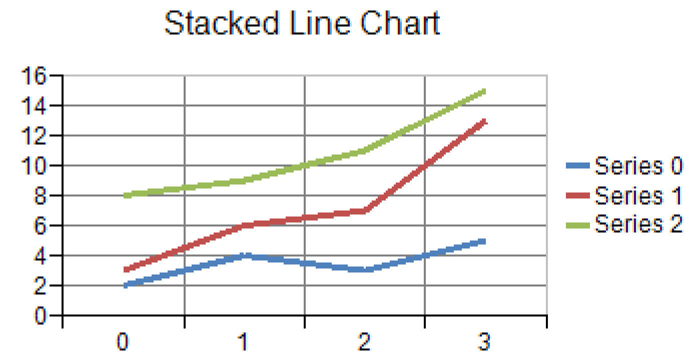
Each point in a line series contains a single data value. The data values are visualized as points on a line.

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.



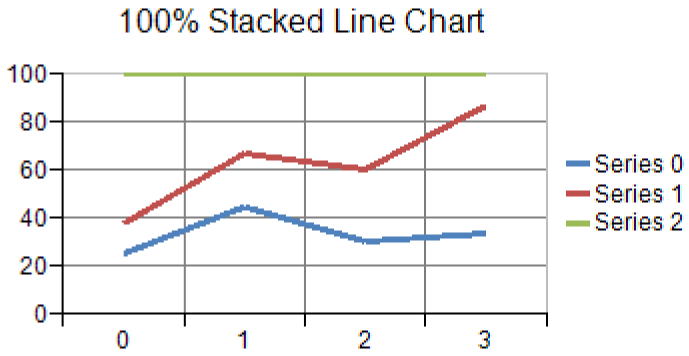
A stacked line series is a composite series that groups together two or more line series. A stacked line series can have a border, fill effect, or depth for the lines. You can also specify whether the line is jagged or smooth and whether drop lines are displayed. Assigning null for a border or fill effect 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

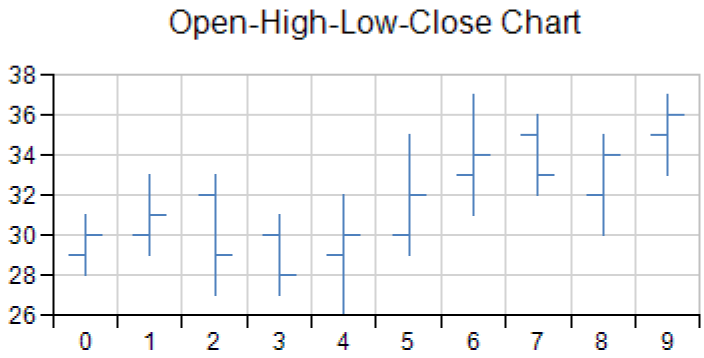
The stacked 100% line chart shows the points vertically stacked and spread all the way to the top (100%) but spread proportionately; otherwise, it is similar to a stacked line chart.



For more information on the line series object in the API, refer to the LineSeries 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:

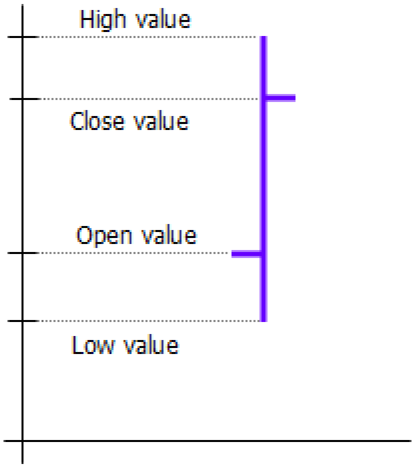


The normal high-low series are displayed as vertical lines with the high value being the vertically highest point on the line and the low value being the lowest point on the line. The opening market value is the small horizontal tick on the left of the line and the closing value is the small horizontal 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.

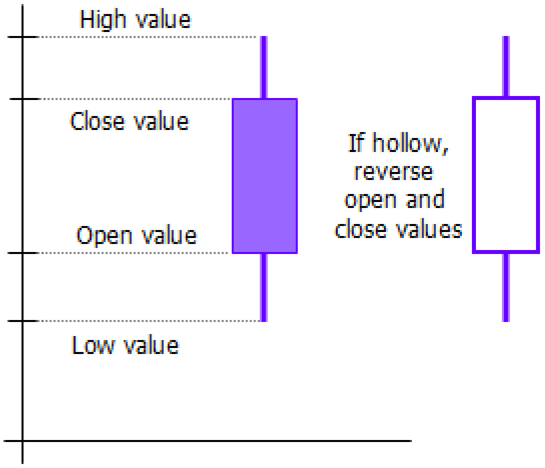
An open-high-low-close series can be assigned a line for up or down points, and a width. The width is measured relative to the floor grid cell (0 = no depth, 1 = width to edge of grid cell).

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.

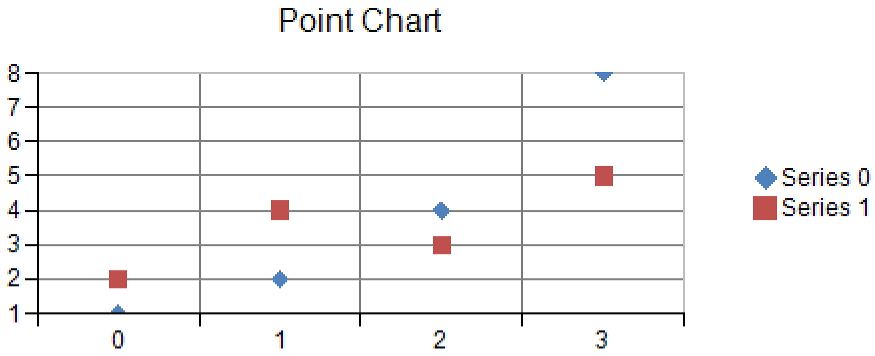
Each point can be assigned a border for up and down points. You can also set a fill effect for up and down points.

For more information on the objects in the API, refer to these classes:

- HighLowAreaSeries
- HighLowBarSeries
- HighLowCloseSeries

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.

- Point
- Stacked Point
- Stacked 100% (normalized) Point

Point Charts

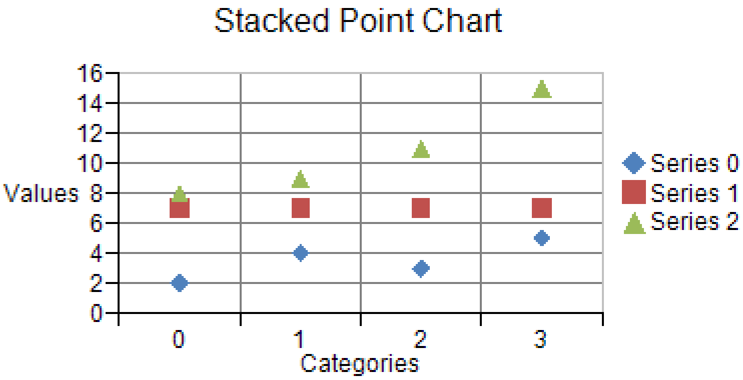
A point marker is used to visualize each data value. Each point in a point series contains a single data value.

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.

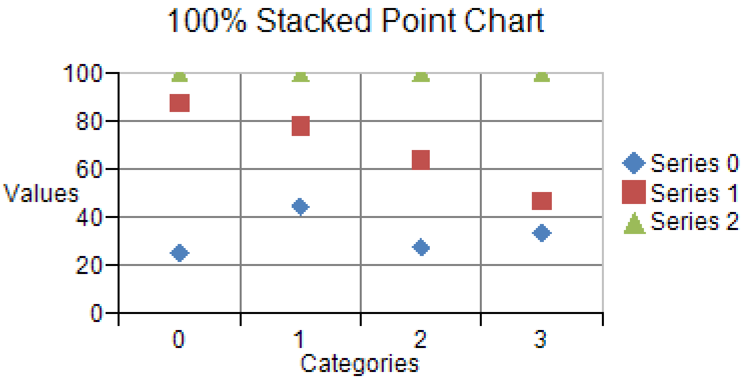
A stacked point series can have a border, fill effect, size, or depth for the point markers. You can also specify whether the group should be displayed as 100% stacked. Assigning null for a border or fill effect indicates that the property is unset. Size is measured in model units. Depth is measured 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

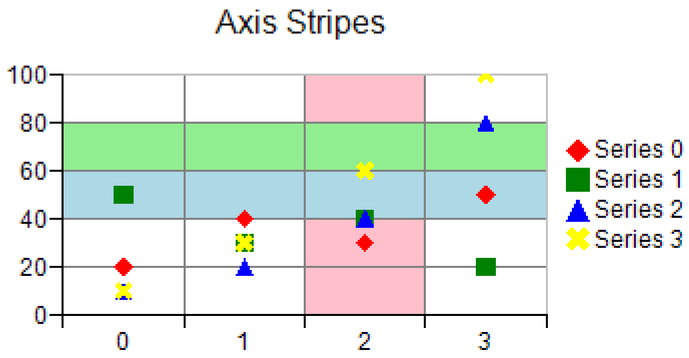
The stacked 100% point chart shows the points vertically stacked and spread all the way to the top (100%) but spread proportionately; otherwise, the stacked 100% point chart is similar to the stacked point chart.



For more information on the point series object in the API, refer to the PointSeries class.

Stripe Charts

Stripes can be used in a basic one-dimensional Cartesian plot such as the one shown in this figure.



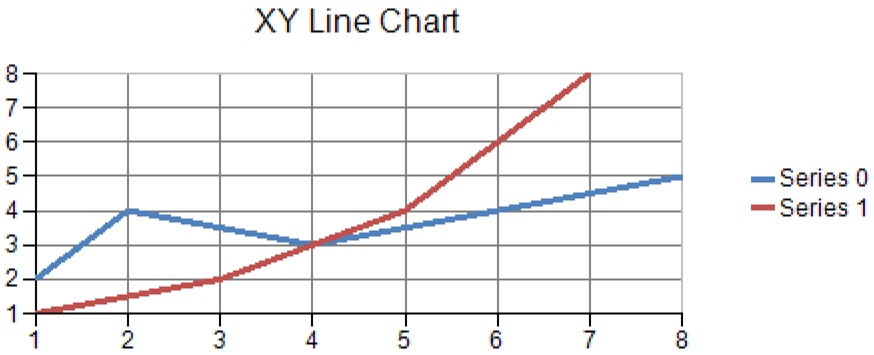
You can specify a start and end value for the stripe on the chart axis. You can also specify a fill type and color for the stripe. A stripe can be added to a chart with the axis properties in the appropriate plot area class.

For more information on the stripe object in the API, refer to the Stripe class.

XY Plot Types

The XY plot area contains series that have values in two dimensions. When visualized in two dimensions, 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 three dimensions, 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.

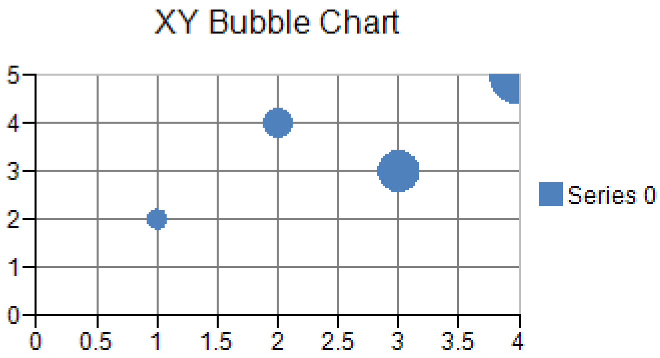


You can have any of these types of XY plots:

- **Bubble Charts**
- **Line Charts**
- **Point Charts**
- **Stripe Charts**

Bubble Charts

The bubble chart can be an XY plot such as the one shown in this figure.



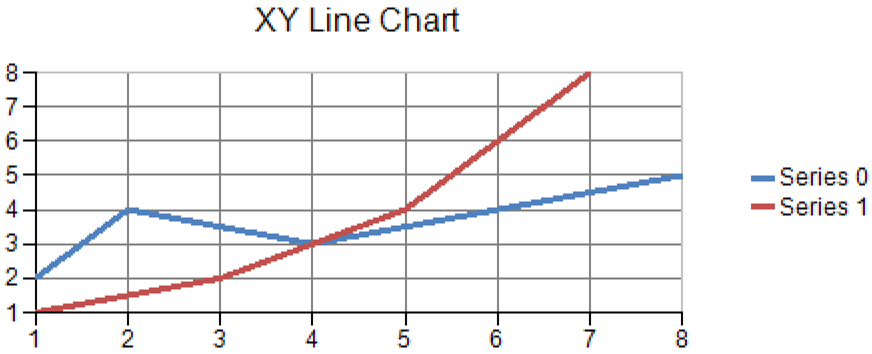
Each point contains two values: value and size.

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 class.

Line Charts

The line chart can be an XY plot such as the one shown in this figure.

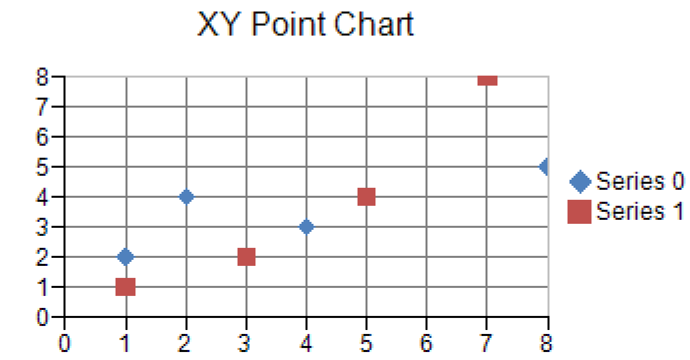


An XY line series can have a border, fill effect, and depth for the line. You can also specify whether the line is jagged or smooth and whether drop lines are displayed. Assigning null for a border or fill effect indicates that the property is unset. Depth is measured relative to the floor grid cell (0 = no depth, 1 = depth of floor grid cell).

For more information on the line series object in the API, refer to the XYLineSeries class.

Point Charts

The point chart can be an XY plot such as the one shown in this figure.



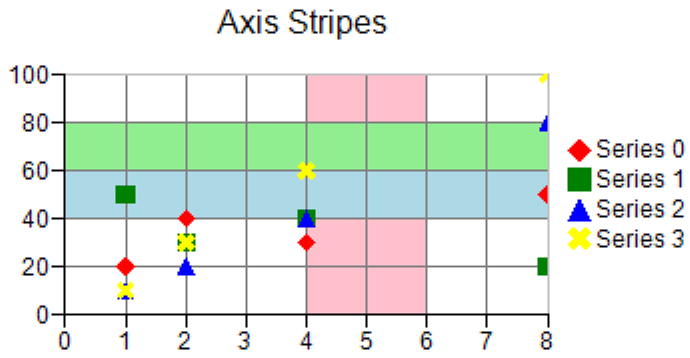
Each point in an XY point series contains two data values: x and y. Each point is visualized as a point marker.

The point markers in an XY point series or the series can have a border, fill effect, shape, size, and depth. Settings at the point level would have precedence. 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).

For more information on the point series object in the API, refer to the PointSeries class.

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.

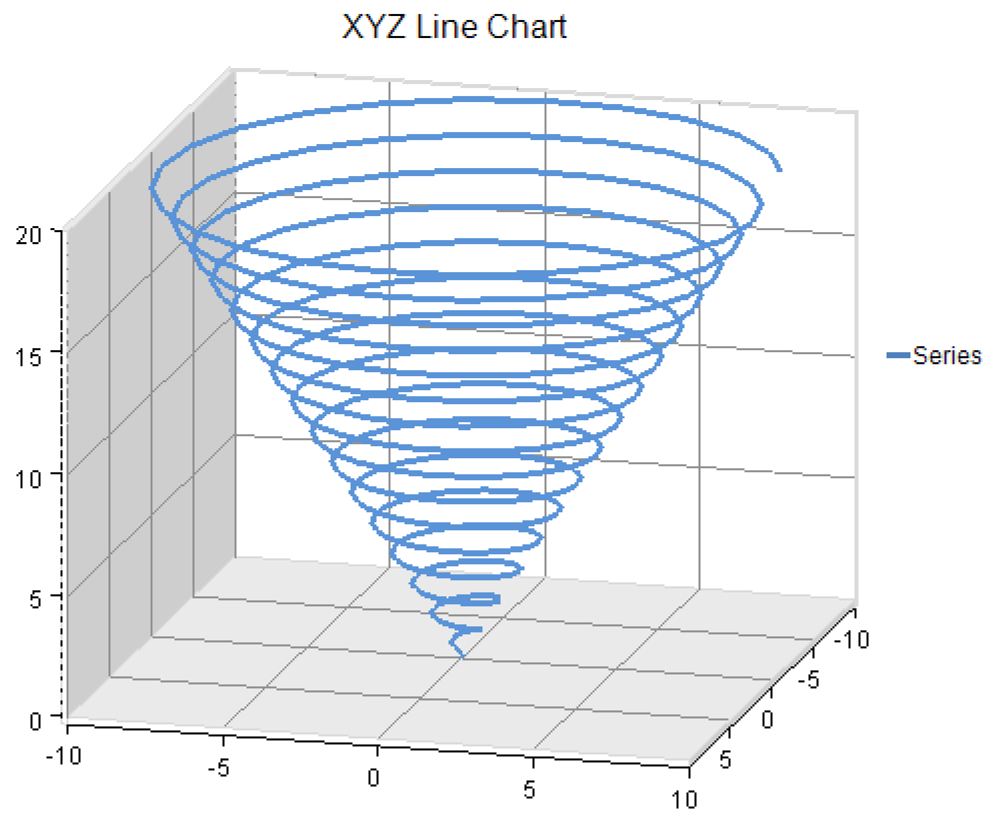
For more information on the stripe object in the API, refer to the Stripe class.

XYZ Plot Types

The XYZ plot area contains series that have values in three dimensions. When visualized in two dimensions, 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 three dimensions, 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 several subtypes of XYZ series: XYZ point, XYZ line, XYZ surface, and XYZ stripe.

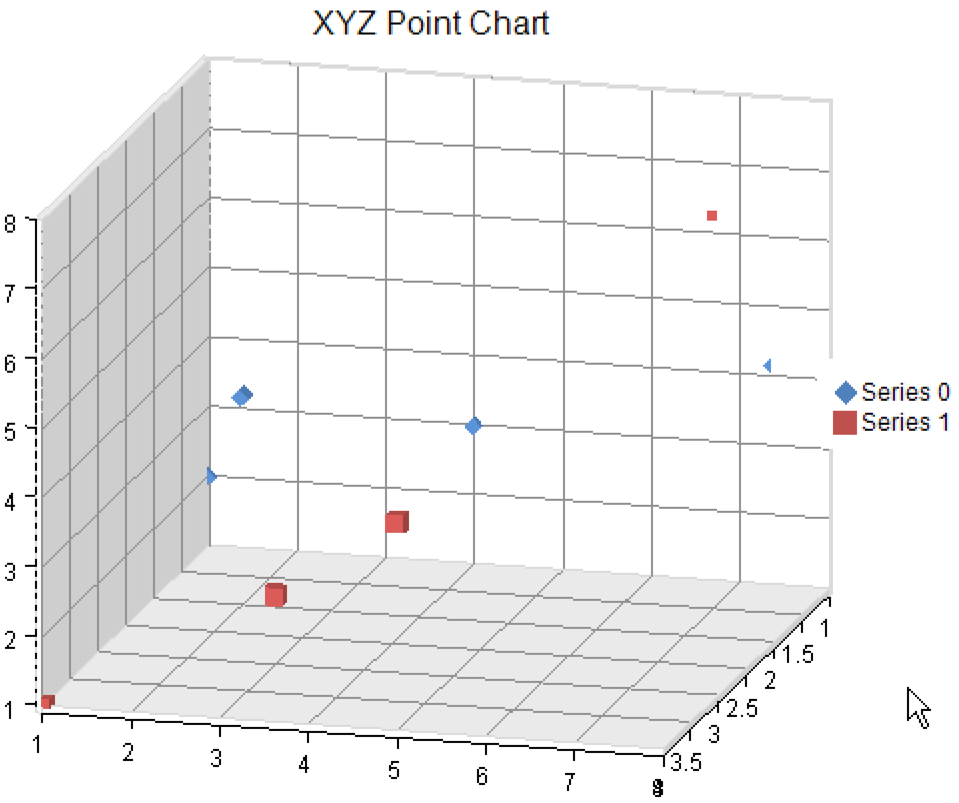


You can have any of these types of XYZ plots:

- **Point Charts**
- **Line Charts**
- **Surface Charts**
- **Stripe Charts**

Point Charts

The point chart can be an XYZ plot such as the one shown in this figure.



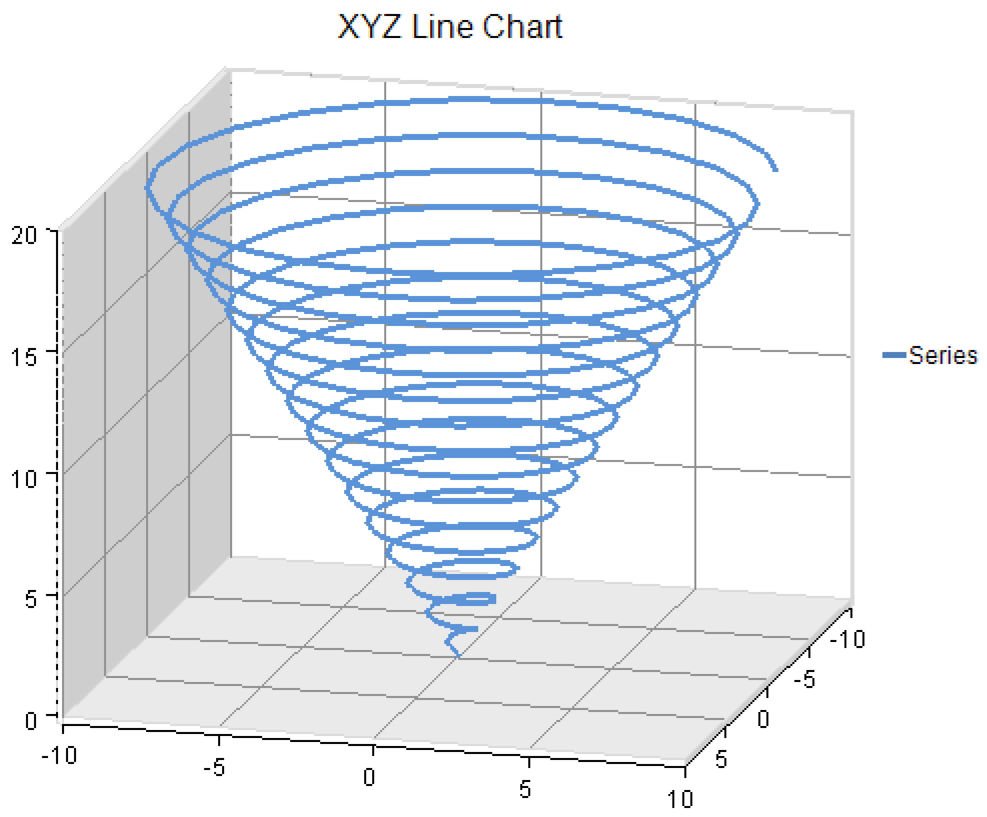
Each point in an XYZ point series has three data values: x, y, and z. Each point is visualized as a point marker.

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 class.

Line Charts

The line chart can be an XYZ plot such as the one shown in this figure.



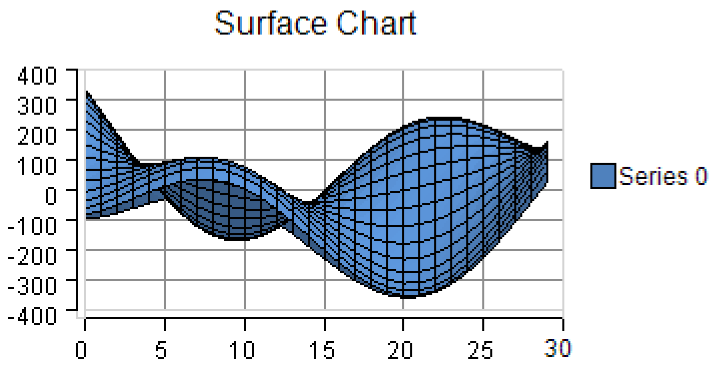
Each point in an XYZ line series has three data values: x, y, and z. Each point is visualized as a point on a line.

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 class.

Surface Charts

The surface chart can be an XYZ plot such as the one shown in this figure.



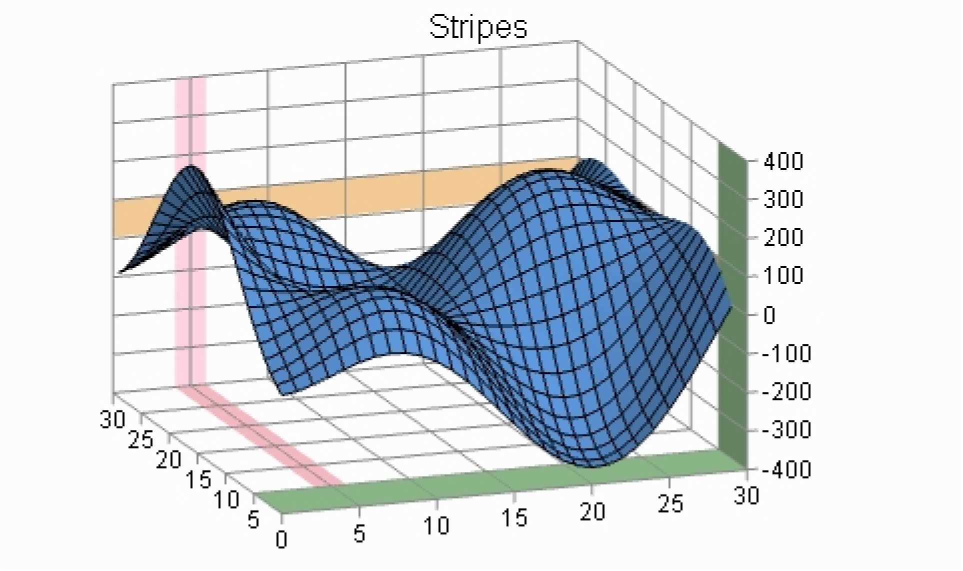
Each point in an XYZ series has three data values: x, y, and z. Each point is visualized as a point on a surface.

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 class.

Stripe Charts

The stripe chart can be an XYZ plot such as the one shown in this figure.

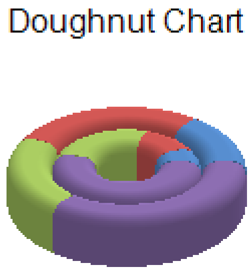


You can specify a start and end value for the stripe on the chart axis. You can also specify a fill type and color for the stripe. A stripe can be added to a chart with the axis properties in the appropriate plot area class.

For more information on the stripe object in the API, refer to the Stripe 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.



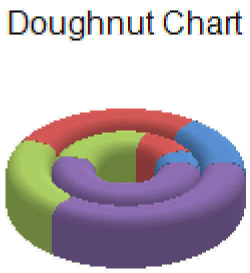
You can have any of these types of Pie plots:

- **Doughnut Charts**
- **Pie Charts**

For details on the API, refer to the PiePlotArea 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 ('HoleSize Property' in the on-line documentation)** property is used to create the doughnut chart. If this property is not set when using the PieSeries class, the chart would be a pie chart.

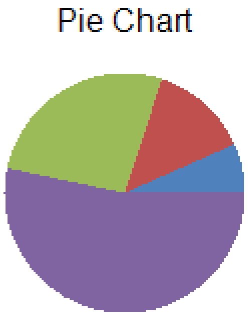
A pie series can be assigned a border and fill effect for the pie slices. Assigning null for a border or fill effect indicates that the property is null.

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 ('PieDetachments Property' in the on-line documentation)** property) is used to create an exploded doughnut chart.

For more information on the pie series object in the API, refer to the PieSeries 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 ('PieDetachments Property' in the on-line documentation)** property) is used to create an exploded pie chart.

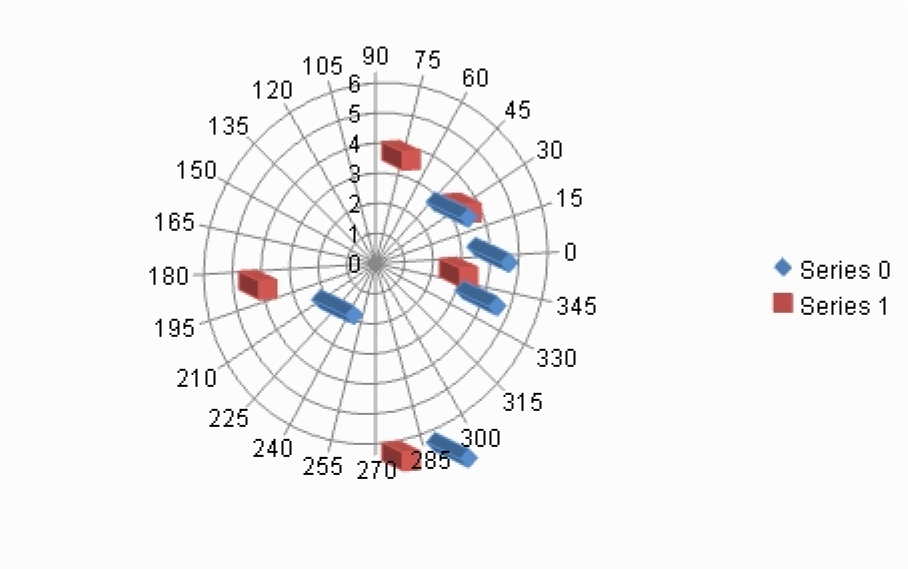
For more information on the pie series object in the API, refer to the PieSeries class.

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 ('Elevation Property' in the on-line documentation)**, **Rotation ('Rotation Property' in the on-line documentation)**, and **ViewType ('ViewType Property' in the on-line documentation)** properties. The **Depth ('Depth Property' in the on-line documentation)** property in the plot area class was used to add depth to the point markers.



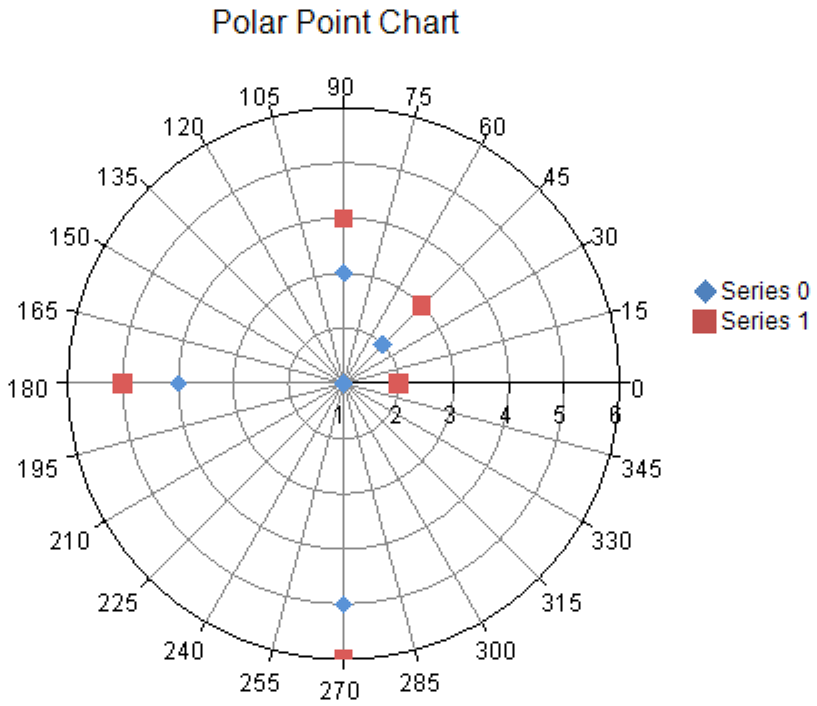
You can have any of these types of Polar plots:

- **Point Charts**
- **Line Charts**
- **Area Charts**
- **Stripe Charts**

For details on the API, refer to the PolarPlotArea class.

Point Charts

The point chart can be a polar plot such as the one shown in the following figure.



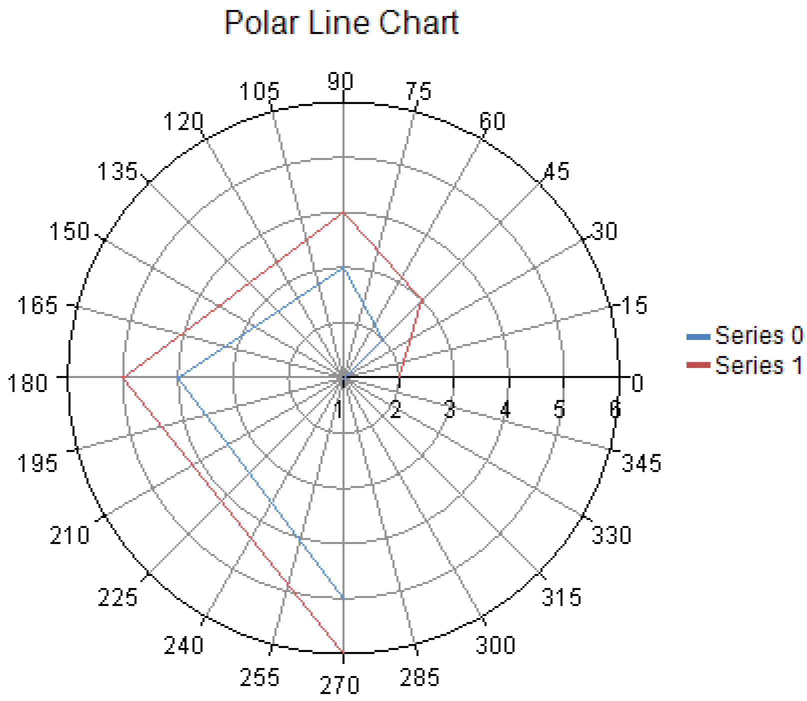
Each point has two data values: x (angle) and y (radius). Each point is visualized as a point marker.

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 class.

Line Charts

The line chart can be a polar plot such as the one shown in the following figure.



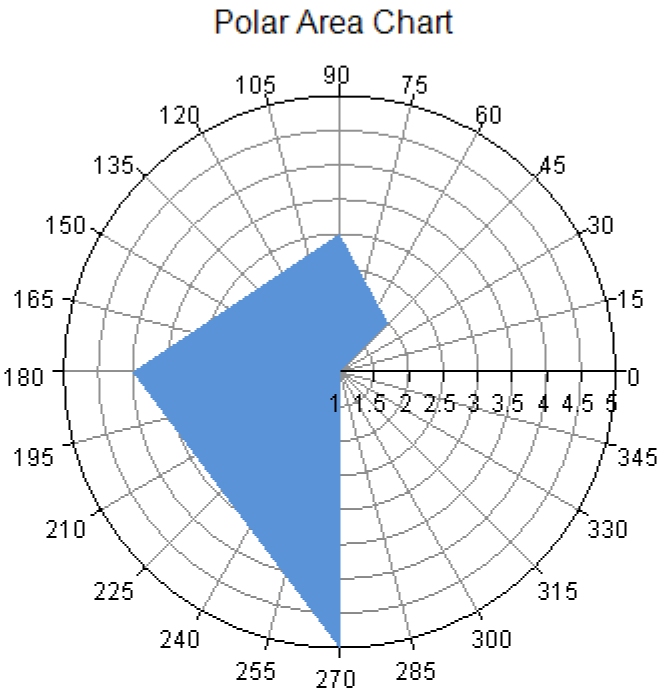
Each point has two data values: x (angle) and y (radius). Each point is visualized as a point on a line.

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 class.

Area Charts

The area chart can be a polar plot such as the one shown in the following figure.



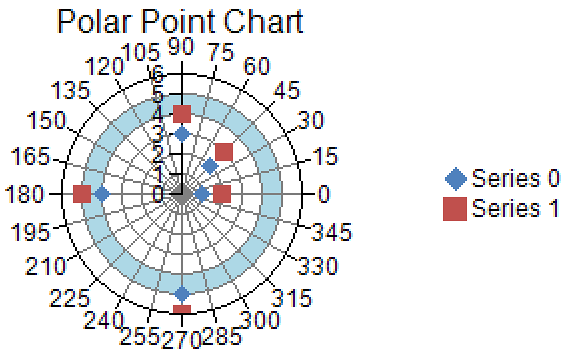
Each point has two data values: x (angle) and y (radius). Each point is visualized as a point on an area.

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 class.

Stripe Charts

The stripe chart can be a polar plot such as the one shown in this figure.



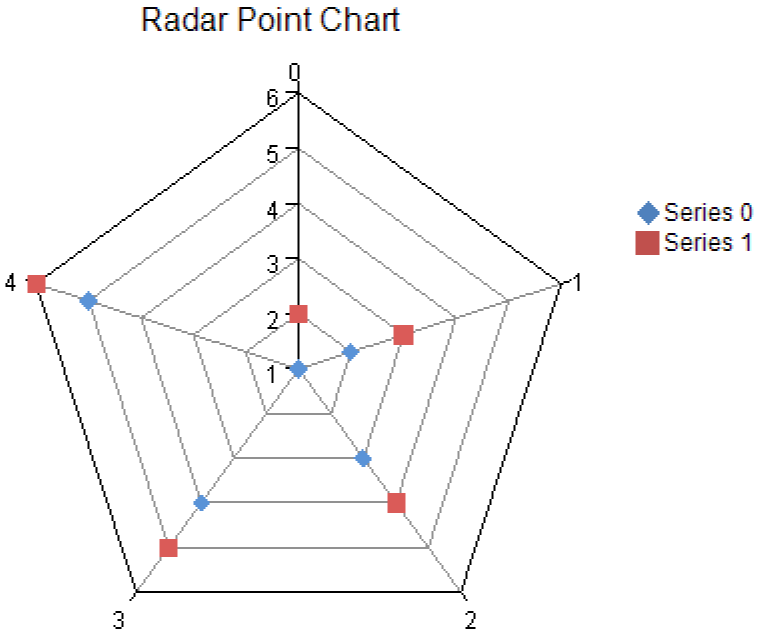
You can specify a start and end value for the stripe on the chart axis. You can also specify a fill type and color for the stripe. A stripe can be added to a chart with the axis properties in the appropriate plot area class.

For more information on the stripe object in the API, refer to the Stripe 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.



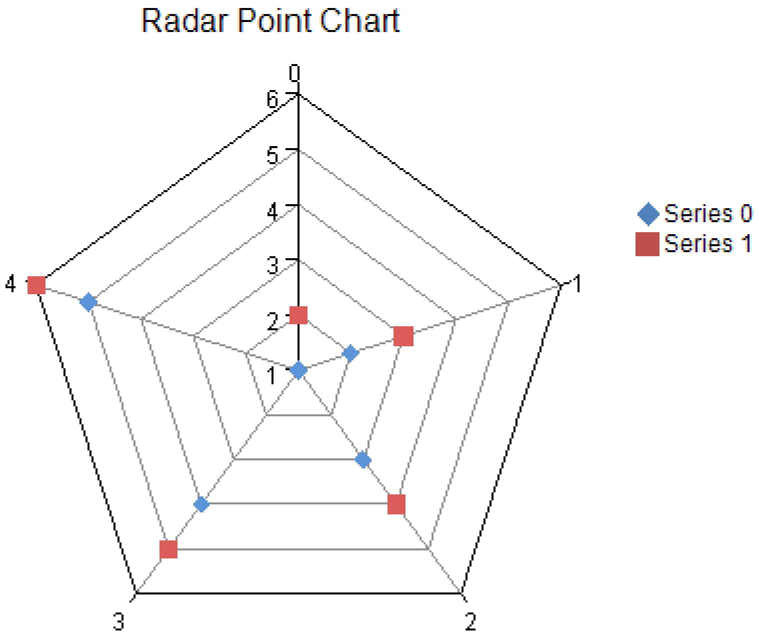
You can have any of these types of Radar plots:

- **Point Charts**
- **Line Charts**
- **Area Charts**
- **Stripe Charts**

For details on the API, refer to the RadarPlotArea class.

Point Charts

The point chart can be a radar plot such as the one shown in this figure.



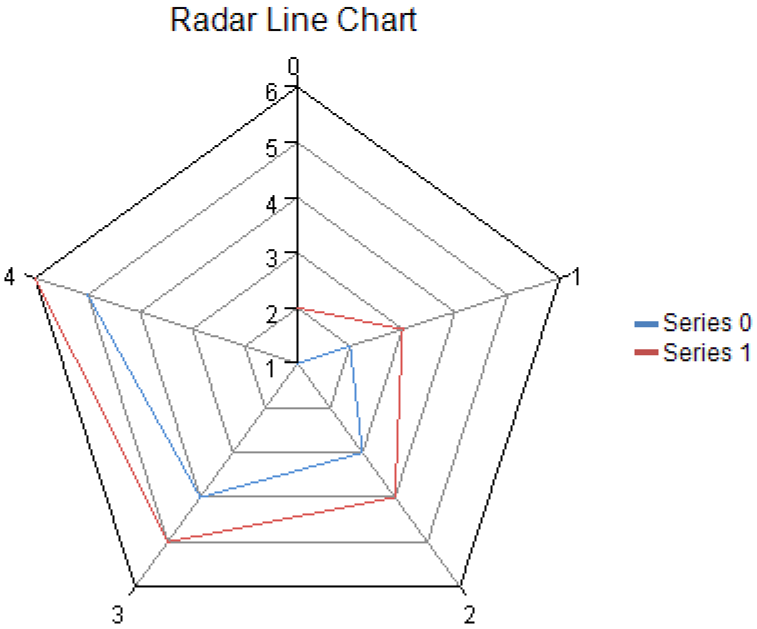
Each point has a single data value: y. Each point is visualized as a point marker.

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 class.

Line Charts

The line chart can be a radar plot such as the one shown in this figure.



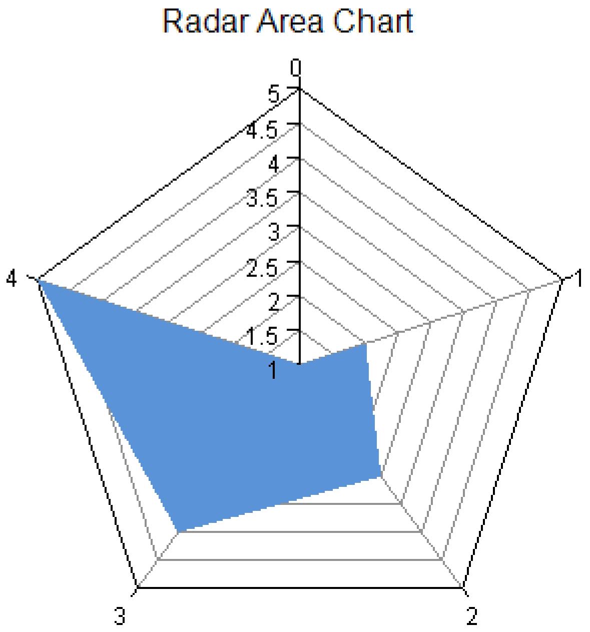
Each point has a single data value: y. Each point is visualized as a point on a line.

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 class.

Area Charts

The area chart can be a radar plot such as the one shown in this figure.



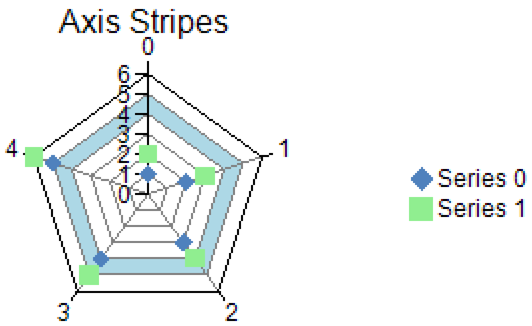
Each point has a single data value: y. Each point is visualized as a point on an area.

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 class.

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.

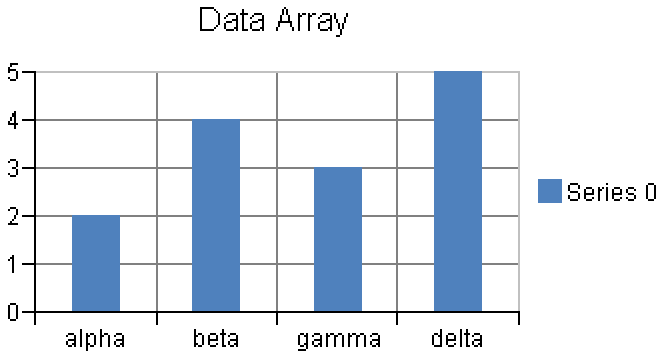
For more information on the stripe object in the API, refer to the Stripe class.

Data Plot Types

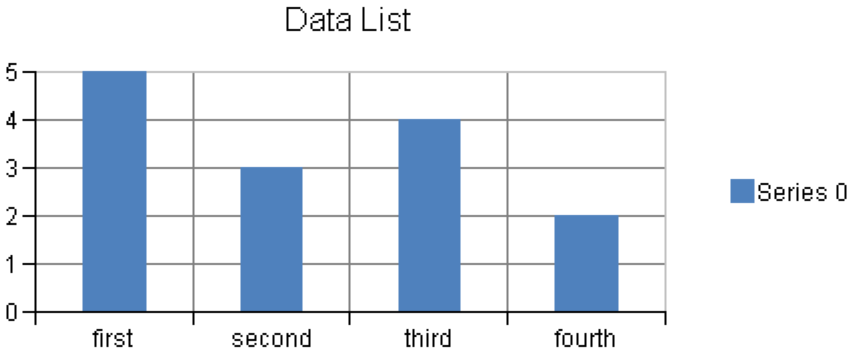
Data charts are charts that can be bound. Any series in any of the plot types can be bound to a data source using the data source property in the series class. You can use any of these types of data sources:

- Array
- List
- Table

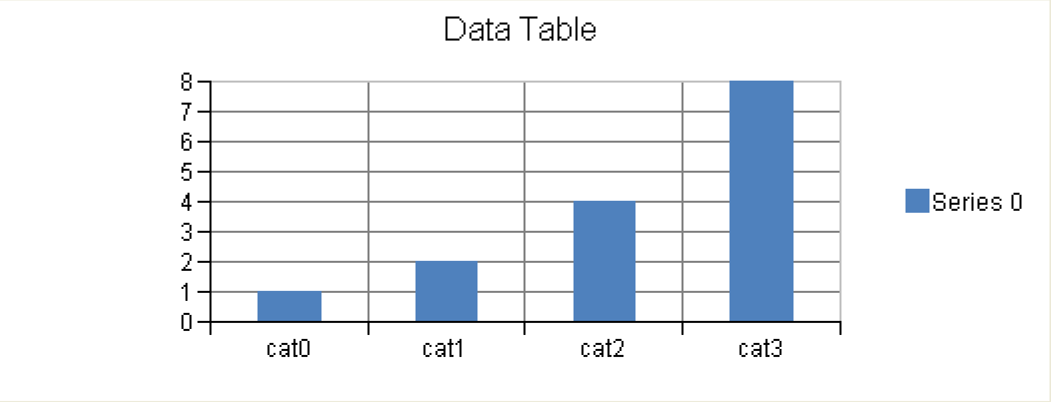
The array data chart can be a one-dimensional plot such as the one shown in this figure. This array chart shows the bars alongside each other vertically.



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:



Series

A series is a collection of data points that are displayed in the plot area. There are several general types of series: Y, XY, XYZ, Pie, Polar, Radar, Sunburst, and Treemap. These types of series correspond to the main types of plot areas. Each main type of series may have additional subtypes of series (such as area, line, or point).

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.

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 (for example, 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 for more information:

- AreaSeries
- BarSeries
- **FunnelSeries ('FunnelSeries Class' in the on-line documentation)**
- HighLowAreaSeries
- HighLowBarSeries
- HighLowCloseSeries
- **HistogramSeries ('HistogramSeries Class' in the on-line documentation)**
- LineSeries
- **ParetoSeries ('ParetoSeries Class' in the on-line documentation)**
- PieSeries
- PointSeries
- PolarSeries
- RadarSeries
- **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
- YSeries

Using Code

Add values to the chart with a bar series.

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.Web.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.Web.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

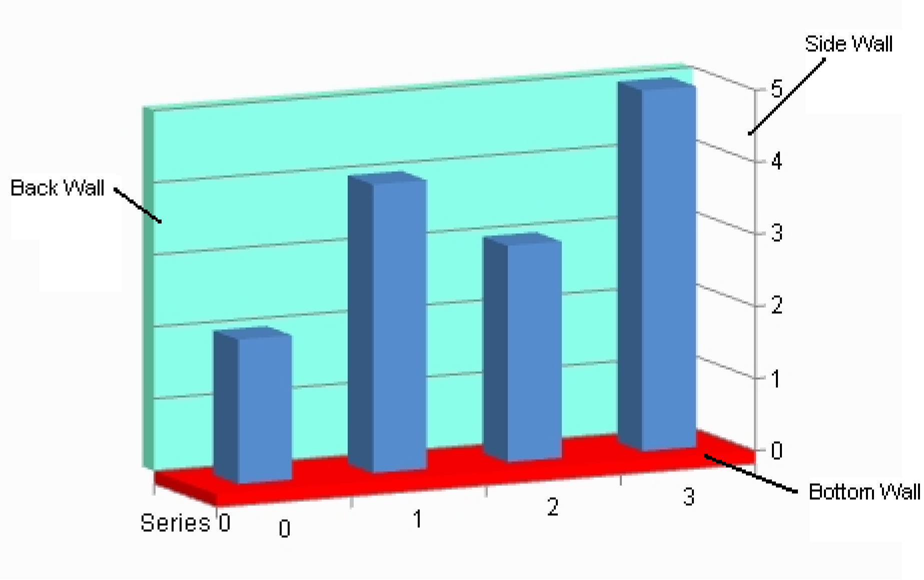
1. Select the **PlotArea Collection** editor.
2. Select the **Series Collection** editor.
3. Set properties as needed.
4. Select **Apply** and **OK** to close the Chart Designer.

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 classes for more information on how to set properties for walls:

- Wall
- 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

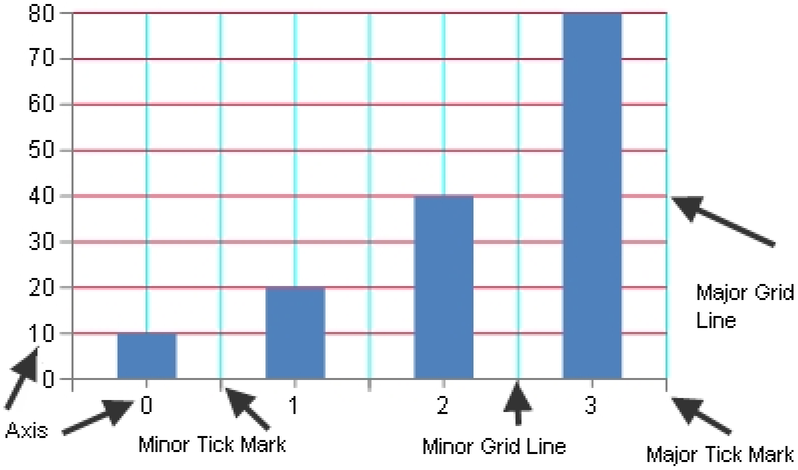
```
Dim plotArea As New FarPoint.Web.Chart.YPlotArea()
plotArea.BackWall.Visible = True
```

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.
4. Select **Apply** and **OK** to close the Chart Designer.

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/minor tick marks, tick mark labels, major/minor grid lines, and stripes. The direction of the axis can be reversed. The minimum, maximum, major/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
- 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:

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.Web.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.
4. Select **Apply** and **OK** to close the Chart Designer.

Fill Effects

A fill effect is when the interior of an object is painted. Three types of fill effects are solid, gradient, and image. A solid fill effect uses a single color and a gradient fill uses two colors and a direction. An image fill effect uses an image instead of a color for the fill.

The following elements can have fill effects: 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:

- No Fill
- Solid Fill
- Image Fill
- Gradient Fill

You can fill elements using the **Fill** property in the following classes:

- **ChartModel ('Fill Property' in the on-line documentation)**
- **LabelArea ('Fill Property' in the on-line documentation)**
- **LegendArea ('Fill Property' in the on-line documentation)**
- **Stripe ('Fill Property' in the on-line documentation)**
- **Wall ('Fill Property' in the on-line documentation)**

To set the fill effect for the entire plot area, you can set the **Fill ('Fill Property' in the on-line documentation)** 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 ('BackWall Property' in the on-line documentation)** property in the YPlotArea class.

Using Code

1. Create a series.
2. Use fill properties to change the color of the bars in the chart.
3. Add the series to the plot area.

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);
FarPoint.Web.Spread.Chart.SpreadChart chart = new
FarPoint.Web.Spread.Chart.SpreadChart();
chart.Model = model;
FpSpread1.Sheets[0].Charts.Add(chart);
```

VB

```
Dim series As New FarPoint.Web.Chart.BarSeries()
series.BarFill = New FarPoint.Web.Chart.SolidFill(System.Drawing.Color.Red)
series.Values.Add(2.0)
series.Values.Add(4.0)
series.Values.Add(3.0)
series.Values.Add(5.0)
Dim plotArea As New FarPoint.Web.Chart.YPlotArea()
plotArea.Location = New System.Drawing.PointF(0.2F, 0.2F)
plotArea.Size = New System.Drawing.SizeF(0.6F, 0.6F)
plotArea.Series.Add(series)
Dim model As New FarPoint.Web.Chart.ChartModel()
model.PlotAreas.Add(plotArea)
Dim chart As New FarPoint.Web.Spread.Chart.SpreadChart()
chart.Model = model
FpSpread1.Sheets(0).Charts.Add(chart)
```

Using Code

You can set the fill effect before or after adding the data points if you set the fill effect for the entire series.

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);
\\ OR
BarSeries series = new BarSeries();
series.Values.Add(2.0);
series.BarFill = new SolidFill(Color.Red);
```

VB

```
Dim series As New FarPoint.Web.Chart.BarSeries()
```

```
series.BarFill = New SolidFill(Color.Red)
series.Values.Add(2.0)
' OR
Dim series As New FarPoint.Web.Chart.BarSeries()
series.Values.Add(2.0)
series.BarFill = New SolidFill(Color.Red)
```

Using Code

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.

Example

The following example sets a fill effect for a bar.

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.Web.Chart.BarSeries()
series.Values.Add(2.0)
series.Values.Add(4.0)
series.BarFills.Add(New SolidFill(Color.Green))
```

Using Code

You can assign fill effects for lines and markers as well.

Example

The following example sets a fill effect for a marker.

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

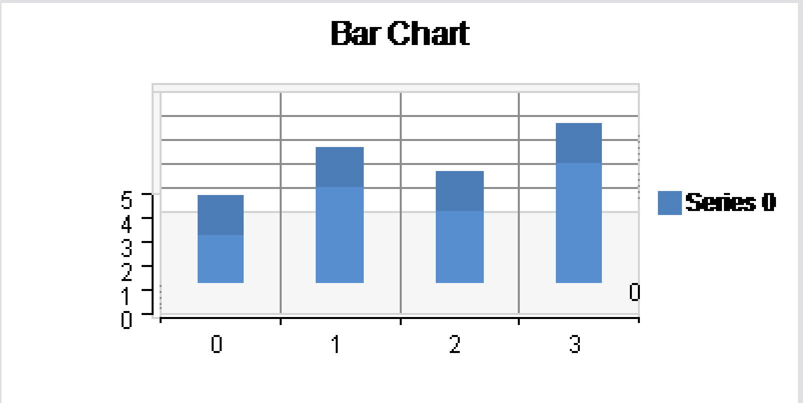
```
Dim series As New PointSeries()
series.PointFill = New SolidFill(Color.Lime)
series.PointBorder = New SolidLine(Color.Red)
series.PointMarker = New BuiltinMarker(MarkerShape.Triangle, 10F)
series.Values.Add(2.0)
series.Values.Add(4.0)
series.Values.Add(3.0)
series.Values.Add(5.0)
```

Using the Chart Designer

1. Create a chart with values.
2. Select the series.
3. Select the **Fill** option.
4. Set properties as needed.
5. Select **Apply** and **OK** to close the Chart Designer.

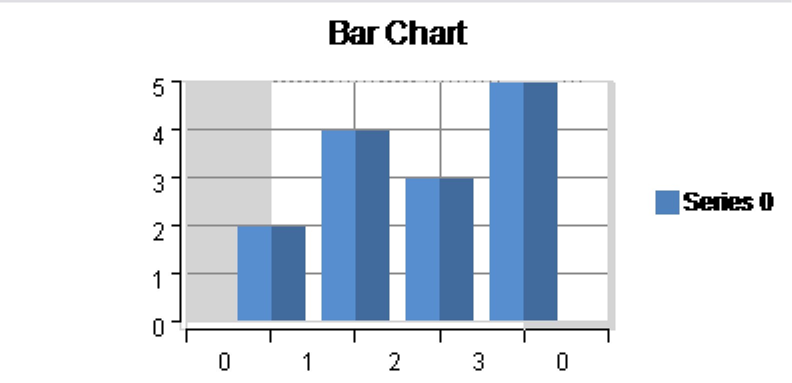
Elevation and Rotation

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.

Using Code

You can set the **Elevation ('Elevation Property' in the on-line documentation)** and **Rotation ('Rotation Property' in the on-line documentation)** properties.

Example

The following example sets **Elevation ('Elevation Property' in the on-line documentation)** and **Rotation ('Rotation Property' in the on-line documentation)**. Use this code with a 3D chart.

C#

```
YPlotArea plotArea = new YPlotArea();
plotArea.Location = new PointF(0.2f, 0.2f);
plotArea.Size = new SizeF(0.6f, 0.6f);
plotArea.Rotation = 20;
plotArea.Elevation = 15;
```

VB

```
Dim plotArea As New FarPoint.Web.Chart.YPlotArea()
plotArea.Location = New System.Drawing.PointF(0.2F, 0.2F)
plotArea.Size = New System.Drawing.SizeF(0.6F, 0.6F)
plotArea.Rotation = 20
plotArea.Elevation = 15
```

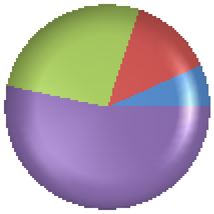
Lighting, Shapes, and Borders

You can set borders for most areas of the chart. See the **LineBorder ('LineBorder Property' in the on-line documentation)** and **PointBorder ('PointBorder Property' in the on-line documentation)** properties in the LineSeries class for more information. The XYBubbleSeries class has additional border settings such as **NegativeBorder ('NegativeBorder Property' in the on-line documentation)** and **PositiveBorder ('PositiveBorder Property' in the on-line documentation)**.

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 ('AmbientColor Property' in the on-line documentation)**, **DiffuseColor ('DiffuseColor Property' in the on-line documentation)**, and **SpecularColor ('SpecularColor Property' in the on-line documentation)** in the **PositionalLight ('PositionalLight Class' in the on-line documentation)** class.
5. Set the **PositionX ('PositionX Property' in the on-line documentation)**, **PositionY ('PositionY Property' in the on-line documentation)**, and **PositionZ ('PositionZ Property' in the on-line documentation)** properties in the **PositionalLight ('PositionalLight Class' in the on-line documentation)** class.
6. Set the **AmbientColor ('AmbientColor Property' in the on-line documentation)**, **DiffuseColor ('DiffuseColor Property' in the on-line documentation)**, and **SpecularColor ('SpecularColor Property' in the on-line documentation)** in the **DirectionalLight ('DirectionalLight Class' in the on-line documentation)** class.
7. Set the **DirectionX ('DirectionX Property' in the on-line documentation)**, **DirectionY ('DirectionY Property' in the on-line documentation)**, and **DirectionZ ('DirectionZ Property' in the on-line documentation)** 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.
11. Add the chart to the Spread control.

Example

The following example demonstrates using light colors, direction, and position.

C#

```
FarPoint.Web.Chart.PieSeries series = new FarPoint.Web.Chart.PieSeries();
series.SeriesName = "Series 1";
series.TopBevel = new FarPoint.Web.Chart.CircleBevel(12.0f, 12.0f);
series.BottomBevel = new FarPoint.Web.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.Web.Chart.PiePlotArea plotArea = new FarPoint.Web.Chart.PiePlotArea();
plotArea.Location = new System.Drawing.PointF(0.2f, 0.2f);
plotArea.Size = new System.Drawing.SizeF(0.6f, 0.6f);
plotArea.Series.Add(series);
FarPoint.Web.Chart.PositionalLight light0 = new FarPoint.Web.Chart.PositionalLight();
light0.AmbientColor = System.Drawing.Color.FromArgb(64, 64, 64);
light0.DiffuseColor = System.Drawing.Color.FromArgb(64, 64, 64);
light0.SpecularColor = System.Drawing.Color.FromArgb(128, 128, 128);
light0.PositionX = 0.0f;
light0.PositionY = 0.0f;
```

```
light0.PositionZ = 100.0f;
FarPoint.Web.Chart.DirectionalLight light1 = new FarPoint.Web.Chart.DirectionalLight();
light1.AmbientColor = System.Drawing.Color.FromArgb(64, 64, 64);
light1.DiffuseColor = System.Drawing.Color.FromArgb(64, 64, 64);
light1.SpecularColor = System.Drawing.Color.FromArgb(128, 128, 128);
light1.DirectionX = 1.0f;
light1.DirectionY = 0.0f;
light1.DirectionZ = 1.0f;
FarPoint.Web.Chart.ChartModel model = new FarPoint.Web.Chart.ChartModel();
model.PlotAreas.Add(plotArea);
model.PlotAreas[0].Lights.Clear();
model.PlotAreas[0].Lights.Add(light0);
model.PlotAreas[0].Lights.Add(light1);
FarPoint.Web.Spread.Chart.SpreadChart chart = new
FarPoint.Web.Spread.Chart.SpreadChart();
chart.Model = model;
chart.ViewType = FarPoint.Web.Chart.ChartViewType.View3D;
FpSpread1.Sheets[0].Charts.Add(chart);
```

VB

```
Dim series As New FarPoint.Web.Chart.PieSeries()
series.SeriesName = "Series 1"
series.TopBevel = New FarPoint.Web.Chart.CircleBevel(12.0F, 12.0F)
series.BottomBevel = New FarPoint.Web.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.Web.Chart.PiePlotArea()
plotArea.Location = New System.Drawing.PointF(0.2F, 0.2F)
plotArea.Size = New System.Drawing.SizeF(0.6F, 0.6F)
plotArea.series.Add(series)
Dim light0 As New FarPoint.Web.Chart.PositionalLight()
light0.AmbientColor = System.Drawing.Color.FromArgb(64, 64, 64)
light0.DiffuseColor = System.Drawing.Color.FromArgb(64, 64, 64)
light0.SpecularColor = System.Drawing.Color.FromArgb(128, 128, 128)
light0.PositionX = 0.0F
light0.PositionY = 0.0F
light0.PositionZ = 100.0F
Dim light1 As New FarPoint.Web.Chart.DirectionalLight()
light1.AmbientColor = System.Drawing.Color.FromArgb(64, 64, 64)
light1.DiffuseColor = System.Drawing.Color.FromArgb(64, 64, 64)
light1.SpecularColor = System.Drawing.Color.FromArgb(128, 128, 128)
light1.DirectionX = 1.0F
light1.DirectionY = 0.0F
light1.DirectionZ = 1.0F
Dim model As New FarPoint.Web.Chart.ChartModel()
model.PlotAreas.Add(plotArea)
model.PlotAreas(0).Lights.Clear()
model.PlotAreas(0).Lights.Add(light0)
model.PlotAreas(0).Lights.Add(light1)
Dim chart As New FarPoint.Web.Spread.Chart.SpreadChart()
chart.Model = model
chart.ViewType = FarPoint.Web.Chart.ChartViewType.View3D
FpSpread1.Sheets(0).Charts.Add(chart)
```

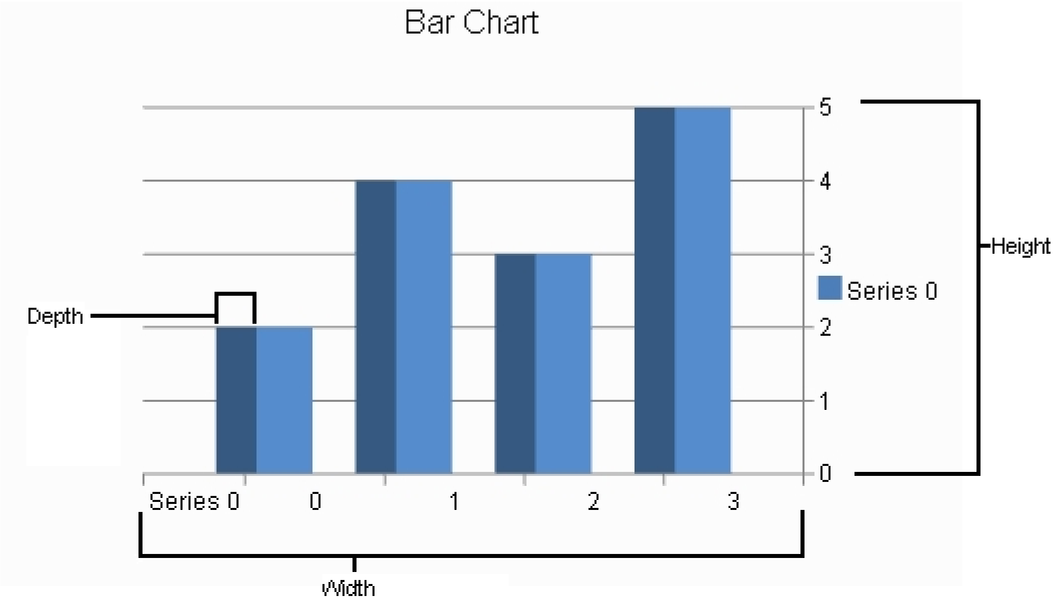
Using the Chart Designer

1. Select the **Plot Areas Collection**.
2. Select the **Light Collection** editor.
3. Set properties as needed.
4. Select **Apply** and **OK** to close the Chart Designer.

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)** (height and width)
- **Depth ('Depth Property' in the on-line documentation)**

Using Code

Use properties in the plot area classes 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.Web.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.
3. Select **Apply** and **OK** to close the Chart Designer.

Labels

The labels contain the plot title and the axis labels. You can set the main title for the chart using the **Text ('Text Property' in the on-line documentation)** property in the LabelArea 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
- IndexAxis
- 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 ('Title Property' in the on-line documentation)** property.
3. Set **TitleTextFill ('TitleTextFill Property' in the on-line documentation)** and **TitleTextFont ('TitleTextFont Property' in the on-line documentation)** for additional formatting. You can also set the **TitleOffset ('TitleOffset Property' in the on-line documentation)**.
4. To set the title in the plot area for the y-axis, set the ValueAxis.**Title ('Title Property' in the on-line documentation)** property.
5. Set **TitleTextFill ('TitleTextFill Property' in the on-line documentation)** and **TitleTextFont ('TitleTextFont Property' in the on-line documentation)** for additional formatting. You can also set the **TitleOffset ('TitleOffset Property' in the on-line documentation)**.

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.Web.Chart.SolidFill(Drawing.Color.Crimson);
plotArea.XAxis.TitleVisible = true;
plotArea.YAxis[0].Title = "Values";
plotArea.YAxes[0].TitleTextFont = new System.Drawing.Font("Comic Sans MS", 12);
plotArea.YAxes[0].TitleTextFill = new
FarPoint.Web.Chart.SolidFill(Drawing.Color.Chartreuse);
```

VB

```
Dim plotArea As New FarPoint.Web.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.Web.Chart.SolidFill(Drawing.Color.Crimson)
plotArea.XAxis.TitleVisible = True
plotArea.YAxis(0).Title = "Values"
plotArea.YAxes(0).TitleTextFont = New System.Drawing.Font("Comic Sans MS", 12)
plotArea.YAxes(0).TitleTextFill = New
FarPoint.Web.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.
4. Select **Apply** and **OK** to close the Chart Designer.

Legends

The legend contains identifiers for each of the data series. 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
- LegendAreaCollection

Using Code

Use the **Location ('Location Property' in the on-line documentation)**, **AlignmentX ('AlignmentX Property' in the on-line documentation)**, and **AlignmentY ('AlignmentY Property' in the on-line documentation)** 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.Web.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 the properties as needed.
3. Select **Apply** and **OK** to close the Chart Designer.

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**
- **Using the Chart Designer**
- **Using the Spread Designer**
- **Using the Chart Control**
- **Using the Chart Control in Spread**

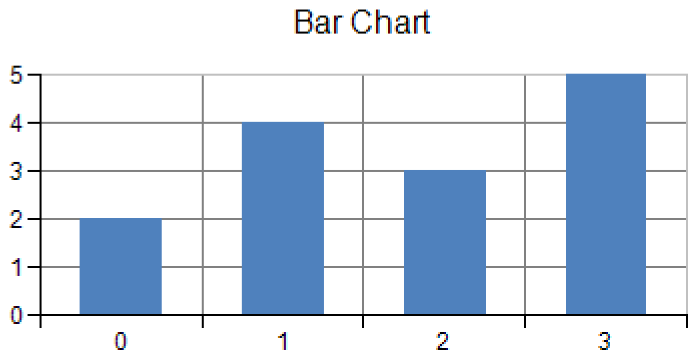
Creating Plot Types

Read the following sections for more information and instructions:

- **Creating a Y Plot**
- **Creating an XY Plot**
- **Creating an XYZ Plot**
- **Creating a Pie Plot**
- **Creating a Polar Plot**
- **Creating a Radar Plot**
- **Creating a Sunburst Chart (on-line documentation)**
- **Creating a Treemap Chart (on-line documentation)**
- **Combining Plot Types**

Creating a Y Plot

You can create a Y Plot Chart using code or the designer. The following image shows a YPlot bar type chart.



For details on the API, see the YPlotArea class.

The following classes are also available:

- AreaSeries
- BarSeries
- **BoxWhiskerSeries ('BoxWhiskerSeries Class' in the on-line documentation)**
- HighLowAreaSeries
- HighLowBarSeries
- HighLowCloseSeries
- **HistogramSeries ('HistogramSeries Class' in the on-line documentation)**
- LineSeries
- **ParetoSeries ('ParetoSeries Class' in the on-line documentation)**
- PointSeries
- **WaterfallSeries ('WaterfallSeries Class' in the on-line documentation)**
- YSeries

Using Code

1. Use the BarSeries class to add data to a Chart control.
2. Use the YPlotArea 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.
8. Add the chart to the Spread control.

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);
FarPoint.Web.Spread.Chart.SpreadChart chart = new
FarPoint.Web.Spread.Chart.SpreadChart();
chart.Model = model;
FpSpread1.Sheets[0].Charts.Add(chart);
```

VB

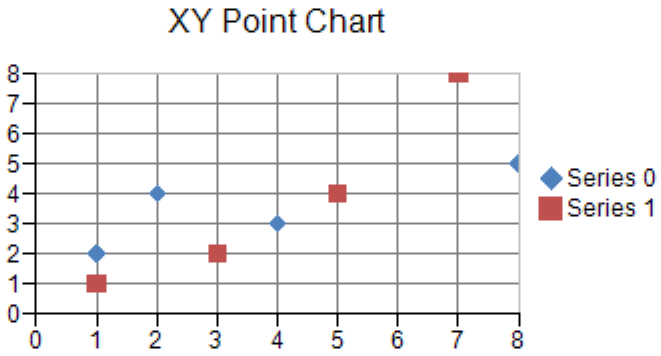
```
Dim series As New FarPoint.Web.Chart.BarSeries()
series.SeriesName = "Series 0"
series.Values.Add(2.0)
series.Values.Add(4.0)
series.Values.Add(3.0)
series.Values.Add(5.0)
Dim plotArea As New FarPoint.Web.Chart.YPlotArea()
plotArea.Location = New PointF(0.2F, 0.2F)
plotArea.Size = New SizeF(0.6F, 0.6F)
plotArea.Series.Add(series)
Dim label As New FarPoint.Web.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.Web.Chart.LegendArea()
legend.Location = New PointF(0.98F, 0.5F)
legend.AlignmentX = 1.0F
legend.AlignmentY = 0.5F
Dim model As New FarPoint.Web.Chart.ChartModel()
model.LabelAreas.Add(label)
model.LegendAreas.Add(legend)
model.PlotAreas.Add(plotArea)
Dim chart As New FarPoint.Web.Spread.Chart.SpreadChart()
chart.Model = model
FpSpread1.Sheets(0).Charts.Add(chart)
```

Using the Chart Designer

1. Select the **PlotArea Collection** editor.
2. Set properties as needed.
3. Select **Apply** and **OK** to close the Chart Designer.

Creating an XY Plot

You can create an XY Plot Chart using code or the designer. The following image shows an XYPlot 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
- XYLineSeries
- XYPointSeries

Using Code

1. Use the XYPointSeries class to add data to a Chart control.
2. Use the XYPlotArea 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.
8. Add the chart to the Spread control.

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);
FarPoint.Web.Spread.Chart.SpreadChart chart = new
FarPoint.Web.Spread.Chart.SpreadChart();
chart.Model = model;
FpSpread1.Sheets[0].Charts.Add(chart);
```

VB

```
Dim series0 As New FarPoint.Web.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.Web.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.Web.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)
```

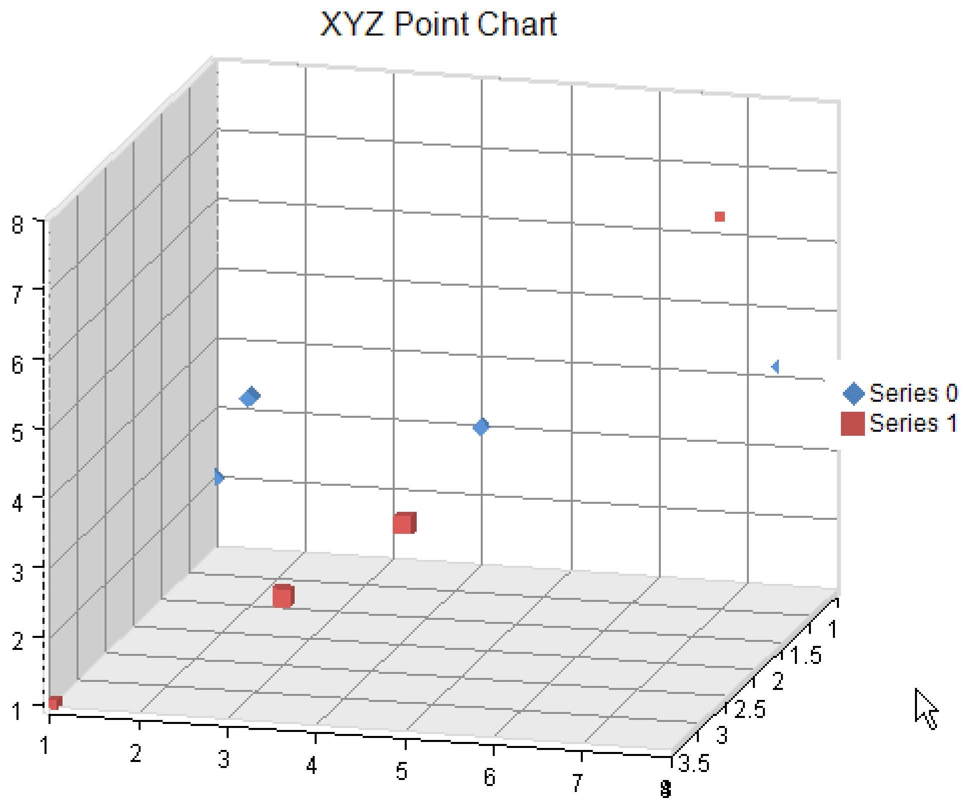
```
Dim label As New FarPoint.Web.Chart.LabelArea()
label.Text = "XY Point Chart"
label.Location = New PointF(0.5F, 0.02F)
label.AlignmentX = 0.5F
label.AlignmentY = 0.0F
Dim legend As New FarPoint.Web.Chart.LegendArea()
legend.Location = New PointF(0.98F, 0.5F)
legend.AlignmentX = 1.0F
legend.AlignmentY = 0.5F
Dim model As New FarPoint.Web.Chart.ChartModel()
model.LabelAreas.Add(label)
model.LegendAreas.Add(legend)
model.PlotAreas.Add(plotArea)
Dim chart As New FarPoint.Web.Spread.Chart.SpreadChart()
chart.Model = model
FpSpread1.Sheets(0).Charts.Add(chart)
```

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.
4. Select **Apply** and **OK** to close the Chart Designer.

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 class.

The following classes are also available when creating XYZ plot type charts:

- XYZSeries
- XYZPointSeries
- XYZLineSeries
- XYZSurfaceSeries

Using Code

1. Use the XYZPointSeries class to add data to a Chart control.
2. Use the XYZPlotArea 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.
8. Add the chart to the Spread control.

Example

The following example demonstrates using unbound data to create an XYZ point chart.

C#

```
XYZPointSeries series0 = new 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)
XYZPointSeries series1 = new 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);
XYZPlotArea plotArea = new XYZPlotArea();
plotArea.Location = new PointF(0.2f, 0.2f);
plotArea.Size = new SizeF(0.6f, 0.6f);
plotArea.Series.Add(series0);
plotArea.Series.Add(series1);
LabelArea label = new LabelArea();
label.Text = "XYZ 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);
FarPoint.Web.Spread.Chart.SpreadChart chart = new
FarPoint.Web.Spread.Chart.SpreadChart();
chart.Model = model;
FpSpread1.Sheets[0].Charts.Add(chart);
```

VB

```
Dim series0 As New FarPoint.Web.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.Web.Chart.XYZPointSeries()
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)
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.Web.Chart.XYZPlotArea()
plotArea.Location = New PointF(0.2F, 0.2F)
plotArea.Size = New SizeF(0.6F, 0.6F)
plotArea.Series.Add(series0)
plotArea.Series.Add(series1)
Dim label As New FarPoint.Web.Chart.LabelArea()
```

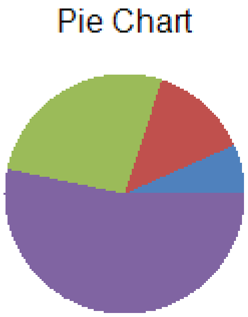
```
label.Text = "XYZ Point Chart"
label.Location = New PointF(0.5F, 0.02F)
label.AlignmentX = 0.5F
label.AlignmentY = 0.0F
Dim legend As New FarPoint.Web.Chart.LegendArea()
legend.Location = New PointF(0.98F, 0.5F)
legend.AlignmentX = 1.0F
legend.AlignmentY = 0.5F
Dim model As New FarPoint.Web.Chart.ChartModel()
model.LabelAreas.Add(label)
model.LegendAreas.Add(legend)
model.PlotAreas.Add(plotArea)
Dim chart As New FarPoint.Web.Spread.Chart.SpreadChart()
chart.Model = model
FpSpread1.Sheets(0).Charts.Add(chart)
```

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.
4. Select **Apply** and **OK** to close the Chart Designer.

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 class.

The following class is also available:

- PieSeries

Using Code

1. Use the PieSeries class to add data to a Chart control.
2. Use the PiePlotArea 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.
8. Add the chart to the Spread control.

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);
FarPoint.Web.Spread.Chart.SpreadChart chart = new
FarPoint.Web.Spread.Chart.SpreadChart();
chart.Model = model;
FpSpread1.Sheets[0].Charts.Add(chart);
```

VB

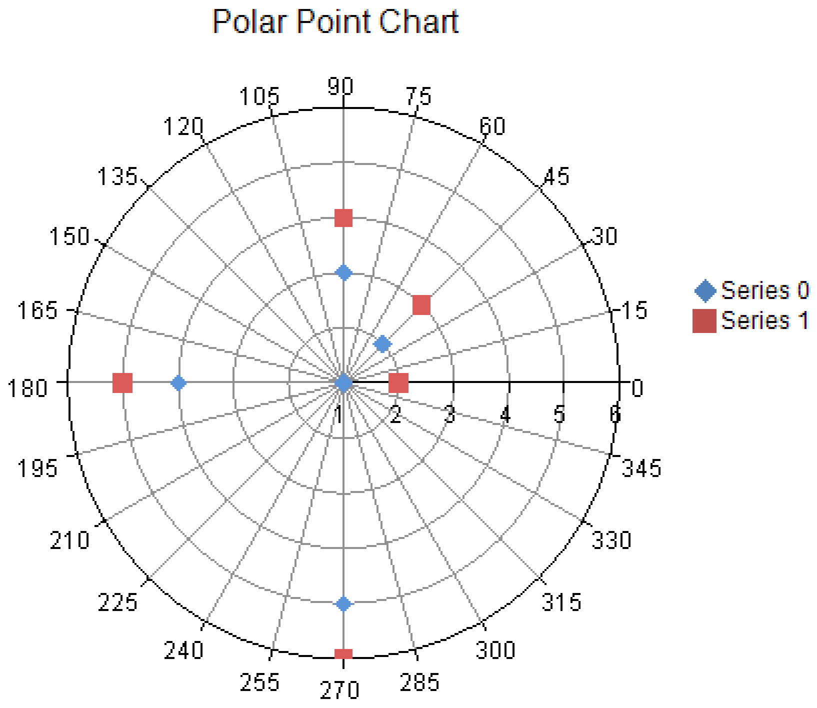
```
Dim series As New FarPoint.Web.Chart.PieSeries()
series.SeriesName = "Series 0"
series.Values.Add(1.0)
series.Values.Add(2.0)
series.Values.Add(4.0)
series.Values.Add(8.0)
Dim plotArea As New FarPoint.Web.Chart.PiePlotArea()
plotArea.Location = New PointF(0.2F, 0.2F)
plotArea.Size = New SizeF(0.6F, 0.6F)
plotArea.Series.Add(series)
Dim label As New FarPoint.Web.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.Web.Chart.LegendArea()
legend.Location = New PointF(0.98F, 0.5F)
legend.AlignmentX = 1.0F
legend.AlignmentY = 0.5F
Dim model As New FarPoint.Web.Chart.ChartModel()
model.LabelAreas.Add(label)
model.LegendAreas.Add(legend)
model.PlotAreas.Add(plotArea)
Dim chart As New FarPoint.Web.Spread.Chart.SpreadChart()
chart.Model = model
FpSpread1.Sheets(0).Charts.Add(chart)
```

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.
4. Select **Apply** and **OK** to close the Chart Designer.

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 class.

The following classes are also available:

- PolarAreaSeries
- PolarSeries
- PolarLineSeries
- PolarPointSeries
- PolarAngleAxis
- PolarRadiusAxis

Using Code

1. Use the PolarPointSeries class to add data to a Chart control.

2. Use the PolarPlotArea 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.
8. Add the chart to the Spread control.

Example

The following example demonstrates using unbound data to create a polar point series chart.

C#

```
PolarPointSeries series0 = new 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);
PolarPointSeries series1 = new 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);
PolarPlotArea plotArea = new 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);
LabelArea label = new LabelArea();
label.Text = "Polar 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);
```

```
FarPoint.Web.Spread.Chart.SpreadChart chart = new
FarPoint.Web.Spread.Chart.SpreadChart();
chart.Model = model;
FpSpread1.Sheets[0].Charts.Add(chart);
```

VB

```
Dim series0 As New 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 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 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 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 LegendArea()
legend.Location = New PointF(0.98F, 0.5F)
legend.AlignmentX = 1F
legend.AlignmentY = 0.5F
Dim model As New ChartModel()
model.LabelAreas.Add(label)
model.LegendAreas.Add(legend)
model.PlotAreas.Add(plotArea)
Dim chart As New FarPoint.Web.Spread.Chart.SpreadChart()
chart.Model = model
FpSpread1.Sheets(0).Charts.Add(chart)
```

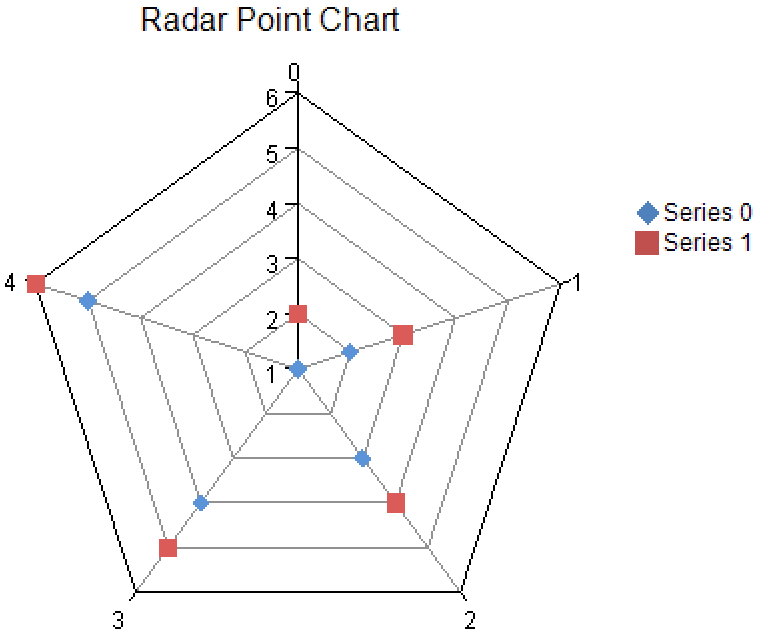
Using the Chart Designer

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.
4. Select **Apply** and **OK** to close the Chart Designer.

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 class.

The following classes are also available:

- RadarAreaSeries
- RadarSeries
- RadarLineSeries
- RadarPointSeries
- RadarIndexAxis
- RadarValueAxis

Using Code

1. Use the RadarPointSeries class to add data to a Chart control.
2. Use the RadarPlotArea 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.
8. Add the chart to the Spread control.

Example

The following example demonstrates using unbound data to create a Radar chart.

C#

```
RadarPointSeries series0 = new 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);
RadarPointSeries series1 = new 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);
RadarPlotArea plotArea = new 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);
LabelArea label = new LabelArea();
label.Text = "Radar 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);
FarPoint.Web.Spread.Chart.SpreadChart chart = new
FarPoint.Web.Spread.Chart.SpreadChart();
chart.Model = model;
FpSpread1.Sheets[0].Charts.Add(chart);
```

VB

```
Dim series0 As New FarPoint.Web.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.Web.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.Web.Chart.RadarPlotArea()
```

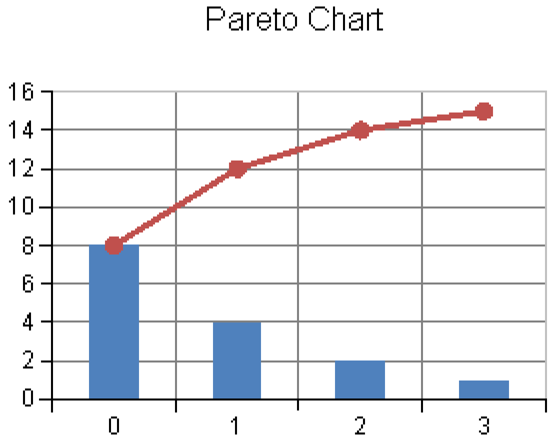
```
plotArea.Location = New PointF(0.2F, 0.2F)
plotArea.Size = New SizeF(0.6F, 0.6F)
plotArea.Series.Add(series0)
plotArea.Series.Add(series1)
Dim label As New FarPoint.Web.Chart.LabelArea()
label.Text = "Radar Point Chart"
label.Location = New PointF(0.5F, 0.02F)
label.AlignmentX = 0.5F
label.AlignmentY = 0.0F
Dim legend As New FarPoint.Web.Chart.LegendArea()
legend.Location = New PointF(0.98F, 0.5F)
legend.AlignmentX = 1.0F
legend.AlignmentY = 0.5F
Dim model As New FarPoint.Web.Chart.ChartModel()
model.LabelAreas.Add(label)
model.LegendAreas.Add(legend)
model.PlotAreas.Add(plotArea)
Dim chart As New FarPoint.Web.Spread.Chart.SpreadChart()
chart.Model = model
FpSpread1.Sheets(0).Charts.Add(chart)
```

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.
4. Select **Apply** and **OK** to close the Chart Designer.

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 BarSeries and a LineSeries can be combined together in a YPlotArea.



For details on the API, see the YPlotArea class.

The following classes are used to create the bar and line series example:

- BarSeries
- LineSeries

Using Code

1. Use the BarSeries and LineSeries classes to add data to the Chart control.
2. Use the YPlotArea 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.
8. Add the chart to the Spread control.

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 labelArea = new LabelArea();
labelArea.Location = new PointF(0.5f, 0.02f);
labelArea.AlignmentX = 0.5f;
labelArea.AlignmentY = 0.0f;
labelArea.Text = "Pareto Chart";
ChartModel model = new ChartModel();
model.LabelAreas.Add(labelArea);
model.PlotAreas.Add(plotArea);
FarPoint.Web.Spread.Chart.SpreadChart chart = new
FarPoint.Web.Spread.Chart.SpreadChart();
chart.Model = model;
FpSpread1.Sheets[0].Charts.Add(chart);
```

VB

```
Dim series0 As New FarPoint.Web.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.Web.Chart.LineSeries()
series1.PointMarker = New
FarPoint.Web.Chart.BuiltinMarker(FarPoint.Web.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.Web.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.Web.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.Web.Chart.ChartModel()
model.LabelAreas.Add(labelArea)
model.PlotAreas.Add(plotArea)
Dim chart As New FarPoint.Web.Spread.Chart.SpreadChart()
chart.Model = model
FpSpread1.Sheets(0).Charts.Add(chart)
```

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 Data Versus 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 (on-line documentation)** (for example: **SeriesNameDataSource** in the RadarLineSeries class).

Using Code

Use the **Values ('Values Property' in the on-line documentation)** property of the series to add a data source.

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

Use the **Values ('Values Property' in the on-line documentation)** property of the series to add a data source.

Example

The following example demonstrates how to bind the control to a data table.

C#

```
System.Data.DataTable dt = new System.Data.DataTable("Test");
System.Data.DataRow dr = default(System.Data.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.Web.Chart.BarSeries series = new FarPoint.Web.Chart.BarSeries();
series.Values.DataSource = dt;
series.Values.DataField = dt.Columns[0].ColumnName;
FarPoint.Web.Chart.YPlotArea plotArea = new FarPoint.Web.Chart.YPlotArea();
FarPoint.Web.Chart.ChartModel model = new FarPoint.Web.Chart.ChartModel();
plotArea.Location = new System.Drawing.PointF(0.2F, 0.2F);
plotArea.Size = new System.Drawing.SizeF(0.6F, 0.6F);
plotArea.Series.Add(series);
model.PlotAreas.Add(plotArea);
FarPoint.Web.Spread.Chart.SpreadChart chart = new
FarPoint.Web.Spread.Chart.SpreadChart();
chart.Model = model;
FpSpread1.Sheets[0].Charts.Add(chart);
```

VB

```
Dim dt As New System.Data.DataTable("Test")
Dim dr As System.Data.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.Web.Chart.BarSeries
series.Values.DataSource = dt
series.Values.DataField = dt.Columns(0).ColumnName
Dim model As New FarPoint.Web.Chart.ChartModel()
Dim plotArea As New FarPoint.Web.Chart.YPlotArea()
plotArea.Location = New System.Drawing.PointF(0.2F, 0.2F)
plotArea.Size = New System.Drawing.SizeF(0.6F, 0.6F)
plotArea.Series.Add(series)
model.PlotAreas.Add(plotArea)
Dim chart As New FarPoint.Web.Spread.Chart.SpreadChart()
chart.Model = model
FpSpread1.Sheets(0).Charts.Add(chart)
```

Using an Unbound Data Source

You can add double values to the Chart control without using a data source.

Using Code

Use the **Values ('Values Property' in the on-line documentation)** property of the series to add data.

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.
5. Select **Apply** and **OK** to close the Chart Designer.

Using Raw Data Versus 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 ('DisplayUnits Property' in the on-line documentation)** property in the ValueAxis class to set the scale.

Using Code

Use the **DisplayUnits ('DisplayUnits Property' in the on-line documentation)** property to create a smaller scale on the axis.

Example

The following example uses the **DisplayUnits ('DisplayUnits Property' in the on-line documentation)** property.

C#

```
FarPoint.Web.Chart.BarSeries series = new FarPoint.Web.Chart.BarSeries();
series.Values.Add(10000.0);
series.Values.Add(20000.0);
series.Values.Add(40000.0);
series.Values.Add(80000.0);
FarPoint.Web.Chart.YPlotArea plotArea = new FarPoint.Web.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.Web.Chart.ChartModel model = new FarPoint.Web.Chart.ChartModel();
model.PlotAreas.Add(plotArea);
FarPoint.Web.Spread.Chart.SpreadChart chart = new
FarPoint.Web.Spread.Chart.SpreadChart();
chart.Model = model;
fpSpread1.Sheets[0].Charts.Add(chart);
```

VB

```
Dim series As New FarPoint.Web.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.Web.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.Web.Chart.ChartModel()
model.PlotAreas.Add(plotArea)
Dim chart As New FarPoint.Web.Spread.Chart.SpreadChart()
chart.Model = model
FpSpread1.Sheets(0).Charts.Add(chart)
```

Using the Chart Designer

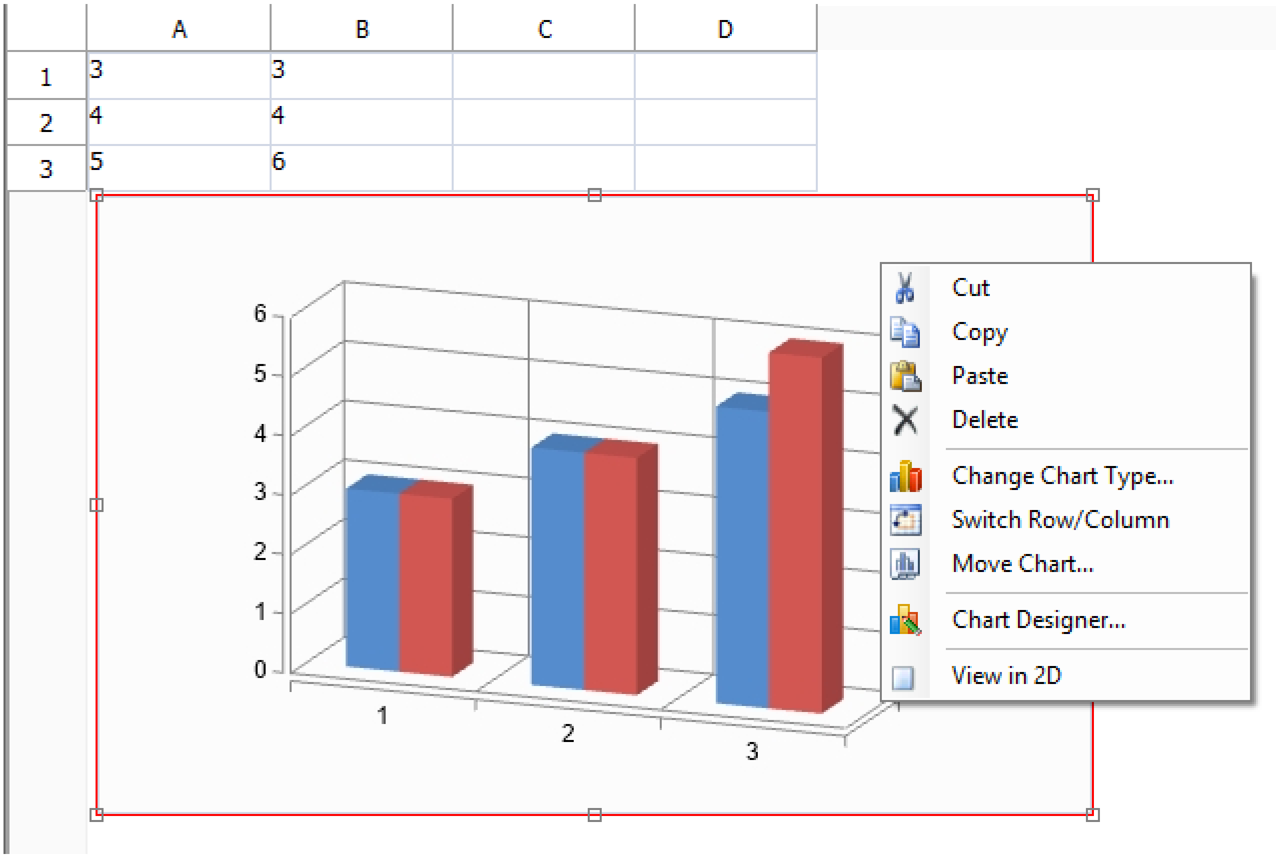
The Chart Designer's graphical interface saves time and effort and provides a visual representation of the Chart control as you change settings in the designer. You can apply the changes to the control.

See the following topics for more information about using the Chart Designer:

- **Opening the Chart Designer**
- **Creating a Chart Control**
- **Using the Chart Collection Editors**

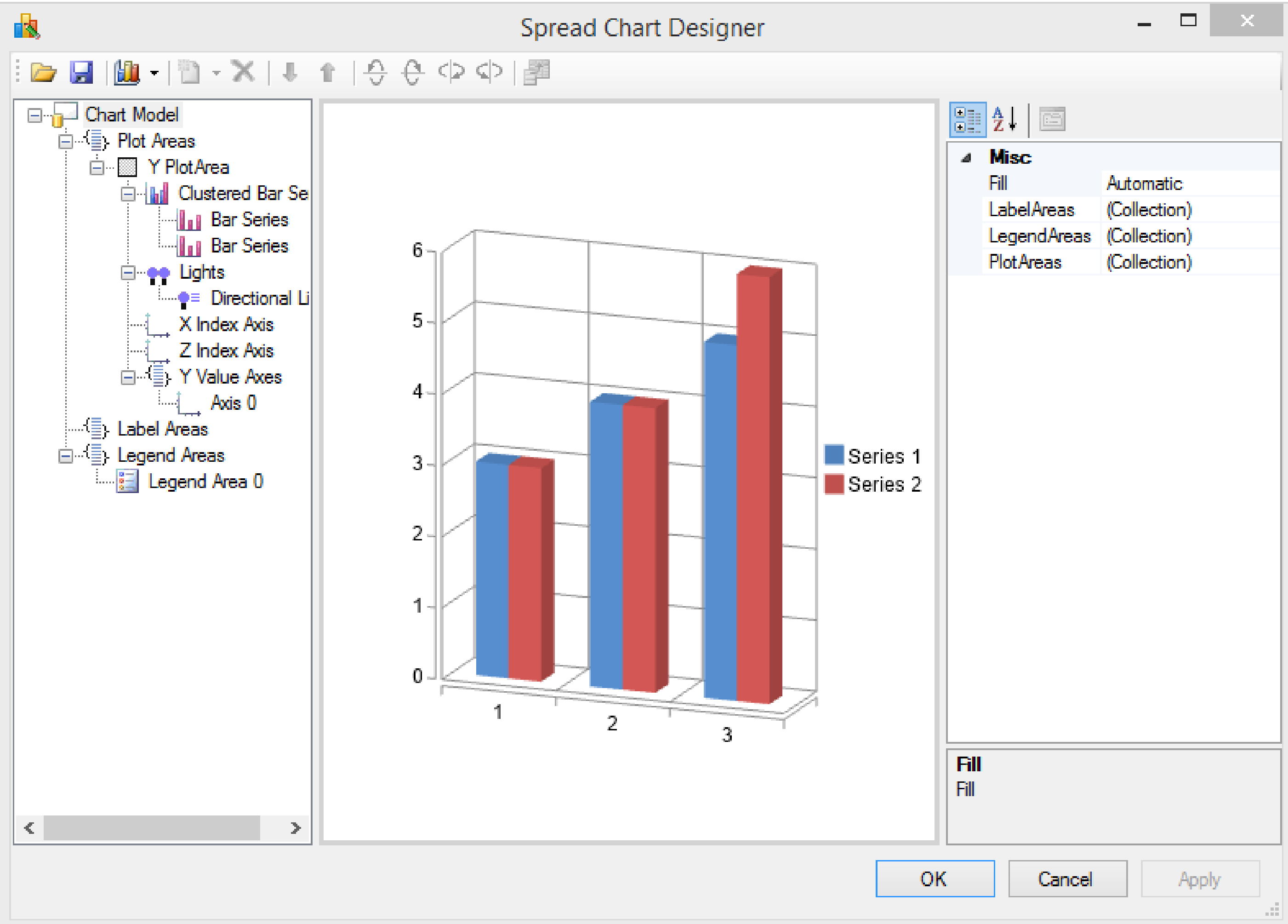
Opening the Chart Designer

You can open the designer by right-clicking on a chart and then selecting **Chart Designer** from the menu.

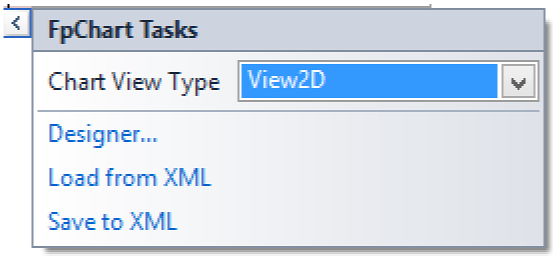


	A	B	C	D
1	3	3		
2	4	4		
3	5	6		

This opens the designer with the various editors that can be used to customize the Chart control.



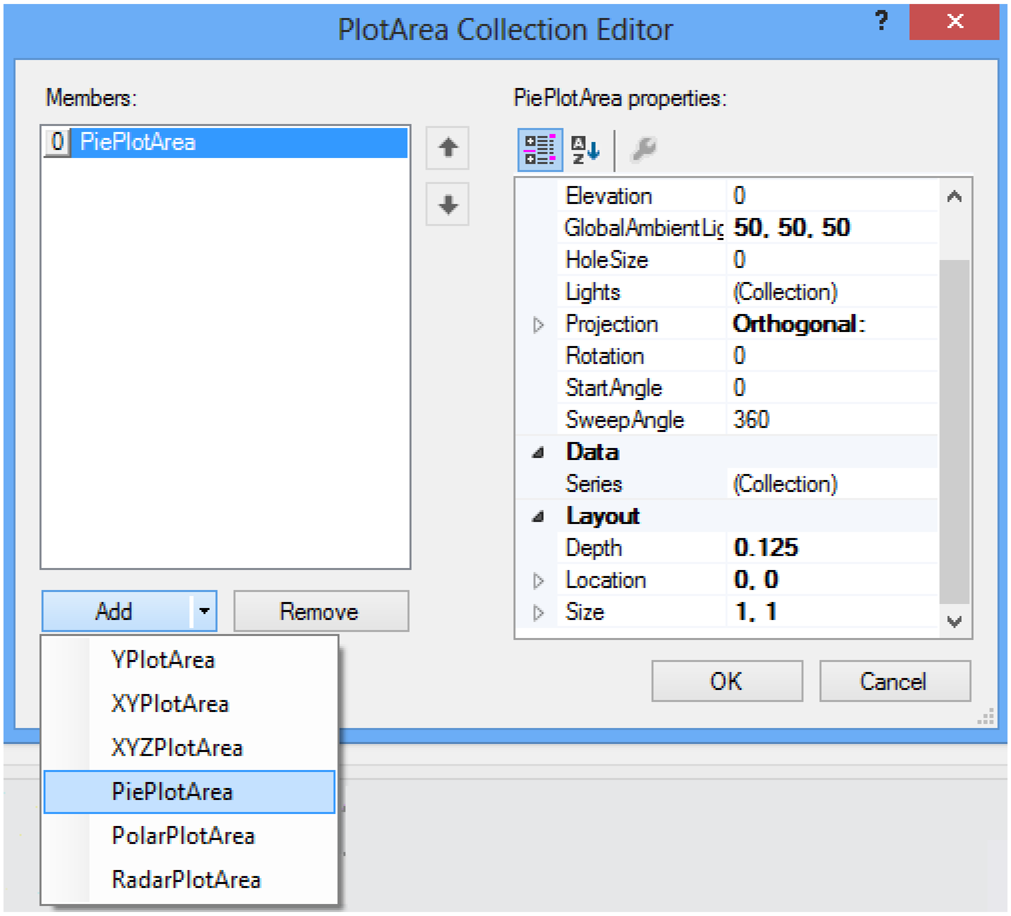
You can also edit the Chart control by adding a Chart control to the form and using the chart verb to bring up the designer.



Creating a Chart Control

You can create a Chart control with the Chart Designer. In the Spread designer, select the chart icon, add a chart, right-click on the chart, and select the Chart Designer (or select a Chart control on the form and select the designer from the chart verb).

1. Click on the **PlotArea Collection** drop-down button. If the **YPlotArea** is selected, click **Remove**. Then click **Add** and select the **PiePlotArea**.

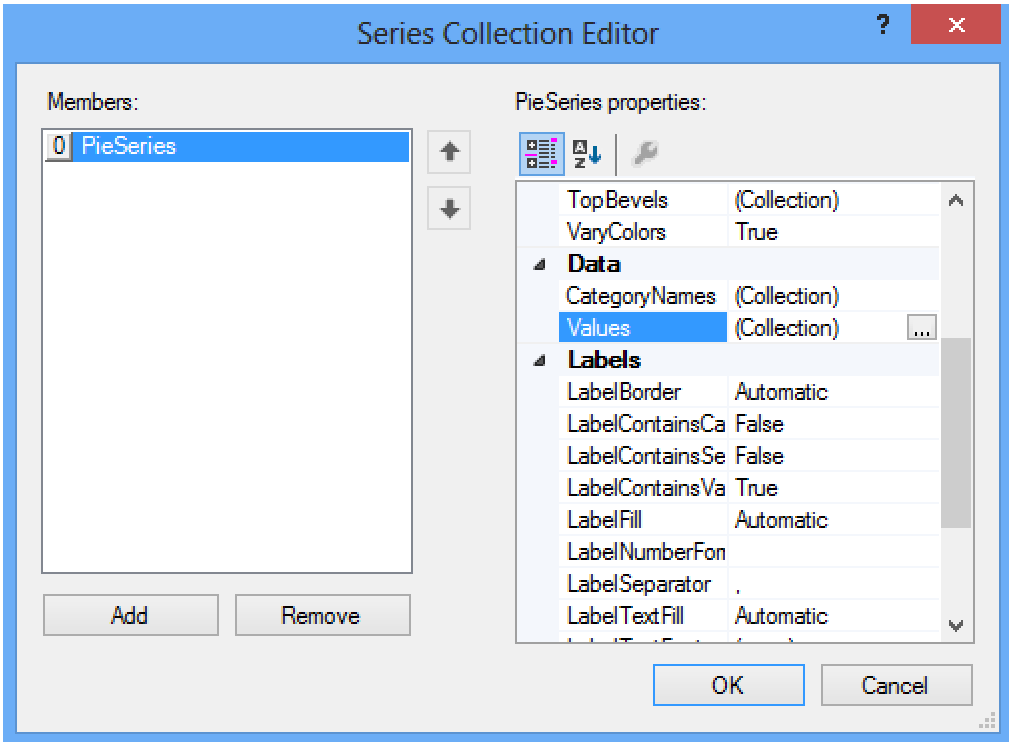


Members:

0 PiePlotArea

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

2. Select the **Series Collection** drop-down button (under **Data**).
3. Click the **Add** button in the **Series Collection Editor**. Then select the **Values Collection** drop-down button.

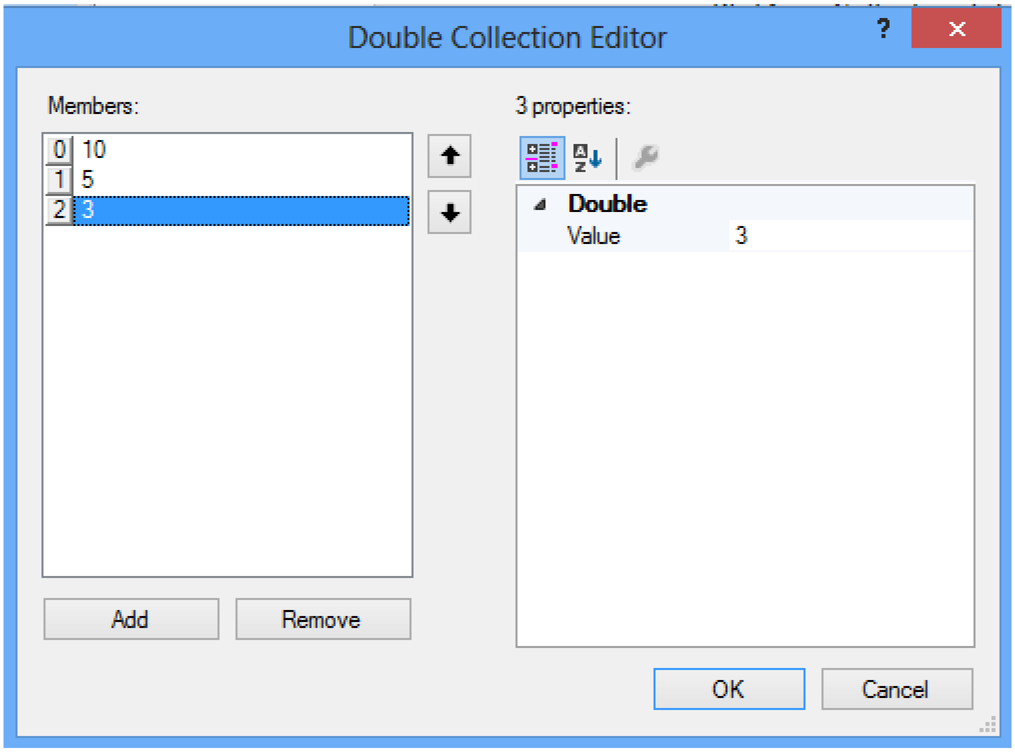


Members:

0 PieSeries

TopBevels	(Collection)
VaryColors	True
Data	
CategoryNames	(Collection)
Values	(Collection)
Labels	
LabelBorder	Automatic
LabelContainsCa	False
LabelContainsSe	False
LabelContainsVa	True
LabelFill	Automatic
LabelNumberFon	
LabelSeparator	,
LabelTextFill	Automatic

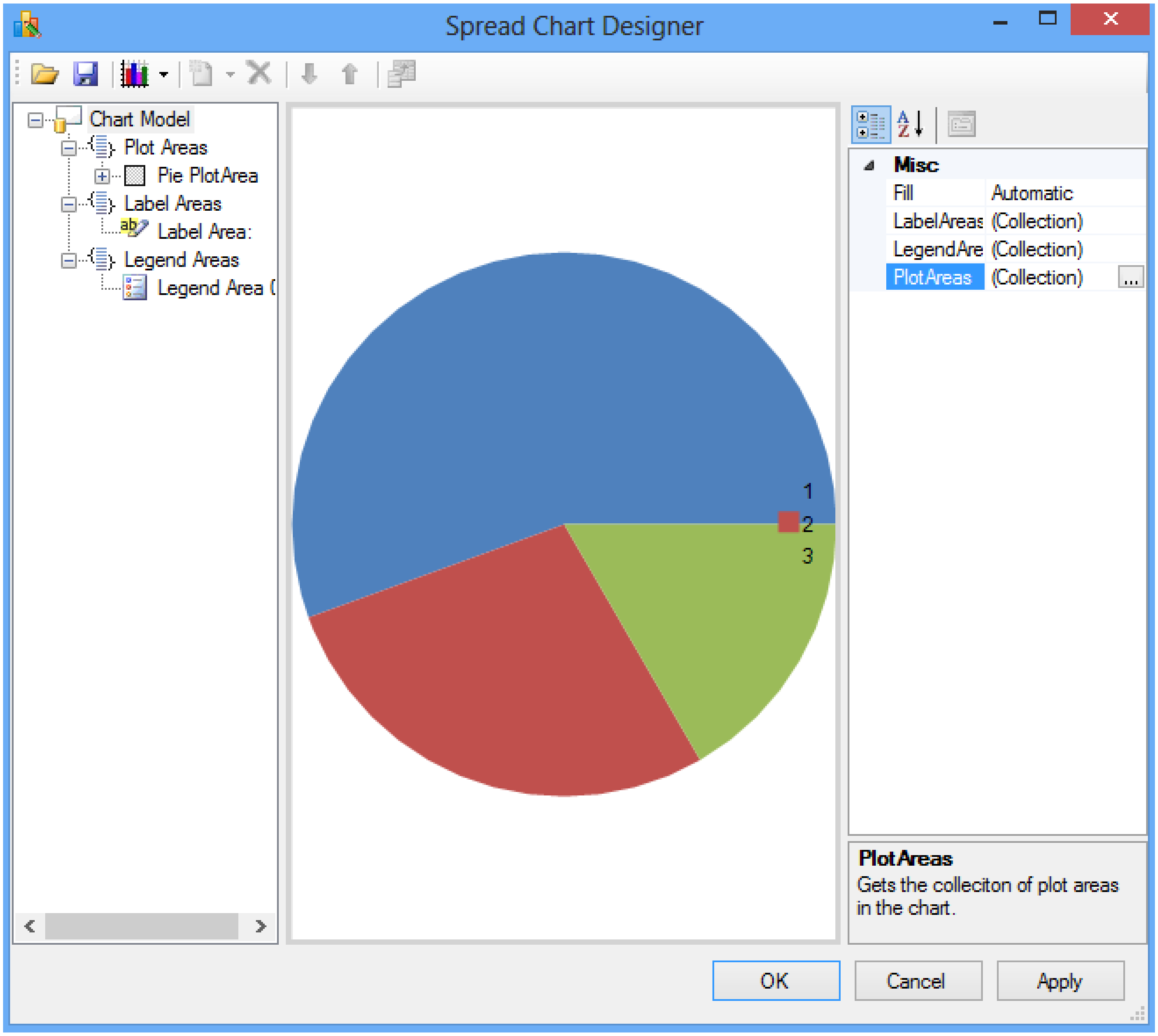
4. Click the **Add** button and add multiple data values. Type a double value in the text area on the right side of the editor.



0	10
1	5
2	3

Double
Value 3

5. 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).



Fill	Automatic
LabelAreas	(Collection)
LegendAre	(Collection)
PlotAreas	(Collection)

PlotAreas
Gets the colleciton of plot areas in the chart.

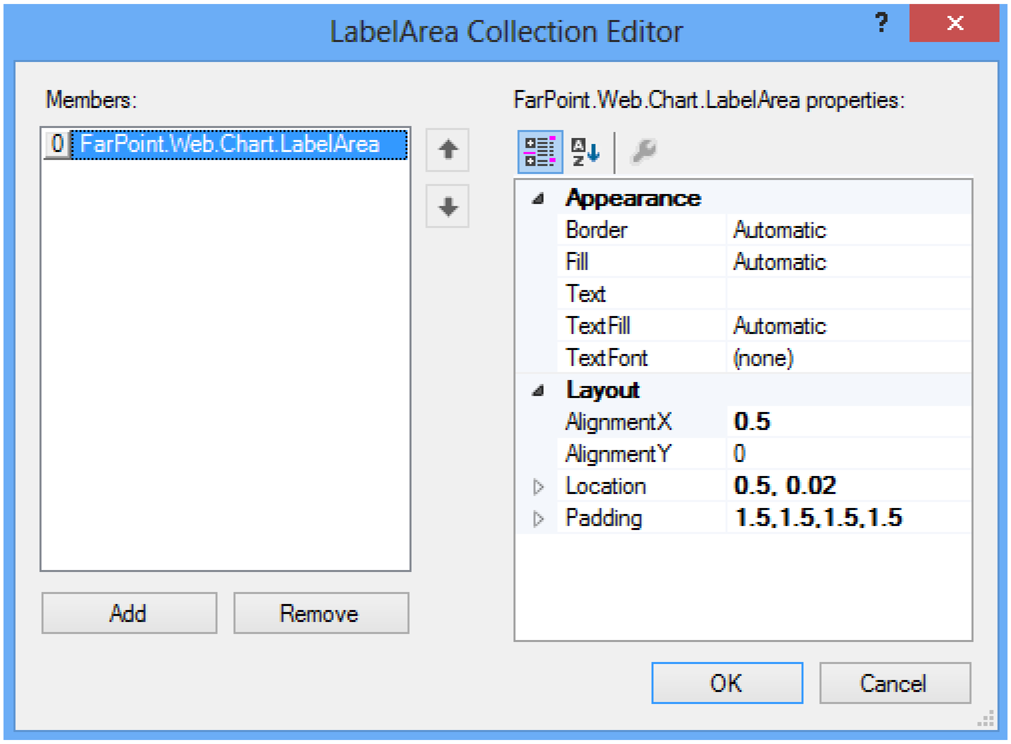
Using the Chart Collection Editors

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.

- **LabelArea Collection Editor**
- **LegendArea Collection Editor**
- **PlotArea Collection Editor**
- **Light Collection Editor**
- **Series Collection Editor**

LabelArea Collection Editor

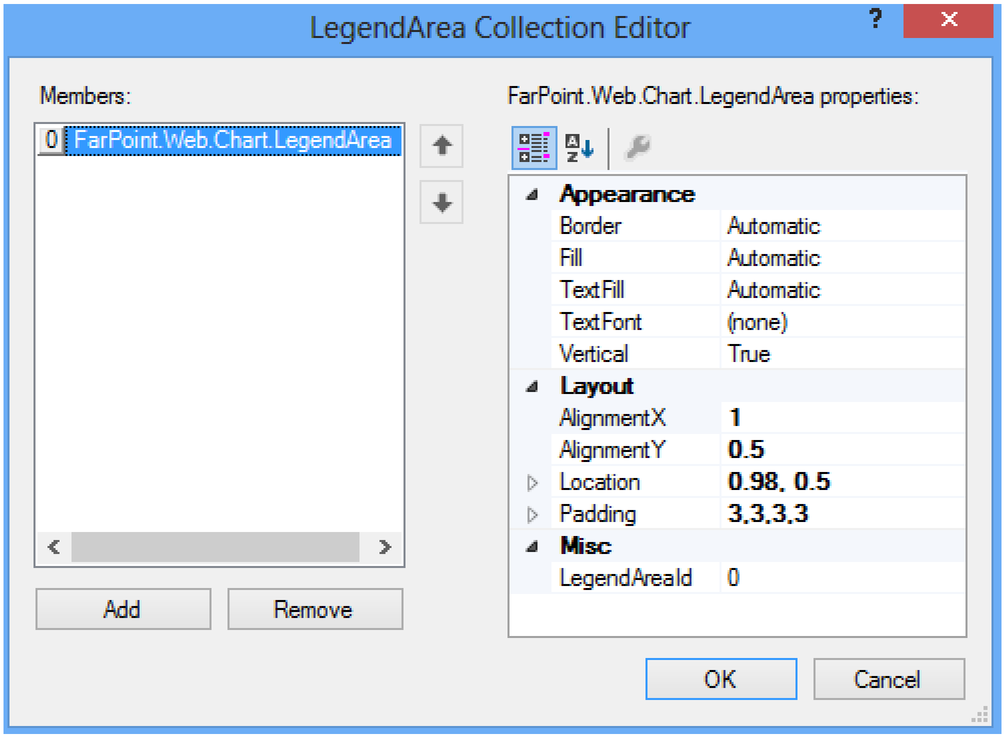
The **Label Collection Editor** can be used to create labels for the chart as shown in the following figure:


LabelArea Collection Editor
?
Members:
0 FarPoint.Web.Chart.LabelArea
Add
Remove
FarPoint.Web.Chart.LabelArea properties:
OK
Cancel

Appearance	
Border	Automatic
Fill	Automatic
Text	
TextFill	Automatic
TextFont	(none)
Layout	
AlignmentX	**0.5**
AlignmentY	0
Location	**0.5, 0.02**
Padding	**1.5,1.5,1.5,1.5**

LegendArea Collection Editor

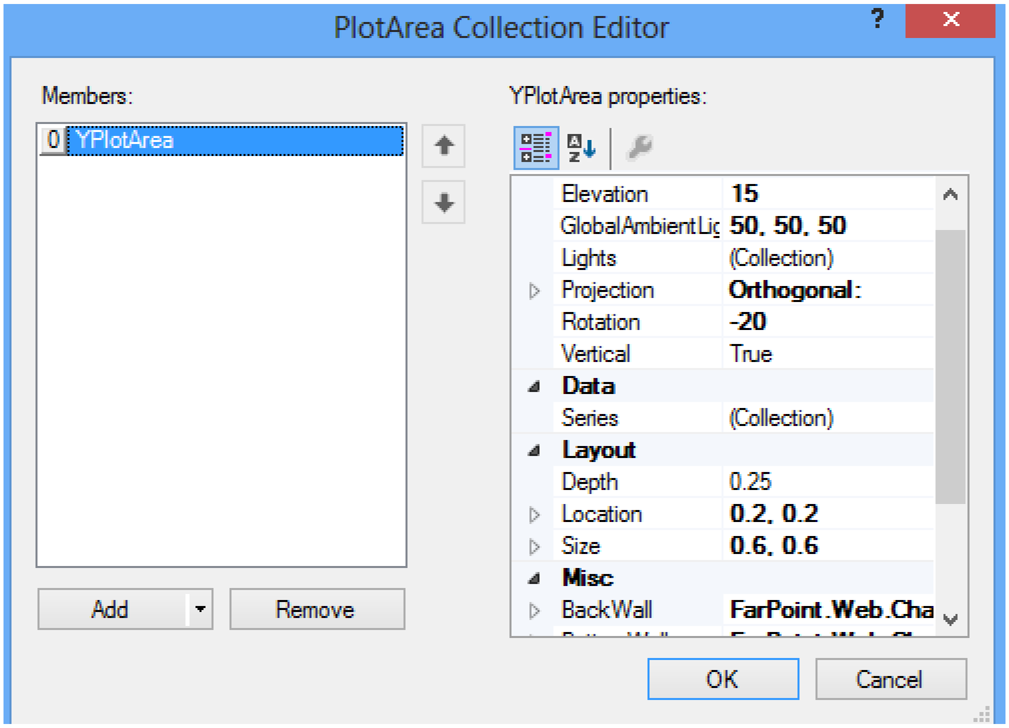
The **Legend Collection Editor** can be used to create legends for the chart as shown in the following figure:


LegendArea Collection Editor
?
Members:
0 FarPoint.Web.Chart.LegendArea
Add
Remove
FarPoint.Web.Chart.LegendArea properties:
OK
Cancel

Appearance	
Border	Automatic
Fill	Automatic
TextFill	Automatic
TextFont	(none)
Vertical	True
Layout	
AlignmentX	**1**
AlignmentY	**0.5**
Location	**0.98, 0.5**
Padding	**3,3,3,3**
Misc	
LegendAreaId	0

PlotArea Collection Editor

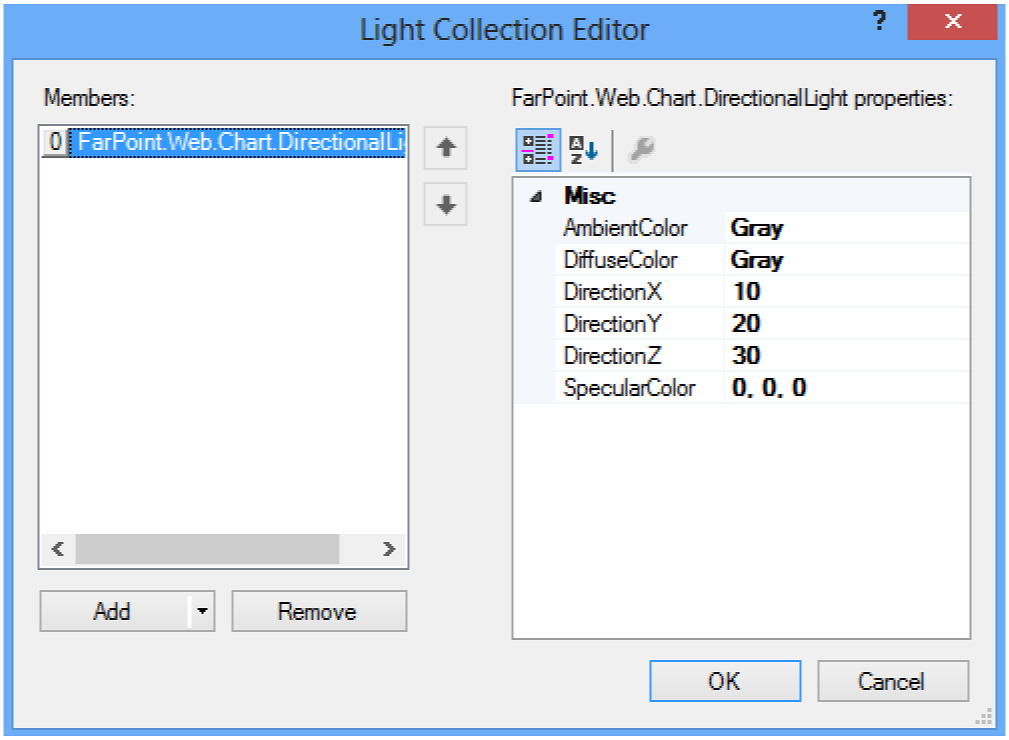
The **PlotArea Collection Editor** can be used to create plots for the chart as shown in the following figure:



Elevation 15
GlobalAmbientLig 50, 50, 50
Lights (Collection)
Projection Orthogonal:
Rotation -20
Vertical True
Data
Series (Collection)
Layout
Depth 0.25
Location 0.2, 0.2
Size 0.6, 0.6
Misc
BackWall FarPoint.Web.Cha

Light Collection Editor

The **Light Collection Editor** can be used to create lighting effects for the chart as shown in the following figure:

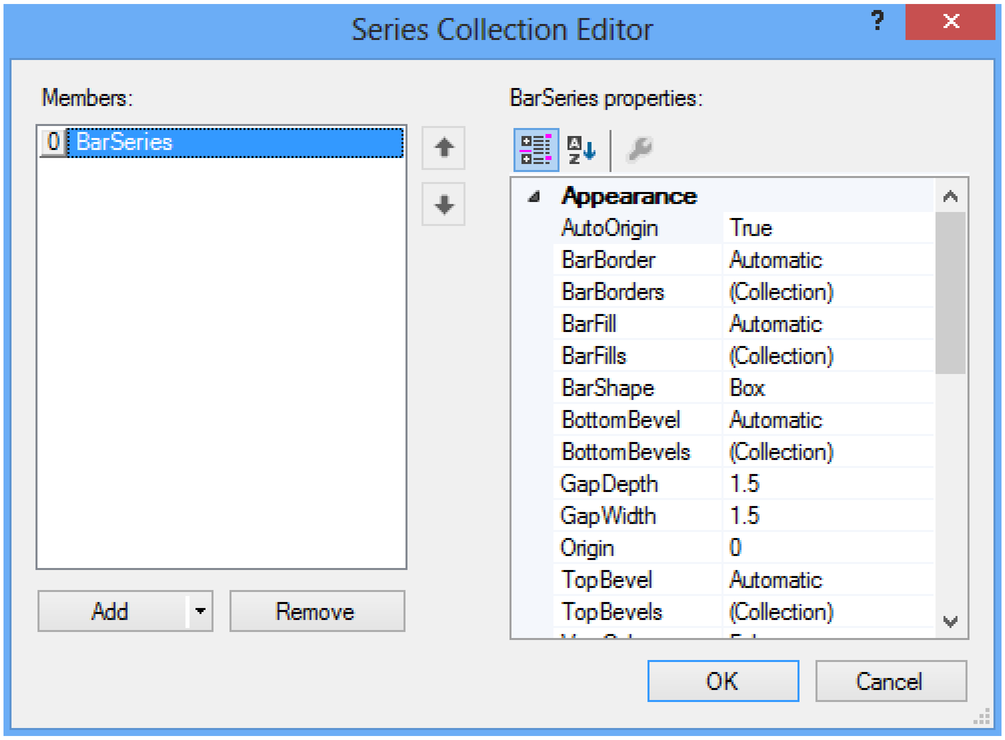

Light Collection Editor
Members:
0 FarPoint.Web.Chart.DirectionalLi
FarPoint.Web.Chart.DirectionalLight properties:
Add
Remove
OK
Cancel

Misc
AmbientColor Gray
DiffuseColor Gray
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 **Lights 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:

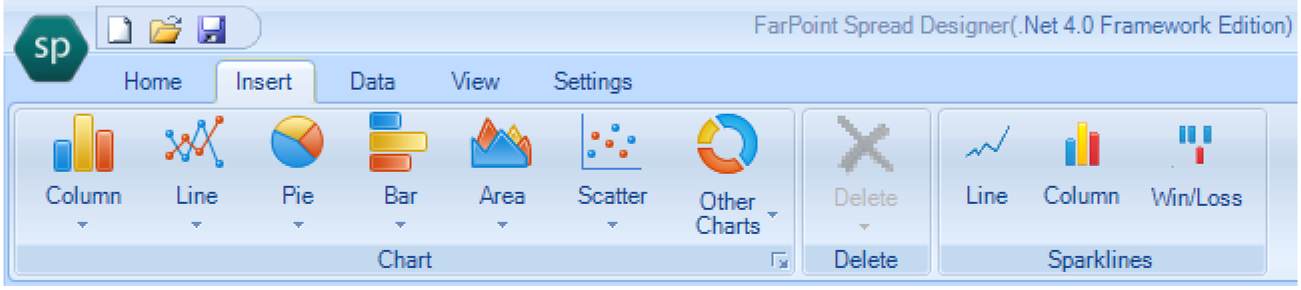


Appearance	
AutoOrigin	True
BarBorder	Automatic
BarBorders	(Collection)
BarFill	Automatic
BarFills	(Collection)
BarShape	Box
BottomBevel	Automatic
BottomBevels	(Collection)
GapDepth	1.5
GapWidth	1.5
Origin	0
TopBevel	Automatic
TopBevels	(Collection)

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.

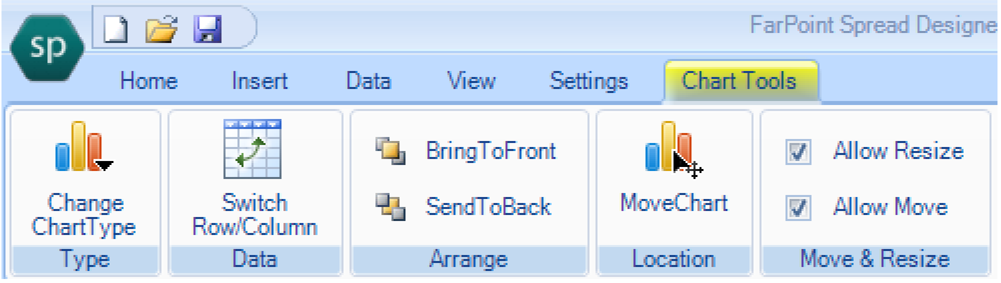
Using the Spread Designer

You can open the designer by right-clicking on the control and then selecting **Spread Designer** from the menu. Click on the **Insert** tab to bring up the **Chart** options. Click on one of the chart types to bring up the **Insert Chart** dialog.



You can use the **Insert Chart** dialog to add a chart. Select a chart type and then click **OK**. The chart is placed in the middle of the current sheet view and the **Chart Tools** dialog is displayed.

The **Chart Tools** dialog can be used to make changes to the chart (move, resize, change the chart type, and so on).



You can type data in the designer before adding a chart or you can add data after adding a chart. If you add the Chart control first, you can then right click on the Chart control and select the **Chart Designer** from the context menu to add data.

If you select a cell range and then add the chart, that data is used in creating the chart. If you select a cell, the control will auto-detect a cell range based on the selected cell. If you do not select a cell or cells(s), an empty chart is created.

For more information about the Chart context menu, see **Using the Chart Context Menu**.

Using the Chart Control

You can use the Chart control without having a Spread control on the form. The Chart control icon is added to the toolbox when installing the control. Select the icon and draw the Chart control on the form. You can then use the Chart Designer or code to add data and formatting to the Chart control.

- **Creating the Chart Control**
- **Rendering or Saving the Chart Control to an Image**
- **Loading or Saving the Chart Control to XML**

Creating the Chart Control

You can create a Chart control with code or with the Chart Designer. Select the Chart control, click on the chart verb, and then select **Designer** to bring up the Chart Designer.

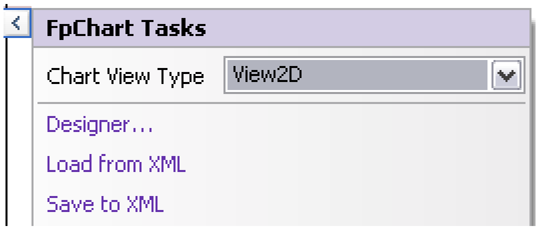


Chart View Type: View2D

You can also create a chart with code. The code is similar to using the Chart control in Spread; however, you do not need the **AddChart ('AddChart Method' in the on-line documentation)** method in this case.

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. Add the chart model to the Chart control.

Example

The following example creates a bar type chart with a label and legend.

C#

```
FarPoint.Web.Chart.BarSeries series = new FarPoint.Web.Chart.BarSeries();
series.SeriesName = "Series 0";
series.Values.Add(2.0);
series.Values.Add(4.0);
series.Values.Add(3.0);
series.Values.Add(5.0);
FarPoint.Web.Chart.YPlotArea plotArea = new FarPoint.Web.Chart.YPlotArea();
plotArea.Location = new System.Drawing.PointF(0.2f, 0.2f);
plotArea.Size = new System.Drawing.SizeF(0.6f, 0.6f);
plotArea.Series.Add(series);
FarPoint.Web.Chart.LabelArea label = new FarPoint.Web.Chart.LabelArea();
label.Text = "Bar Chart";
label.Location = new System.Drawing.PointF(0.5f, 0.02f);
label.AlignmentX = 0.5f;
label.AlignmentY = 0.0f;
FarPoint.Web.Chart.LegendArea legend = new FarPoint.Web.Chart.LegendArea();
legend.Location = new System.Drawing.PointF(0.98f, 0.5f);
legend.AlignmentX = 1.0f;
legend.AlignmentY = 0.5f;
FarPoint.Web.Chart.ChartModel model = new FarPoint.Web.Chart.ChartModel();
model.LabelAreas.Add(label);
model.LegendAreas.Add(legend);
model.PlotAreas.Add(plotArea);
FpChart1.Model = model;
```

VB

```
Dim series As New FarPoint.Web.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.Web.Chart.YPlotArea()
plotArea.Location = New System.Drawing.PointF(0.2f, 0.2f)
plotArea.Size = New System.Drawing.SizeF(0.6f, 0.6f)
plotArea.Series.Add(series)
Dim label As New FarPoint.Web.Chart.LabelArea()
label.Text = "Bar Chart"
label.Location = New System.Drawing.PointF(0.5f, 0.02f)
label.AlignmentX = 0.5f
label.AlignmentY = 0.0f
Dim legend As New FarPoint.Web.Chart.LegendArea()
legend.Location = New System.Drawing.PointF(0.98f, 0.5f)
legend.AlignmentX = 1.0f
legend.AlignmentY = 0.5f
Dim model As New FarPoint.Web.Chart.ChartModel()
model.LabelAreas.Add(label)
model.LegendAreas.Add(legend)
model.PlotAreas.Add(plotArea)
FpChart1.Model = model
```

Rendering or Saving the Chart Control to an Image

You can specify how to render the Chart control as an image. The Chart control has two built-in image render classes:

- FileImageRender
- HttpHandlerImageRender

The HttpHandlerImageRender class should be used when the charts are easy to generate or when there will be requests for many different charts. The images can be saved to the server using a session or cache. You can specify the storage type with the **ImageTransferStorage ('ImageTransferStorage Property' in the on-line documentation)** property.

The FileImageRender class renders the chart image to the client using temporary files. Since repeated requests for the same chart use the same file, this reduces the load on the server. New requests create new files as needed. Files are not cleaned up automatically. This class should be used when charts are complex and expensive to generate or when you expect a large number of requests for a small number of different charts. You can select the image type with the **ChartImageType ('ChartImageType Property' in the on-line documentation)** property.

Use the **ImageRender ('ImageRender Property' in the on-line documentation)** property to specify which image render class to use.

There are several server Chart events that can be used when working with a chart image:

- **MapAreaClick ('MapAreaClick Event' in the on-line documentation)**
- **BeforeRenderMapAreas ('BeforeRenderMapAreas Event' in the on-line documentation)**

The **MapAreaClick ('MapAreaClick Event' in the on-line documentation)** event can be used to determine which chart element the user clicked on. Each map area on the client side corresponds to a chart element (plot area, legend, series, and so on). The **BeforeRenderMapAreas ('BeforeRenderMapAreas Event' in the on-line documentation)** event can be used to customize client behavior of the map area.

The **RenderMapArea ('RenderMapArea Property' in the on-line documentation)** property should be set to true to use the **MapAreaClick ('MapAreaClick Event' in the on-line documentation)** event. The **HitTest ('HitTest Method' in the on-line documentation)** method can be used to get information about the chart element that is clicked.

The **EnableClickEvent ('EnableClickEvent Property' in the on-line documentation)** property specifies whether the **Click ('Click Event' in the on-line documentation)** event fires.

The **MapAreaClick ('MapAreaClick Event' in the on-line documentation)** event fires if the **RenderMapArea ('RenderMapArea Property' in the on-line documentation)** property is true and the **EnableClickEvent ('EnableClickEvent Property' in the on-line documentation)** property is false. If the **RenderMapArea ('RenderMapArea Property' in the on-line documentation)** property is false and the **EnableClickEvent ('EnableClickEvent Property' in the on-line documentation)** property is true, only the **Click ('Click Event' in the on-line documentation)** event fires. If the **RenderMapArea ('RenderMapArea Property' in the on-line documentation)** property is true and the **EnableClickEvent ('EnableClickEvent Property' in the on-line documentation)** property is true, the **MapAreaClick ('MapAreaClick Event' in the on-line documentation)** event fires if the user clicks on a chart element. The **Click ('Click Event' in the on-line documentation)** event fires if the user clicks on the chart background.

The **HotSpotMode ('HotSpotMode Property' in the on-line documentation)** property is used if **RenderMapArea ('RenderMapArea Property' in the on-line documentation)** is true. **HotSpotMode ('HotSpotMode Property' in the on-line documentation)** determines the behavior when the user clicks in a map area. The options are inactive, navigate, and post back. The inactive option is useful if you wish to display a tooltip in the map area. If the **HotSpotMode ('HotSpotMode Property' in the on-line documentation)** is set to post back, the **MapAreaClick ('MapAreaClick Event' in the on-line documentation)** and Client events are fired. If the **RenderMapArea ('RenderMapArea Property' in the on-line documentation)** property is false, the **HotSpotMode ('HotSpotMode Property' in the on-line documentation)** property has no effect.

Loading or Saving the Chart Control to XML

You can load or save the Chart control to or from an XML file. This option is available at design time if you select the Chart control on the form and select the chart verb. Select the **Save to XML** or **Load from XML** menu option. Use the **LoadFromTemplate ('LoadFromTemplate Method' in the on-line documentation)** or **SaveToTemplate ('SaveToTemplate Method' in the on-line documentation)** methods at run time.

Using Code

Save or load a Chart control to an XML file.

Example

The following example saves or loads a Chart control.

C#

```
string f;
f = "c:\\chart.xml";
System.IO.FileStream s = new System.IO.FileStream(f, IO.FileMode.OpenorCreate,
IO.FileAccess.ReadWrite);
FpChart1.SaveToTemplate(s);
//FpChart1.LoadFromTemplate(s);
```

VB

```
Dim f As String
f = "c:\chart.xml"
Dim s As New System.IO.FileStream(f, IO.FileMode.OpenOrCreate, IO.FileAccess.ReadWrite)
FpChart1.SaveToTemplate(s)
'FpChart1.LoadFromTemplate(s)
```

Using the Chart Control in Spread

You can scroll, select, resize, or move the chart. You can also create borders and set the type of view (2D or 3D). You can create a chart with code using Spread classes or a combination of Chart and Spread classes.

See the following topics for more information:

- **Creating the Chart Control with Code**
- **Binding the Chart Control with Spread**
- **Moving and Resizing the Chart Control in Spread**
- **Selecting the Chart Control in Spread**
- **Setting the Chart Control Border in Spread**
- **Setting the Chart View Type**
- **Using the Chart Context Menu**

Creating the Chart Control with Code

You can create a chart with Spread methods or you can create a chart by using Chart classes and then adding the chart to Spread.

There is a Spread **AddChart ('AddChart Method' in the on-line documentation)** method and a **SpreadChart ('SpreadChart Constructor' in the on-line documentation)** constructor that can be used to create a Chart control. The **AddChart ('AddChart Method' in the on-line documentation)** method and the **SpreadChart** constructor have overloads that allow you to specify the cell range to get the data from, the type of series, height and width of the chart, location of the chart, the view type of the chart, and whether to show a legend.

You can control how empty cell data is displayed in the Chart control with the **DataSetting ('DataSetting Property' in the on-line documentation)** property.

For more information on using Chart classes, see the **Creating Plot Types** topic.

Using Code

Add values to cells and then use the **AddChart ('AddChart Method' in the on-line documentation)** method to add a Chart control to Spread.

Example

The following example uses the **AddChart ('AddChart Method' in the on-line documentation)** method.

C#

```
FpSpread1.Sheets[0].RowCount = 10;
FpSpread1.Sheets[0].ColumnCount = 10;
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[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[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[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;
FarPoint.Web.Spread.Model.CellRange range = new FarPoint.Web.Spread.Model.CellRange(0,
0, 4, 4);
FpSpread1.Sheets[0].AddChart(range, typeof(FarPoint.Web.Chart.ClusteredBarSeries), 200,
200, 0, 0, FarPoint.Web.Chart.ChartViewType.View3D, false);
```

VB

```
FpSpread1.Sheets(0).RowCount = 10
FpSpread1.Sheets(0).ColumnCount = 10
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(0, 1).Value = "c1"
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(0, 2).Value = "c2"
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(0, 3).Value = "c3"
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
Dim range As New FarPoint.Web.Spread.Model.CellRange(0, 0, 4, 4)
FpSpread1.Sheets(0).AddChart(range, GetType(FarPoint.Web.Chart.ClusteredBarSeries),
200, 200, 0, 0, FarPoint.Web.Chart.ChartViewType.View3D, False)
```

Using Code

You can also use chart classes to create a chart and then add the chart to the Spread control.

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);
FarPoint.Web.Spread.Chart.SpreadChart chart = new
FarPoint.Web.Spread.Chart.SpreadChart();
chart.Model = model;
FpSpread1.Sheets[0].Charts.Add(chart);
```

VB

```
Dim series As New FarPoint.Web.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.Web.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.Web.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.Web.Chart.LegendArea()
legend.Location = New PointF(0.98F, 0.5F)
legend.AlignmentX = 1.0F
legend.AlignmentY = 0.5F
Dim model As New FarPoint.Web.Chart.ChartModel()
model.LabelAreas.Add(label)
model.LegendAreas.Add(legend)
model.PlotAreas.Add(plotArea)
Dim chart As New FarPoint.Web.Spread.Chart.SpreadChart()
chart.Model = model
FpSpread1.Sheets(0).Charts.Add(chart)
```

Binding the Chart Control with Spread

You can bind the Chart control to data in the Spread control with the formula parameters in the **SpreadChart ('SpreadChart Constructor' in the on-line documentation)** constructor. The constructor uses formula syntax to specify the cell range.

You can also bind the Chart control in Spread with the **CategoryFormula ('CategoryFormula Property' in the on-line documentation)**, **DataFormula ('DataFormula Property' in the on-line documentation)**, **Formula ('Formula Property' in the on-line documentation)**, or **SeriesNameFormula ('SeriesNameFormula Property' in the on-line documentation)** properties.

Using Code

Add values to the cells and then use the formula parameters in the **SpreadChart ('SpreadChart Constructor' in the on-line documentation)** constructor to create the Chart control.

Example

The following example uses the **SpreadChart ('SpreadChart Constructor' in the on-line documentation)** constructor to create a Chart control.

C#

```
FpSpread1.Sheets[0].RowCount = 10;
FpSpread1.Sheets[0].ColumnCount = 10;
FpSpread1.Sheets[0].Cells[0, 0].Value = 3;
FpSpread1.Sheets[0].Cells[1, 1].Value = 7;
FpSpread1.Sheets[0].Cells[2, 2].Value = 7;
FpSpread1.Sheets[0].Cells[3, 3].Value = 5;
FarPoint.Web.Spread.Chart.SpreadChart chart = new
FarPoint.Web.Spread.Chart.SpreadChart("Sheet1!$A$2:$A$7", "Sheet1!$B$1:$D$1",
"Sheet1!$B$2:$D$7", typeof(FarPoint.Web.Chart.BarSeries));
```

```
FpSpread1.Sheets[0].Charts.Add(chart);
```

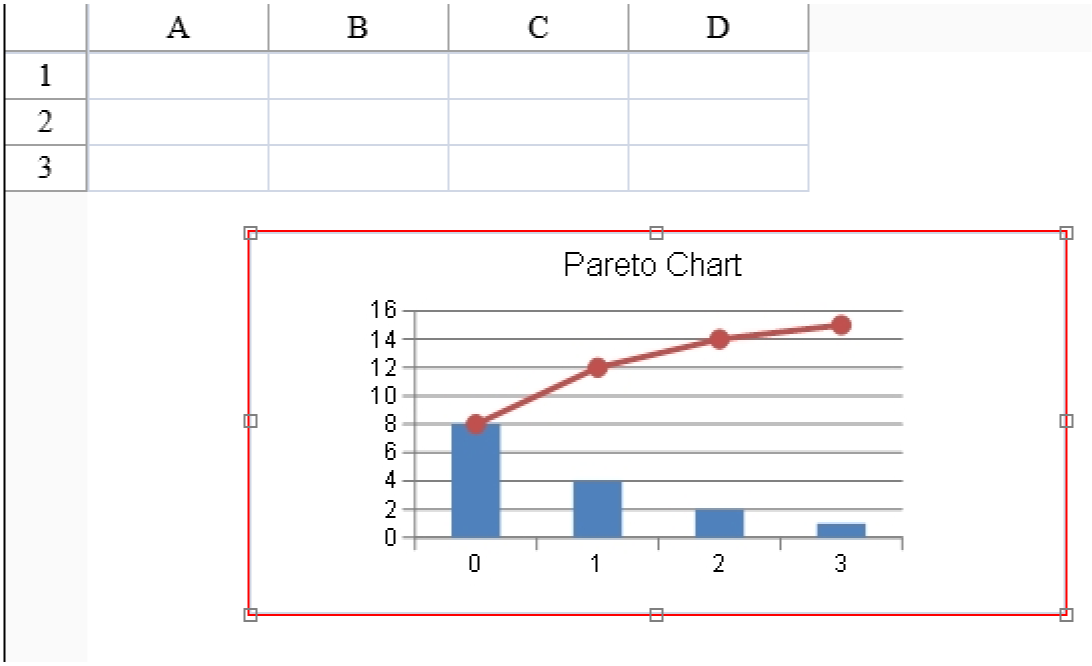
VB

```
FpSpread1.Sheets(0).RowCount = 10
FpSpread1.Sheets(0).ColumnCount = 10
FpSpread1.Sheets(0).Cells(0, 0).Value = 3
FpSpread1.Sheets(0).Cells(1, 1).Value = 7
FpSpread1.Sheets(0).Cells(2, 2).Value = 7
FpSpread1.Sheets(0).Cells(3, 3).Value = 5
Dim chart As New FarPoint.Web.Spread.Chart.SpreadChart("Sheet1!$A$2:$A$7",
"Sheet1!$B$1:$D$1", "Sheet1!$B$2:$D$7", GetType(FarPoint.Web.Chart.BarSeries))
FpSpread1.Sheets(0).Charts.Add(chart)
```

Moving and Resizing the Chart Control in Spread

You can move and resize the Chart control at design time in the Spread Designer or at run time.

At design time or run time you can move the chart by clicking on it and then dragging the chart on the Spread control with the mouse or you can click on the chart and use the keyboard direction keys. You can resize the chart by selecting it, moving the mouse pointer over one of the indicators and waiting until the mouse pointer changes to a left-right icon or an up-down icon, and then dragging. The following image shows the red outline and indicator symbols.



At run time you can allow the user to move or resize the Chart control by setting the **CanSize ('CanSize Property' in the on-line documentation)** and **CanMove ('CanMove Property' in the on-line documentation)** properties. The **CanSelect ('CanSelect Property' in the on-line documentation)** property must be true to allow moving and resizing.

Using Code

Set the **CanSize ('CanSize Property' in the on-line documentation)** and **CanMove ('CanMove Property' in the on-line documentation)** properties in the **SpreadChart ('SpreadChart Class' in the on-line documentation)** class.

Example

The following example sets the **CanSize ('CanSize Property' in the on-line documentation)** and **CanMove ('CanMove Property' in the on-line documentation)** properties.

C#

```
FarPoint.Web.Spread.Chart.SpreadChart chart = new
FarPoint.Web.Spread.Chart.SpreadChart;
chart.Model = model;
chart.CanSize = true;
chart.CanMove = false;
fpSpread1.Sheets[0].Charts.Add(chart);
```

VB

```
Dim chart As New FarPoint.Web.Spread.Chart.SpreadChart()
chart.Model = model
chart.CanSize = True
chart.CanMove = False
FpSpread1.Sheets(0).Charts.Add(chart)
```

Using the Spread Designer

1. Select the **Chart Tools** menu.
2. Check the **AllowResize** or **AllowMove** option (or both).
3. Click **Apply and Exit** to close the Spread Designer.

Selecting the Chart Control in Spread

You can select the Chart control at design time in the Spread Designer or at run time.

At design time or run time you can select the chart by clicking on it with the mouse.

At run time the user cannot move or resize the chart if the **CanSelect ('CanSelect Property' in the on-line documentation)** property has been set to false. The **ActiveChart ('ActiveChart Property' in the on-line documentation)** property can be used to programmatically select a chart.

You can specify the order in which the charts are displayed with the **ZIndex ('ZIndex Property' in the on-line documentation)** property.

The default selected border is a red, solid border with a width of one pixel. You can change this by setting the **SelectedCssClass ('SelectedCssClass Property' in the on-line documentation)** property.

Using Code

Set the **CanSelect ('CanSelect Property' in the on-line documentation)** property in the **SpeadChart ('SpreadChart Class' in the on-line documentation)** class.

Example

The following example sets the **CanSelect ('CanSelect Property' in the on-line documentation)** property.

C#

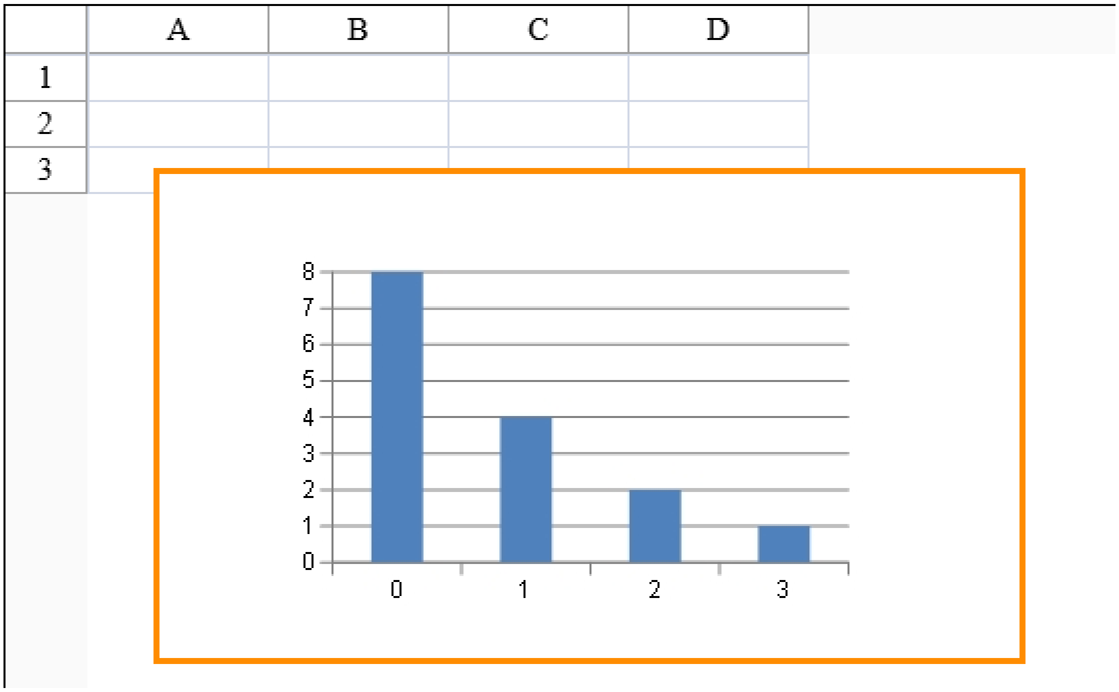
```
FarPoint.Web.Spread.Chart.SpreadChart chart = new
FarPoint.Web.Spread.Chart.SpreadChart;
chart.Model = model;
chart.CanSelect = true;
fpSpread1.Sheets[0].Charts.Add(chart);
```

VB

```
Dim chart As New FarPoint.Web.Spread.Chart.SpreadChart()
chart.Model = model
chart.CanSelect = True
FpSpread1.Sheets(0).Charts.Add(chart)
```

Setting the Chart Control Border in Spread

You can specify the color, type, and size of the border that goes around the edge of the Chart control.



Using Code

1. To set the border color, set **BorderColor ('BorderColor Property' in the on-line documentation)** in the **SpeadChart ('SpreadChart Class' in the on-line documentation)** class.
2. To set the border style, set **BorderStyle ('BorderStyle Property' in the on-line documentation)** in the **SpeadChart ('SpreadChart Class' in the on-line documentation)** class.
3. To set the border width, set **BorderWidth ('BorderWidth Property' in the on-line documentation)** in the **SpeadChart ('SpreadChart Class' in the on-line documentation)** class.

Example

The following example creates an orange border around the Chart control.

C#

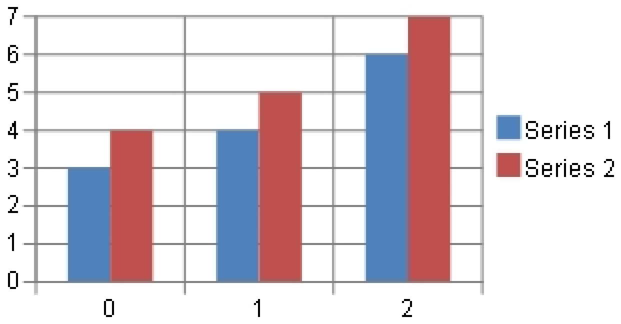
```
FarPoint.Web.Spread.Chart.SpreadChart chart = new
FarPoint.Web.Spread.Chart.SpreadChart;
chart.BorderColor = Drawing.Color.DarkOrange;
chart.BorderStyle = BorderStyle.Solid;
chart.BorderWidth = 3;
fpSpread1.Sheets[0].Charts.Add(chart);
```

VB

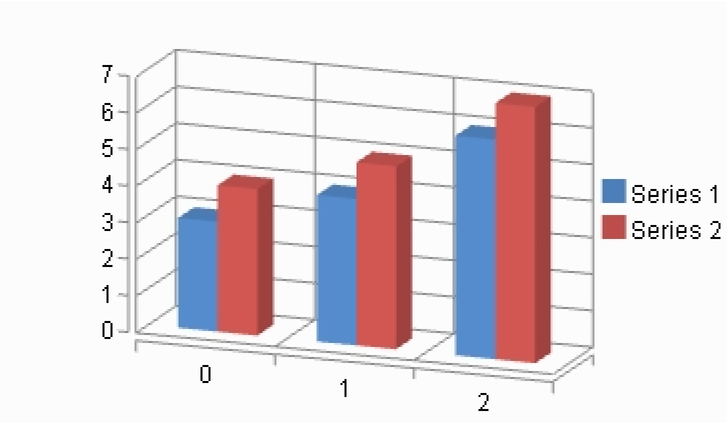
```
Dim chart As New FarPoint.Web.Spread.Chart.SpreadChart()
chart.BorderColor = Drawing.Color.DarkOrange
chart.BorderStyle = BorderStyle.Solid
chart.chart.BorderWidth = 3
FpSpread1.Sheets(0).Charts.Add(chart)
```

Setting the Chart View Type

You can specify a 2D or 3D view of the Chart control. You can set this with the *ChartViewType* parameter in the **SpreadChart ('SpreadChart Constructor' in the on-line documentation)** constructor when creating a chart. The following image shows a 2D chart.



The following image shows a 3D chart.



Using Code

Set the **ViewType ('ViewType Property' in the on-line documentation)** property.

Example

The following example sets the **ViewType ('ViewType Property' in the on-line documentation)** property.

C#

```
FarPoint.Web.Spread.Chart.SpreadChart chart = new
FarPoint.Web.Spread.Chart.SpreadChart;
chart.ViewType = FarPoint.Web.Chart.ChartViewType.View2D;
fpSpread1.Sheets[0].Charts.Add(chart);
```

VB

```
Dim chart As New FarPoint.Web.Spread.Chart.SpreadChart()
chart.ViewType = FarPoint.Web.Chart.ChartViewType.View2D
FpSpread1.Sheets(0).Charts.Add(chart)
```

Context Menu

You can set the view type to 2D or 3D by using the chart context menu.

1. Right-click the Chart control on the form at design time.
2. Select **ViewType** in the context menu and set the type.

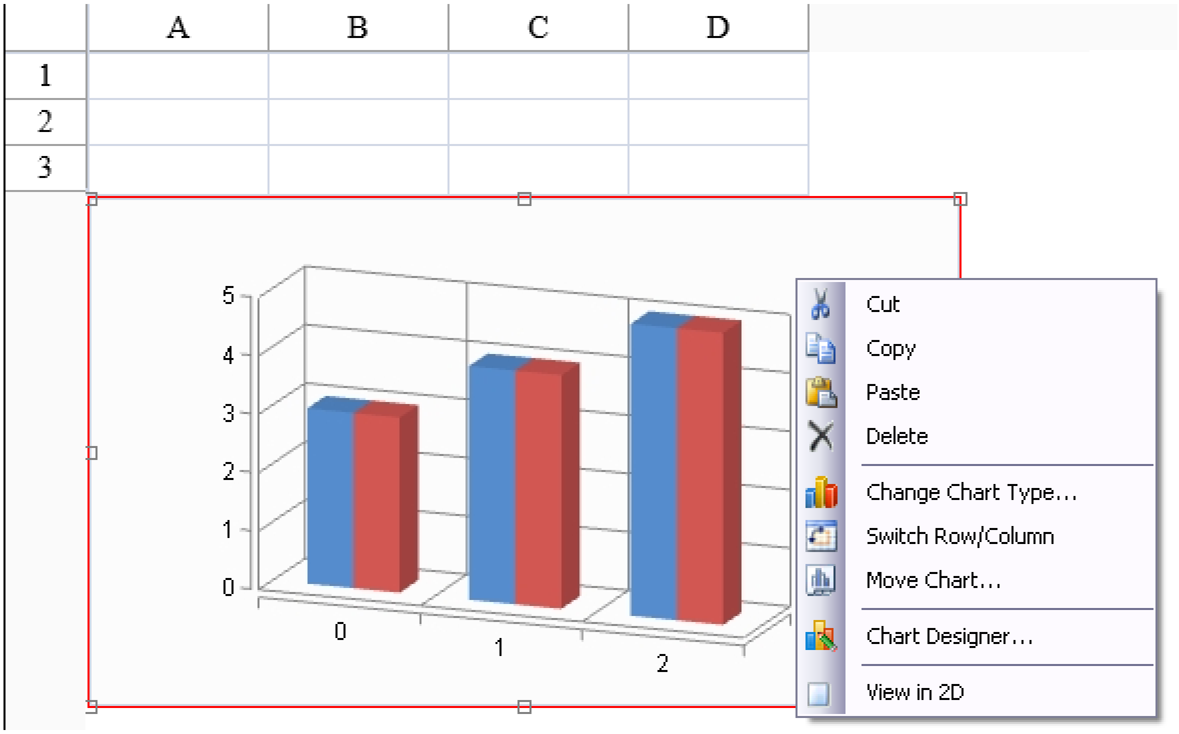
Using the Spread Designer

You can set the view type to 2D or 3D using the properties window in the designer.

1. Select the Chart control in the designer.
2. The **ViewType** setting is under the **Behavior** category in the **Properties** window on the right side of the designer (selected item section).
3. Click **Apply and Exit** to close the Spread Designer.

Using the Chart Context Menu

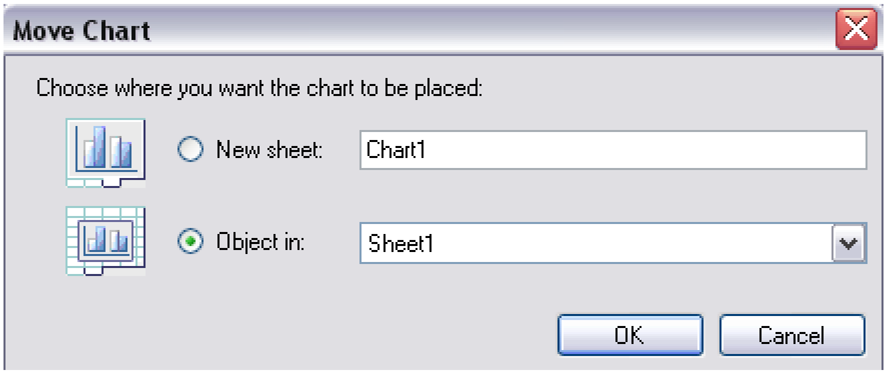
Select and right-click on the Chart control at design time to bring up the chart context menu. This menu gives you the standard cut, copy, paste, and delete options as well as options to change the chart.



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 **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 **Move Chart** dialog appears as follows:


Move Chart

Choose where you want the chart to be placed:

○ New sheet: Chart1

● Object in: Sheet1

The **Chart Designer** option allows you to bring up the Chart Designer.

The **View** option allows you to change the view to 2D or 3D.

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). Touch is also supported with Apple Safari on the iPad.

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.

The following topics provide additional information:

- **Understanding Touch Gestures**

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.

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:

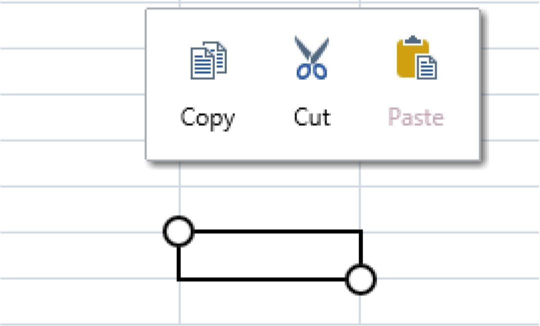
- **Using the Touch Menu Bar**
- **Using Touch Support with AutoFit**
- **Using Touch Support with Charts**
- **Using Touch Support with Editable Cells**
- **Using Touch Support with Filtering**
- **Using Touch Support with Grouping**
- **Using Touch Support when Moving Columns**
- **Using Touch Support when Moving Rows (on-line documentation)**

- **Using Touch Support when Resizing Columns or Rows**
- **Using Touch Support with Scrolling**
- **Using Touch Support with Selections**
- **Using Touch Support with Sorting**

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. The following image displays a default touch menu.



The touch menu bar provides additional options when selecting a column or row (insert, delete, hide, or unhide).

You can add custom menu items to the touch strip.

You can prevent the touch strip from being displayed on the server side by setting the **TouchStrips ('TouchStrips Property' in the on-line documentation)** property to None.

There are several client-side events and classes that can be used with the touch strip: **TouchStripOpening (on-line documentation)**, **TouchStripOpened (on-line documentation)**, **TouchStripClosed (on-line documentation)**, **MenuItem (on-line documentation)**, **TouchStrip (on-line documentation)**, and **TouchStripItem (on-line documentation)**.

Using Code

1. Create a touch strip.
2. Create touch strip items.
3. Add the menu items to the touch strip.
4. Specify when to display the touch strip.

Example

This example adds custom items to the touch strip.

C#

```
FarPoint.Web.Spread.TouchStrip touchStrip = new FarPoint.Web.Spread.TouchStrip();
FarPoint.Web.Spread.TouchStripItem parent = new
FarPoint.Web.Spread.TouchStripItem("Other....");
parent.ChildItems.Add(new FarPoint.Web.Spread.MenuItem("Child item"));
touchStrip.Items.Add(parent);
FpSpread1.TouchStrips[FarPoint.Web.Spread.TouchStripShowingArea.Cell] = touchStrip;
```

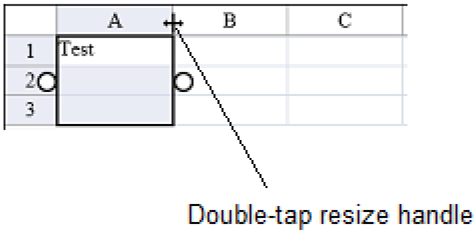
VB

```
Dim touchStrip As New FarPoint.Web.Spread.TouchStrip()
Dim parent As New FarPoint.Web.Spread.TouchStripItem("Other....")
parent.ChildItems.Add(New FarPoint.Web.Spread.MenuItem("Child item"))
touchStrip.Items.Add(parent)
FpSpread1.TouchStrips(FarPoint.Web.Spread.TouchStripShowingArea.Cell) = touchStrip
```

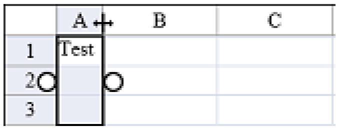
Using Touch Support with AutoFit

You can use touch support gestures with automatic fit.

Tap to select a column (resize handle becomes visible). Double-tap to resize the column automatically. The **Resizable ('Resizable Property' in the on-line documentation)** property must be true for the column.



The following image displays the resized column.



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 chart if **CanSelect ('CanSelect Property' in the on-line documentation)** is true.
Hold and Release	Right-click	Selects a chart and displays the touch strip if **CanSelect ('CanSelect Property' in the on-line documentation)** is set to true.
Press resize handles, then slide	Press left button on edge then move	Resizes a chart if **CanSize ('CanSize Property' in the on-line documentation)** is set to true.
Press chart then slide	Press left button on chart then move	Moves a chart if **CanMove ('CanMove Property' in the on-line documentation)** is set to true.

Using Touch Support with Editable Cells

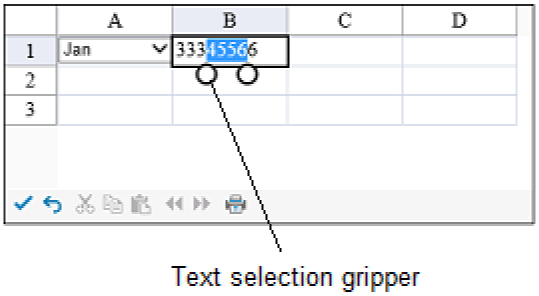
You can use touch gestures to edit cells that allow editing.

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.

Common touch gestures and the mouse action equivalent are listed in the following table.

Touch Gesture	Mouse Action
Tap	Click
Double-tap	Double-click
Press and slide	Press left mouse button and move

Combo box, list box, text, double, currency, and date time cells use the standard .NET control and the standard control's touch policy. When the standard .NET control text box has focus and is in edit mode, a text selection gripper is displayed. This is supported by the browser.



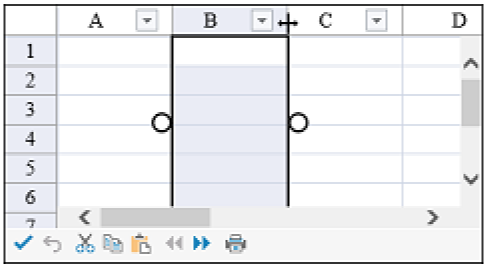
Tap an item in the list box cell to select the item.

The multiple-column combo box cell uses a larger row height in the drop-down list when using touch support.

Using Touch Support with Filtering

You can use touch gestures when filtering.

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

Set the **HeaderIndicatorPositionAdjusting ('HeaderIndicatorPositionAdjusting Property' in the on-line documentation)** property.

Example

This example creates a filter and sets the **HeaderIndicatorPositionAdjusting ('HeaderIndicatorPositionAdjusting Property' in the on-line documentation)** property.

C#

```
FpSpread1.Sheets[0].Columns.Count = 10;
FpSpread1.Sheets[0].Rows.Count = 20;
FpSpread1.Sheets[0].AutoFilterMode = FarPoint.Web.Spread.AutoFilterMode.Enhanced;
FarPoint.Web.Spread.NamedStyle instyle = new FarPoint.Web.Spread.NamedStyle();
FarPoint.Web.Spread.NamedStyle outstyle = new FarPoint.Web.Spread.NamedStyle();
instyle.BackColor = System.Drawing.Color.Yellow;
outstyle.BackColor = System.Drawing.Color.Aquamarine;
FarPoint.Web.Spread.FilterColumnDefinition fcd = new
FarPoint.Web.Spread.FilterColumnDefinition(1,
FarPoint.Web.Spread.FilterListBehavior.SortByMostOccurrences |
FarPoint.Web.Spread.FilterListBehavior.Default);
FarPoint.Web.Spread.FilterColumnDefinition fcd1 = new
FarPoint.Web.Spread.FilterColumnDefinition(2);
FarPoint.Web.Spread.FilterColumnDefinition fcd2 = new
FarPoint.Web.Spread.FilterColumnDefinition();
FarPoint.Web.Spread.StyleRowFilter sf = new
FarPoint.Web.Spread.StyleRowFilter(FpSpread1.Sheets[0], instyle, outstyle);
sf.AddColumn(fcd);
sf.AddColumn(fcd1);
sf.AddColumn(fcd2);
FpSpread1.Sheets[0].RowFilter = sf;
FpSpread1.HeaderIndicatorPositionAdjusting = 10;
```

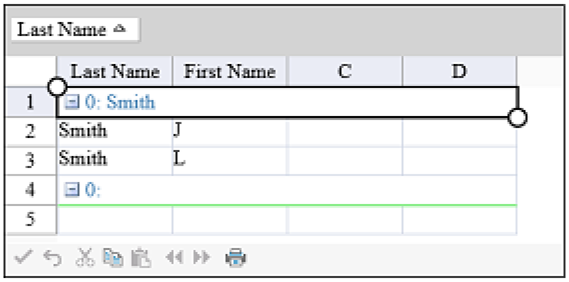
VB

```
FpSpread1.Sheets(0).Columns.Count = 10
FpSpread1.Sheets(0).Rows.Count = 20
FpSpread1.Sheets(0).AutoFilterMode = FarPoint.Web.Spread.AutoFilterMode.Enhanced
Dim instyle As New FarPoint.Web.Spread.NamedStyle()
Dim outstyle As New FarPoint.Web.Spread.NamedStyle()
instyle.BackColor = Drawing.Color.Yellow
outstyle.BackColor = Drawing.Color.Aquamarine
Dim fcd As New FarPoint.Web.Spread.FilterColumnDefinition(1,
FarPoint.Web.Spread.FilterListBehavior.SortByMostOccurrences Or
FarPoint.Web.Spread.FilterListBehavior.Default)
Dim fcd1 As New FarPoint.Web.Spread.FilterColumnDefinition(2)
Dim fcd2 As New FarPoint.Web.Spread.FilterColumnDefinition()
Dim sf As New FarPoint.Web.Spread.StyleRowFilter(FpSpread1.Sheets(0), instyle,
outstyle)
sf.AddColumn(fcd)
sf.AddColumn(fcd1)
sf.AddColumn(fcd2)
FpSpread1.Sheets(0).RowFilter = sf
FpSpread1.HeaderIndicatorPositionAdjusting = 10
```

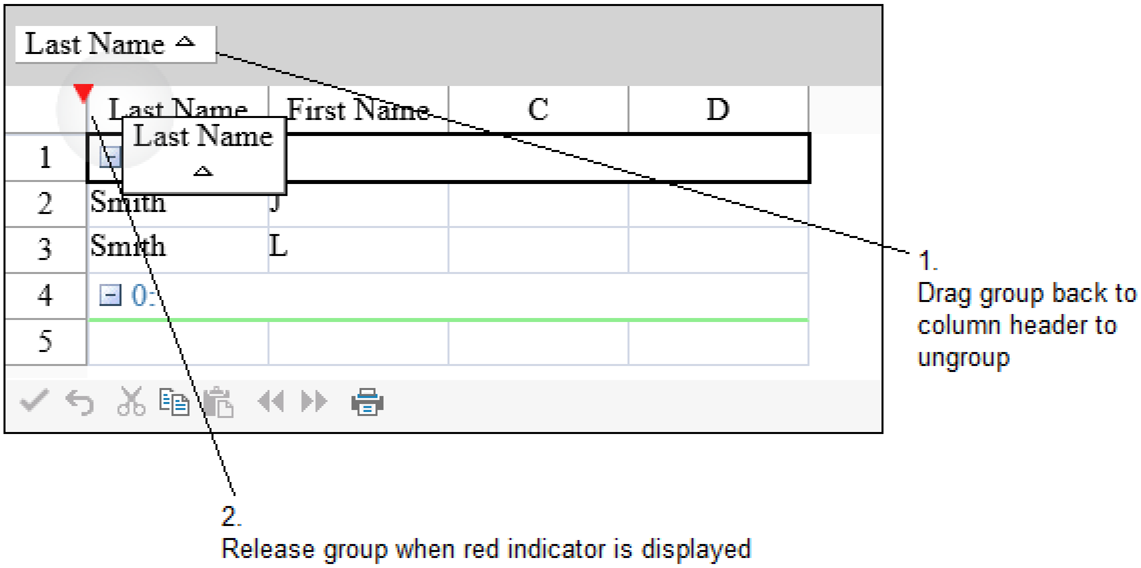
Using Touch Support with Grouping

You can use touch gestures when grouping.

Tap to select a column, then press down on a column header and slide to the group bar area. Release to create a group. You can also double-tap on a column header to create a group. The following image displays a group.



You can remove the group by dragging the group back to the column header area.


Last Name
Last Name
First Name
C
D
Last Name
1
2
Smith
J
3
Smith
L
4
0:
5
1.
Drag group back to column header to ungroup
2.
Release group when red indicator is displayed

Tap the group header button area to sort.

Using Code

Set the **AllowGroup ('AllowGroup Property' in the on-line documentation)**, **GroupBarVisible ('GroupBarVisible Property' in the on-line documentation)**, and **AllowColumnMove ('AllowColumnMove Property' in the on-line documentation)** properties.

Example

This example allows the user to group.

C#

```
FpSpread1.ActiveSheetView.ColumnHeader.Cells[0, 0].Text = "Last Name";
FpSpread1.ActiveSheetView.ColumnHeader.Cells[0, 1].Text = "First Name";
FpSpread1.ActiveSheetView.Cells[0, 0].Text = "Smith";
FpSpread1.ActiveSheetView.Cells[0, 1].Text = "J";
FpSpread1.ActiveSheetView.Cells[1, 0].Text = "Smith";
FpSpread1.ActiveSheetView.Cells[1, 1].Text = "L";
FpSpread1.ActiveSheetView.AllowColumnMove = true;
FpSpread1.ActiveSheetView.GroupBarVisible = true;
FpSpread1.ActiveSheetView.AllowGroup = true;
```

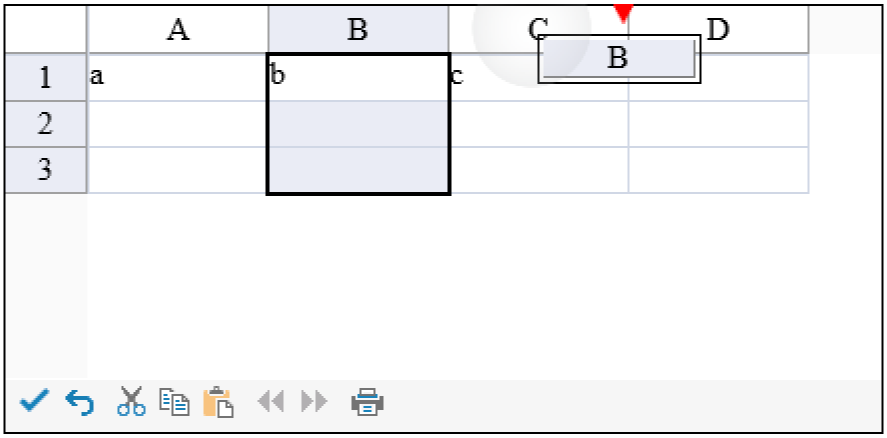
VB

```
FpSpread1.ActiveSheetView.ColumnHeader.Cells(0, 0).Text = "Last Name"
FpSpread1.ActiveSheetView.ColumnHeader.Cells(0, 1).Text = "First Name"
FpSpread1.ActiveSheetView.Cells(0, 0).Text = "Smith"
FpSpread1.ActiveSheetView.Cells(0, 1).Text = "J"
FpSpread1.ActiveSheetView.Cells(1, 0).Text = "Smith"
FpSpread1.ActiveSheetView.Cells(1, 1).Text = "L"
FpSpread1.ActiveSheetView.AllowColumnMove = True
FpSpread1.ActiveSheetView.GroupBarVisible = True
FpSpread1.ActiveSheetView.AllowGroup = True
```

Using Touch Support when Moving Columns

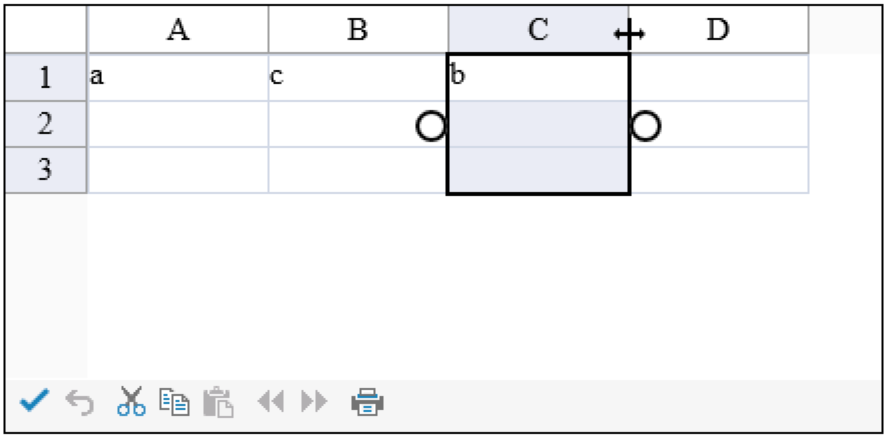
You can use touch gestures to move columns.

Tap the column header to select it, then slide to the target location. Release to move the column.



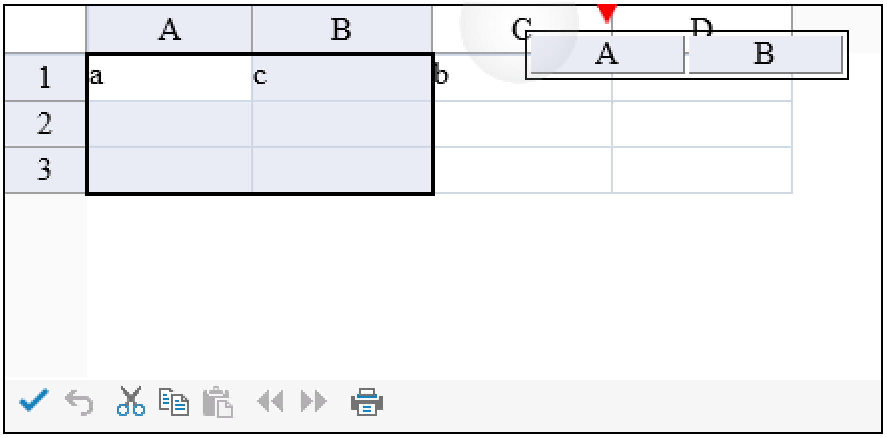
	A	B	C	D
1	a	b	c	
2				
3				

The following image displays the column after the move action has been completed.



	A	B	C	D
1	a	c	b	
2				
3				

Select a column 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.

Refer to **Using Touch Support with Selections** for more information on how to select a column or row.

Using Code

Set the **AllowColumnMove ('AllowColumnMove Property' in the on-line documentation)** property.

Example

This example code sets the **AllowColumnMove ('AllowColumnMove Property' in the on-line documentation)** property.

C#

```
FpSpread1.ActiveSheetView.AllowColumnMove = true;
```

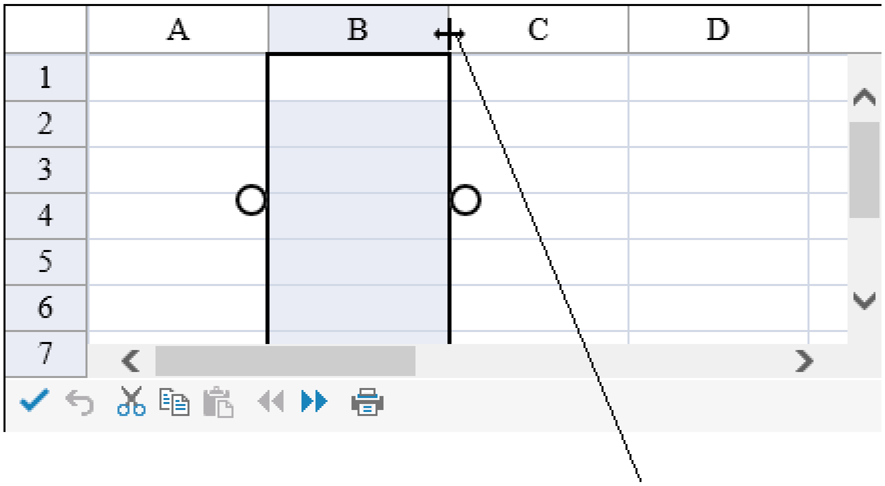
VB

```
FpSpread1.ActiveSheetView.AllowColumnMove = True
```

Using Touch Support when Resizing Columns or Rows

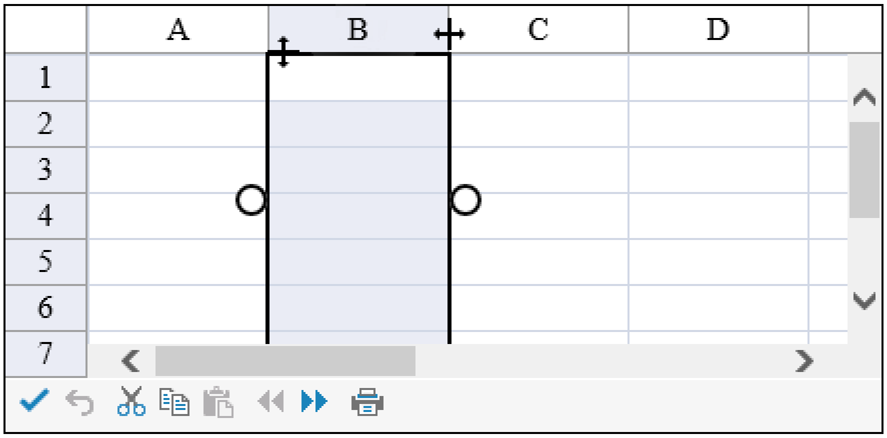
You can resize columns or rows using touch gestures.

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.



Resize handle

Spread displays the column and row resize handles if the **AllowHeaderResize ('AllowHeaderResize Property' in the on-line documentation)** property is true. The following image displays both handles.



Row resizing is disabled if the **RowTemplateLayoutMode** is used.

Using Code

Set the **AllowHeaderResize ('AllowHeaderResize Property' in the on-line documentation)** and **Resizable ('Resizable Property' in the on-line documentation)** properties.

Example

This example allows the user to resize headers, columns, and rows.

C#

```
FpSpread1.Sheets[0].Columns.Count = 10;
FpSpread1.Sheets[0].Rows.Count = 20;
FpSpread1.AllowHeaderResize = true;
FpSpread1.ActiveSheetView.Columns[0, 5].Resizable = true;
FpSpread1.ActiveSheetView.Rows[0, 10].Resizable = true;
```

VB

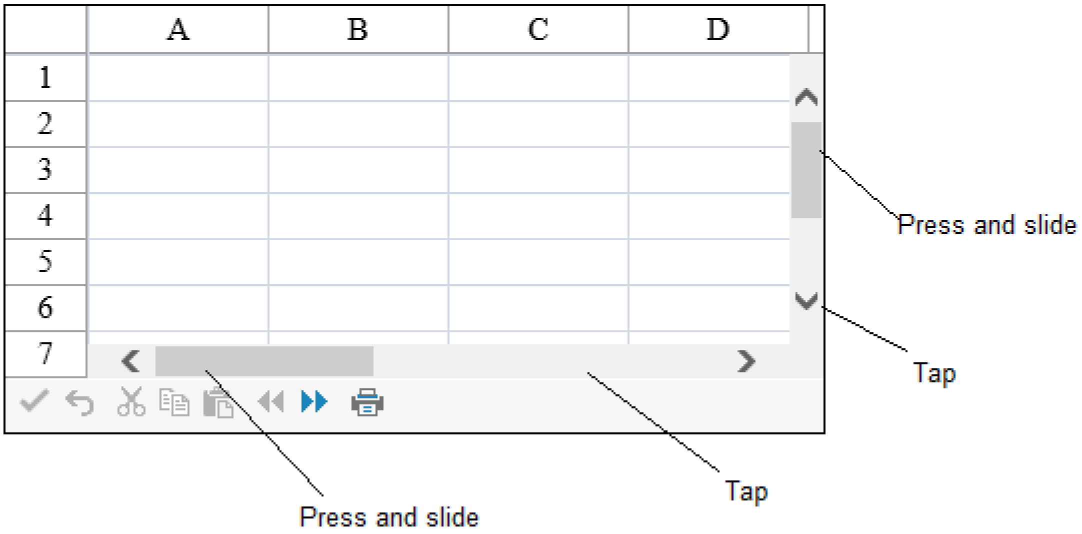
```
FpSpread1.Sheets(0).Columns.Count = 10
```

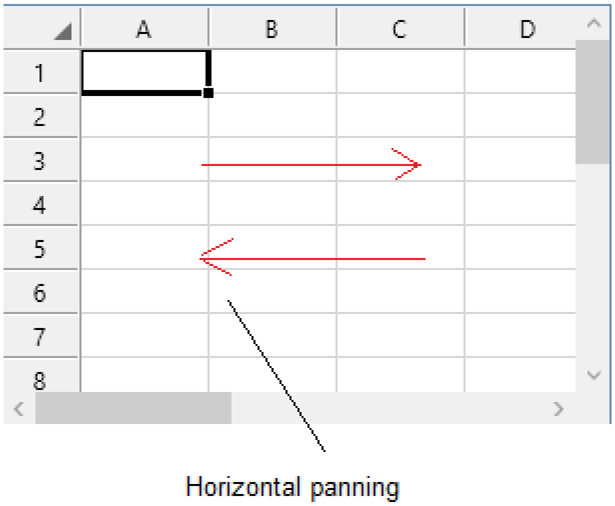
```
FpSpread1.Sheets(0).Rows.Count = 20
FpSpread1.AllowHeaderResize = True
FpSpread1.ActiveSheetView.Columns(0, 5).Resizable = True
FpSpread1.ActiveSheetView.Rows(0, 10).Resizable = True
```

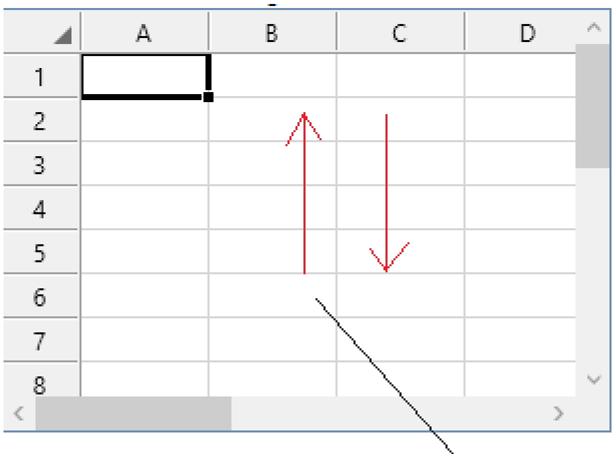
Using Touch Support with Scrolling

You can use touch gestures when scrolling in the control.

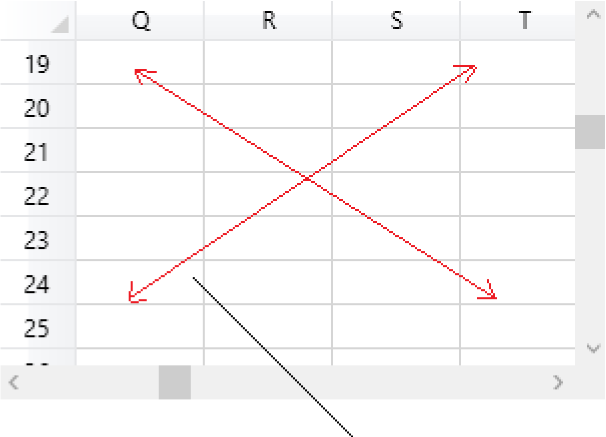
You can tap the scroll bar or press and slide the scroll bar to scroll. You can also use panning gestures in the cell area of the control (vertical, horizontal, or oblique). Panning in the diagonal direction scrolls horizontally and vertically. Specify the type of panning mode with the **PanningMode ('PanningMode Property' in the on-line documentation)** property and **SpreadPanningMode ('SpreadPanningMode Enumeration' in the on-line documentation)** enumeration.







Vertical panning



Diagonal panning

Panning does not apply to header or footer areas.

Using Code

Set the **PanningMode ('PanningMode Property' in the on-line documentation)** property.

Example

This example sets the **PanningMode ('PanningMode Property' in the on-line documentation)** property to Both.

C#

```
FpSpread1.Sheets[0].Columns.Count = 10;
FpSpread1.Sheets[0].Rows.Count = 20;
FpSpread1.TouchInfo.PanningMode = FarPoint.Web.Spread.SpreadPanningMode.Both;
```

VB

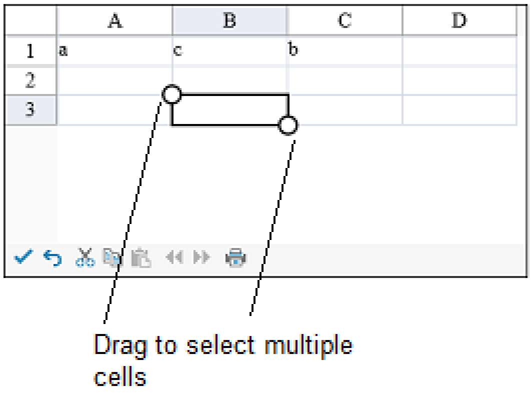
```
FpSpread1.Sheets(0).Columns.Count = 10
FpSpread1.Sheets(0).Rows.Count = 20
FpSpread1.TouchInfo.PanningMode = FarPoint.Web.Spread.SpreadPanningMode.Both
```

Using Touch Support with Selections

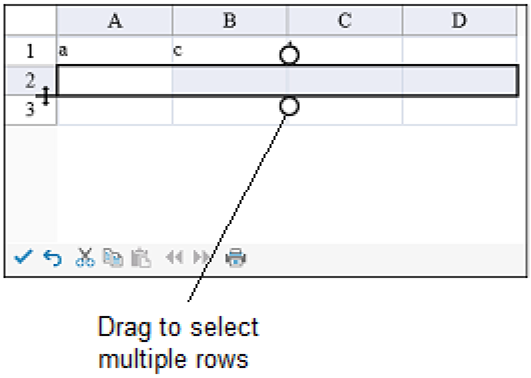
You can select columns, rows, cell ranges, and the entire control using touch gestures.

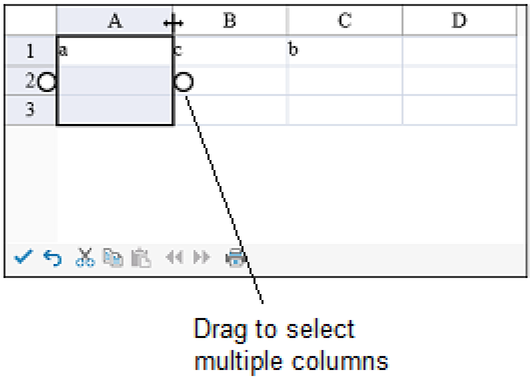
The selection gripper is displayed when touching the cell, column, or row. The gripper is not displayed when using the mouse or keyboard. The selection grippers are displayed on the outside edge of the range (top-left and bottom-right edges, by default). The border is displayed around the selected cell range when using touch operations.

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.

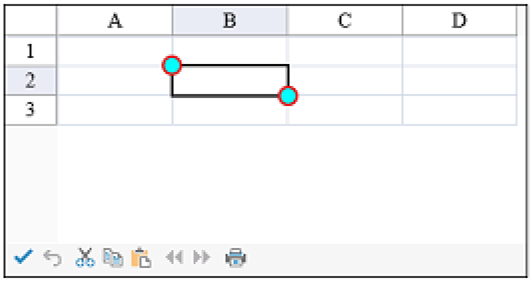




You can select the entire control by tapping the corner header.

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 customize the gripper appearance using the **SelectionGripperThickness ('SelectionGripperThickness Property' in the on-line documentation)**, **SelectionGripperLineColor ('SelectionGripperLineColor Property' in the on-line documentation)**, and **SelectionGripperBackColor ('SelectionGripperBackColor Property' in the on-line documentation)** properties as in the following image.



Using Code

Set the **SelectionGripperThickness ('SelectionGripperThickness Property' in the on-line documentation)**, **SelectionGripperLineColor ('SelectionGripperLineColor Property' in the on-line documentation)**, and **SelectionGripperBackColor ('SelectionGripperBackColor Property' in the on-line documentation)** properties.

Example

This example sets the selection gripper appearance.

C#

```
FpSpread1.TouchInfo.SelectionGripperBackColor = System.Drawing.Color.Aqua;
FpSpread1.TouchInfo.SelectionGripperLineColor = System.Drawing.Color.Red;
FpSpread1.TouchInfo.SelectionGripperThickness = 2;
```

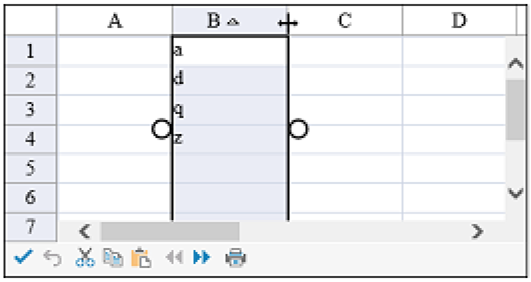
VB

```
FpSpread1.TouchInfo.SelectionGripperBackColor = System.Drawing.Color.Aqua
FpSpread1.TouchInfo.SelectionGripperLineColor = System.Drawing.Color.Red
FpSpread1.TouchInfo.SelectionGripperThickness = 2
```

Using Touch Support with Sorting

You can use touch gestures when sorting.

Double-tap the sort indicator to sort. The **AllowSort ('AllowSort Property' in the on-line documentation)** property must be true to use touch gestures. The following image displays a sort indicator.



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

Set the **HeaderIndicatorPositionAdjusting ('HeaderIndicatorPositionAdjusting Property' in the on-line documentation)** property.

Example

This example sets the **HeaderIndicatorPositionAdjusting ('HeaderIndicatorPositionAdjusting Property' in the on-line documentation)** property.

C#

```
FpSpread1.Sheets[0].Columns.Count = 10;
FpSpread1.Sheets[0].Rows.Count = 20;
FpSpread1.Sheets[0].AllowSort = true;
FpSpread1.HeaderIndicatorPositionAdjusting = 5;
```

VB

```
FpSpread1.Sheets(0).Columns.Count = 10
FpSpread1.Sheets(0).Rows.Count = 20
FpSpread1.Sheets(0).AllowSort = True
FpSpread1.HeaderIndicatorPositionAdjusting = 5
```

2 Index