ComponentOne

# Grid for WPF

*Corporate Headquarters*
**ComponentOne LLC**
201 South Highland Avenue
3rd Floor
Pittsburgh, PA 15206 • USA

**Internet:** info@ComponentOne.com
**Web site:** http://www.componentone.com

**Sales**
E-mail: sales@componentone.com
Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)

**Trademarks**

The ComponentOne product name is a trademark and ComponentOne is a registered trademark of ComponentOne LLC. All other trademarks used herein are the properties of their respective owners.

**Warranty**

ComponentOne warrants that the original CD (or diskettes) are free from defects in material and workmanship, assuming normal use, for a period of 90 days from the date of purchase. If a defect occurs during this time, you may return the defective CD (or disk) to ComponentOne, along with a dated proof of purchase, and ComponentOne will replace it at no charge. After 90 days, you can obtain a replacement for a defective CD (or disk) by sending it and a check for $25 (to cover postage and handling) to ComponentOne.

Except for the express warranty of the original CD (or disks) set forth here, ComponentOne makes no other warranties, express or implied. Every attempt has been made to ensure that the information contained in this manual is correct as of the time it was written. We are not responsible for any errors or omissions. ComponentOne's liability is limited to the amount you paid for the product. ComponentOne is not liable for any special, consequential, or other damages for any reason.

**Copying and Distribution**

While you are welcome to make backup copies of the software for your own use and protection, you are not permitted to make copies for the use of anyone else. We put a lot of time and effort into creating this product, and we appreciate your support in seeing that it is used by licensed users only.

This manual was produced using ComponentOne Doc-To-Help™.

# Table of Contents

# ComponentOne Grid for WPF Overview

Turn data manipulation into a truly unique interaction experience for your end users by taking advantage of **ComponentOne Grid for WPF**! **ComponentOne Grid for WPF** takes the standard grid to a whole new level with revolutionary dynamic data manipulation and a powerful carousel panel. Traditional grids allow you to display, format, and edit tabular data; **Grid for WPF** builds on this enabling you to adapt grid appearance, behavior, and layout to create a completely tailored user experience quickly and easily.

Using **Grid for WPF**'s C1DataGrid control you can completely customize sorting, grouping, and filtering interactions. View controls in a highly-interactive carousel using the C1CarouselPanel panel. With code-free, XAML-based customization, you can design your grid application's user interface from the ground up or base your design on one of the included Microsoft Office 2007 or Windows Vista themes and flexible data view templates.

**Getting Started**

Get started with the following topics:

- Key Features (page 20)

- Quick Start (page 25)

- Samples (page 137)

- Task-Basked Help (page 137)

## What's New in Grid for WPF

This documentation was last revised for 2010 v1 on January 6, 2010. The following enhancements were made to **ComponentOne Grid for WPF** in the 2010 v1 release:

### Performance Gains with the C1CollectionView Class

The C1CollectionView class was added to **Grid for WPF**. This class can be used as a replacement for native **CollectionView**, **ListCollectionView** and **BindingListCollectionView** classes. Using the C1CollectionView class rather than the **CollectionView**, **ListCollectionView** and **BindingListCollectionView** classes can provide the following performance benefits:

- **Improved Local Sorting**

  Local sorting performed by the view (this is analogous of sorting operations performed by the Microsoft **ListCollectionView**) completes 5 times faster.

- **Improved Grouping**

  Grouping completes up to ten times as fast as Microsoft's **ListCollectionView** or **BindingListCollectionView**. Particularly when in a performance-intensive scenario, such as if there is one group per item and a huge number of groups, C1CollectionView improves performance significantly.

See Using the C1CollectionView Class (page 53) for additional functional benefits.

### New Classes

The following members were added to **Grid for WPF**:

| Class | Description |
|---|---|
| C1CollectionView | Exports the grid content to Microsoft Excel format file or stream using the specified export settings. For more information, see Exporting Data (page 58) and for an example, see Exporting Data to an Excel File (page 168). |

**Tip:** A version history containing a list of new features, improvements, fixes, and changes for each product is available on HelpCentral at http://helpcentral.componentone.com/VersionHistory.aspx.

# Revision History

The revision history provides recent enhancements to **ComponentOne Grid for WPF**.

## What's New in 2009 v3

The following enhancements were made to **ComponentOne Grid for WPF** in the 2009 v3 release:

- **Export to Excel**

  Support for exporting grid content to Microsoft Excel was added with the new ExportToExcel method. For more information, see Exporting Data (page 58) and for an example, see Exporting Data to an Excel File (page 168).

- **Filter Bar Content Template**

  Using the newly added ColumnFilterCellContentTemplate property you can create and assign a template defining the filter bar cell content's user interface. For more information about templates, see Editing Templates (page 108).

### New Class Members

The following members were added to **Grid for WPF**:

| Class | Member | Description |
|---|---|---|
| C1DataGrid | ExportToExcel method | Exports the grid content to Microsoft Excel format file or stream using the specified export settings. For more information, see Exporting Data (page 58) and for an example, see Exporting Data to an Excel File (page 168). |
| Column | ColumnFilterCellContentTemplate property | Gets or sets a template that defines the UI for a content of the Column cell in the C1DataGrid Filter Bar. For more information about templates, see Editing Templates (page 108). |

## What's New in 2009 v2

There were no new features added to **ComponentOne Grid for WPF** in the 2009 v2 release.

## What's New in 2009 v1

The following enhancements were made to **ComponentOne Grid for WPF** in the 2009 v1 release:

### New Features

The following new features were added to **Grid for WPF**:

- **Calculate Totals Automatically**

You can now add totaling to columns in your grid. Easily total columns with sum, average, minimum, maximum, and count functions, or create your own custom totals function. See the Calculating Column Totals (page 56) topic and the Totals property description for details.

- **Reorder Columns Programmatically**

  You can now programmatically reorder columns in **C1DataGrid**'s actual column collections allowing you to change the visual order of columns. For an example, see the Reordering Columns Programmatically (page 166) topic.

- **Style Columns by Data Type**

  You can now style columns by type using type based column style mapping. You can format how columns appear depending on their **DataType** – for example, you can format columns differently depending on if the data they contain are strings, Boolean values, integers, and so on. For more information, see Using Type Based Column Styles (page 88).

- **Control When Updated Cell Values Are Committed**

  You can now control when cell values updated by the end user are committed to the underlying data source. You can choose if grid and column items are updated immediately or on cell end edit. For more details see Committing Data to a Data Source (page 58).

- **Add Column Value Formatting**

  You can now easily format a column as, for example, date, percent, currency, or so on using the Format property. For an example, see Formatting a Column as Currency (page 155).

- **Customize Text Wrapping and Alignment**

  You now have even more control over the appearance and formatting of text in the grid. In the 2009 v1 release, several properties controlling grid cell text wrapping and alignment were added. See Setting Text Alignment (page 106) and Setting Text Wrapping (page 107) for more information.

### Changes

The following changes were made to **Grid for WPF**:

- The return type of the **C1DataGrid**'s ActualColumns, ActualOrdinaryColumns, and ActualListColumns properties has been changed from ReadOnlyColumnCollection to ActualColumnCollection.
- Some default cell show templates have been redefined to use FormattedValue instead of Value. If you implemented custom templates based on such default templates, you will not see an effect after applying Format. You should correct this to bind to the FormattedValue property for the Format property to take effect.

### New Class Members

The following types were added to **Grid for WPF**:

| Class/Enumeration | Description |
|---|---|
| ActualColumnCollection class | Derived from the ReadOnlyColumnCollection class, this class adds the Move method that allows you to reorder columns in the collection. |
| GridTextAlignment enumeration | Justify various elements of the grid by setting their alignment to **General** (default), **Left**, **Right**, **Center**, or **Justify**. See Setting Text Alignment (page 106) for more information. |
| TextToHorizontalAlignmentConverter class | Represents a Binding two-way converter that converts from the **TextAlignment** to the **HorizontalAlignment** type. |
| Total class | Defines a total calculated for a Column. See the Calculating Column |

| | |
|---|---|
| | Totals (page 56) topic for details. |

The following members were added to existing classes in **Grid for WPF**:

| Class | Member | Description |
|---|---|---|
| C1DataGrid | AllowEdit property | Prohibits editing of column values for all grid columns. See Locking the Grid from Being Edited (page 151) for an example. |
| C1DataGrid | CommitItemCellValue property | Gets or sets a value indicating at which moment an updated grid item cell value will be committed to a data source (immediately or on cell end edit). For more details see Committing Data to a Data Source (page 58). |
| C1DataGrid | ConvertNullToDBNull property | Indicates whether a null value of grid cells are converted to **DBNull** when value commits to a data source. |
| C1DataGrid | DefaultTypeBasedColumnStyles property | An analogue of the TypeBasedColumnStyles property but intended for default style writers. |
| C1DataGrid | HeaderCellTextAlignment property | Defines a default text alignment of grid column header captions. |
| C1DataGrid | HeaderCellTextWrapping property | Defines a default text wrapping of grid column header captions. |
| C1DataGrid | ItemCellTextAlignment property | Defines a default text alignment of grid item cell values. |
| C1DataGrid | ItemCellTextWrapping property | Defines a default text wrapping for grid item cell values. |
| C1DataGrid | TotalResultCellTextAlignment property | Defines a default text alignment of grid column totals. |
| C1DataGrid | TotalResultCellTextWrapping property | Defines a default text wrapping of grid column totals. |
| C1DataGrid | TypeBasedColumnStyles property | Represents a dictionary that contains column styles that are applied to columns based on the column's **DataType** property value. Which Column.DataTape the style is applied to is defined by Style's key (x:Key) in the dictionary, which must be of the System.Type class. |
| C1GridFrame | Groups property | Gets a collection of GridViewGroup objects representing grid groups' data when grid is in group mode. |
| C1GridFrame | GroupTotalBarVisibility property | Gets or sets the visibility of the group's total bar. See Setting Group Total Visibility (page 103) for more information. |
| C1GridFrame | TotalBarVisibility property | Gets or sets the visibility of the total bar. See Setting Total Bar Visibility (page 101) for more information. |
| Column | ActualCommitItemCellValue property | Gets an effective commit mode for column's item cell values. For more details see Committing Data to |

| | | |
|---|---|---|
| | | a Data Source (page 58). |
| Column | ActualHeaderCellTextAlignment property | Gets an effective text alignment of column header caption. |
| Column | ActualHeaderCellTextWrapping property | Gets an effective text wrapping of the column header caption. |
| Column | ActualItemCellTextAlignment property | Gets an effective text alignment of column item cell values. |
| Column | ActualItemCellTextWrapping property | Gets an effective text wrapping of column item cell values. |
| Column | ActualTotalResultCellTextAlignment property | Gets an effective text alignment of column total result. |
| Column | ActualTotalResultCellTextWrapping property | Gets an effective text wrapping of a column total result. |
| Column | Format property | Defines a format that will be applied to column cell values when cell is in the show mode. For an example, see Formatting a Column as Currency (page 155). |
| Column | ItemCellTextAlignment property | Defines the text alignment of column item cell values. |
| Column | ItemCellTextWrapping property | Defines the text wrapping of column item cell values. |
| Column | HeaderCellTextAlignment property | Defines the text alignment of column header caption. |
| Column | HeaderCellTextWrapping property | Defines the text wrapping of the column header caption. |
| Column | TotalResultCellTextAlignment property | Defines the text alignment of column total result. |
| Column | TotalResultCellTextWrapping property | Defines the text wrapping of a column total result. |
| Column | CommitItemCellValue property | Gets or sets a value indicating at which moment an updated column item cell value will be committed to a data source (immediately or on cell end edit). For more details see Committing Data to a Data Source (page 58). |
| Column | Totals property | Represents a collection of Totals calculated for the column. |
| ContentCellPresenterBase | TextAlignment property | Gets a cell's text alignment |
| ContentCellPresenterBase | TextWrapping property | Gets a cell's text wrapping. |
| ItemCell | FormattedValue property | Returns an Value formatted according to a format specified in the corresponding Format property, or the Value itself if format is not specified.<br><br>Note: some default cell show templates are redefined so as to use FormattedValue instead of |

| | | |
|---|---|---|
| | | Value. If you implemented custom templates based on such default templates, you will not see an effect after applying Format. You should correct them so as to bind to the FormattedValue property in order the Format property will take effect. |

# Installing Grid for WPF

The following sections provide helpful information on installing **ComponentOne Grid for WPF**.

## Grid for WPF Setup Files

The installation program will create the directory **C:\Program Files\ComponentOne\Studio for WPF**, which contains the following subdirectories:

**Bin** Contains copies of all ComponentOne binaries (DLLs, EXEs). For **Component Grid for WPF**, the following DLLs are installed:

- C1.WPF.C1DataGrid.dll
- C1.WPF.C1DataGrid.Expression.Design.dll
- C1.WPF.C1DataGrid.VisualStudio.Design.dll
- C1.WPF.C1DataGrid.Expression.Design.4.dll
- C1.WPF.C1DataGrid.VisualStudio.Design.4.dll

**H2Help** Contains documentation for all Studio components.

**Samples**

Samples for the product are installed in the **ComponentOne Samples** folder by default. The path of the **ComponentOne Samples** directory is slightly different on Windows XP and Windows 7/Vista machines:

**Windows XP path:** C:\Documents and Settings\<username>\My Documents\ComponentOne Samples

**Windows 7/Vista path:** C:\Users\<username>\Documents\ComponentOne Samples

The **ComponentOne Samples** folder contains the following subdirectories:

**Common** Contains support and data files that are used by many of the demo programs.

**C1WPFGrid** Contains samples for **Grid for WPF**.

## System Requirements

System requirements include the following:

**Operating Systems:** Microsoft Windows® XP with Service Pack 2 (SP2)

Windows Vista™

Windows 2008 Server

Windows 7

| | |
|---|---|
| **Environments:** | .NET Framework 3.0 or later<br><br>Visual Studio® 2005 extensions for .NET Framework 2.0 November 2006 CTP<br><br>Visual Studio® 2008 |
| **Microsoft® Expression® Blend Compatibility:** | **Grid for WPF** includes special design-time support for Expression Blend, which includes workarounds for correct XAML serialization and a customizable user interface. |

**Note:** The **C1.WPF.C1DataGrid.VisualStudio.Design.dll** assembly is required by Visual Studio 2008 and the **C1.WPF.C1DataGrid.Expression.Design.dll** assembly is required by Expression Blend. The **C1.WPF.C1DataGrid.Expression.Design.dll** and **C1.WPF.C1DataGrid.VisualStudio.Design.dll** assemblies installed with **Grid for WPF** should always be placed in the same folder as **C1.WPF.C1DataGrid.dll**; the DLLs should NOT be placed in the Global Assembly Cache (GAC).

### Installing Demonstration Versions

If you wish to try **ComponentOne Grid for WPF** and do not have a serial number, follow the steps through the installation wizard and use the default serial number.

The only difference between unregistered (demonstration) and registered (purchased) versions of our products is that registered versions will stamp every application you compile so that a ComponentOne banner will not appear when your users run the applications.

### Uninstalling Grid for WPF

To uninstall **ComponentOne Grid for WPF**:

1. Open the **Control Panel** and select **Add or Remove Programs** (**Programs and Features** in Windows 7/Vista).
2. Select **ComponentOne Studio for WPF** and click the **Remove** button.
3. Click **Yes** to remove the program.

# End-User License Agreement

All of the ComponentOne licensing information, including the ComponentOne end-user license agreements, frequently asked licensing questions, and the ComponentOne licensing model, is available online at http://www.componentone.com/SuperPages/Licensing/.

# Licensing FAQs

This section describes the main technical aspects of licensing. It may help the user to understand and resolve licensing problems he may experience when using ComponentOne .NET and ASP.NET products.

### What is Licensing?

Licensing is a mechanism used to protect intellectual property by ensuring that users are authorized to use software products.

Licensing is not only used to prevent illegal distribution of software products. Many software vendors, including ComponentOne, use licensing to allow potential users to test products before they decide to purchase them.

Without licensing, this type of distribution would not be practical for the vendor or convenient for the user. Vendors would either have to distribute evaluation software with limited functionality, or shift the burden of managing software licenses to customers, who could easily forget that the software being used is an evaluation version and has not been purchased.

## How does Licensing Work?

ComponentOne uses a licensing model based on the standard set by Microsoft, which works with all types of components.

**Note:** The **Compact Framework** components use a slightly different mechanism for run-time licensing than the other ComponentOne components due to platform differences.

When a user decides to purchase a product, he receives an installation program and a Serial Number. During the installation process, the user is prompted for the serial number that is saved on the system. (Users can also enter the serial number by clicking the **License** button on the **About Box** of any ComponentOne product, if available, or by rerunning the installation and entering the serial number in the licensing dialog box.)

When a licensed component is added to a form or Web page, Visual Studio obtains version and licensing information from the newly created component. When queried by Visual Studio, the component looks for licensing information stored in the system and generates a run-time license and version information, which Visual Studio saves in the following two files:

- An assembly resource file which contains the actual run-time license.
- A "licenses.licx" file that contains the licensed component strong name and version information.

These files are automatically added to the project.

In WinForms and ASP.NET 1.x applications, the run-time license is stored as an embedded resource in the assembly hosting the component or control by Visual Studio. In ASP.NET 2.x applications, the run-time license may also be stored as an embedded resource in the **App_Licenses.dll** assembly, which is used to store all run-time licenses for all components directly hosted by WebForms in the application. Thus, the **App_licenses.dll** must always be deployed with the application.

The **licenses.licx** file is a simple text file that contains strong names and version information for each of the licensed components used in the application. Whenever Visual Studio is called upon to rebuild the application resources, this file is read and used as a list of components to query for run-time licenses to be embedded in the appropriate assembly resource. Note that editing or adding an appropriate line to this file can force Visual Studio to add run-time licenses of other controls as well.

Note that the **licenses.licx** file is usually not shown in the Solution Explorer; it appears if you press the **Show All Files** button in the Solution Explorer's Toolbox or, from Visual Studio's main menu, select **Show All Files** on the **Project** menu.

Later, when the component is created at run time, it obtains the run-time license from the appropriate assembly resource that was created at design time and can decide whether to simply accept the run-time license, to throw an exception and fail altogether, or to display some information reminding the user that the software has not been licensed.

All ComponentOne products are designed to display licensing information if the product is not licensed. None will throw licensing exceptions and prevent applications from running.

## Common Scenarios

The following topics describe some of the licensing scenarios you may encounter.

### Creating components at design time

This is the most common scenario and also the simplest: the user adds one or more controls to the form, the licensing information is stored in the **licenses.licx** file, and the component works.

Note that the mechanism is exactly the same for Windows Forms and Web Forms (ASP.NET) projects.

### Creating components at run time

This is also a fairly common scenario. You do not need an instance of the component on the form, but would like to create one or more instances at run time.

In this case, the project will not contain a **licenses.licx** file (or the file will not contain an appropriate run-time license for the component) and therefore licensing will fail.

To fix this problem, add an instance of the component to a form in the project. This will create the **licenses.licx** file and things will then work as expected. (The component can be removed from the form after the **licenses.licx** file has been created).

Adding an instance of the component to a form, then removing that component, is just a simple way of adding a line with the component strong name to the **licenses.licx** file. If desired, you can do this manually using notepad or Visual Studio itself by opening the file and adding the text. When Visual Studio recreates the application resources, the component will be queried and its run-time license added to the appropriate assembly resource.

### Inheriting from licensed components

If a component that inherits from a licensed component is created, the licensing information to be stored in the form is still needed. This can be done in two ways:

- Add a **LicenseProvider** attribute to the component.

  This will mark the derived component class as licensed. When the component is added to a form, Visual Studio will create and manage the **licenses.licx** file and the base class will handle the licensing process as usual. No additional work is needed. For example:

```
[LicenseProvider(typeof(LicenseProvider))]
class MyGrid: C1.Win.C1FlexGrid.C1FlexGrid
{
  // ...
}
```

- Add an instance of the base component to the form.

  This will embed the licensing information into the **licenses.licx** file as in the previous scenario and the base component will find it and use it. As before, the extra instance can be deleted after the **licenses.licx** file has been created.

Please note that ComponentOne licensing will not accept a run-time license for a derived control if the run-time license is embedded in the same assembly as the derived class definition and the assembly is a DLL. This restriction is necessary to prevent a derived control class assembly from being used in other applications without a design-time license. If you create such an assembly, you will need to take one of the actions previously described create a component at run time.

### Using licensed components in console applications

When building console applications, there are no forms to add components to and therefore Visual Studio won't create a **licenses.licx** file.

In these cases, create a temporary Windows Forms application and add all the desired licensed components to a form. Then close the Windows Forms application and copy the **licenses.licx** file into the console application project.

Make sure the **licenses.licx** file is configured as an embedded resource. To do this, right-click the **licenses.licx** file in the Solution Explorer window and select **Properties**. In the Properties window, set the **Build Action** property to **Embedded Resource**.

### *Using licensed components in Visual C++ applications*

There is an issue in VC++ 2003 where the **licenses.licx** is ignored during the build process; therefore, the licensing information is not included in VC++ applications.

To fix this problem, extra steps must be taken to compile the licensing resources and link them to the project. Note the following:

1. Build the C++ project as usual. This should create an EXE file and also a licenses.licx file with licensing information in it.
2. Copy the **licenses.licx** file from the application directory to the target folder (**Debug** or **Release**).
3. Copy the **C1Lc.exe** utility and the licensed DLLs to the target folder. (Don't use the standard lc.exe, it has bugs.)
4. Use **C1Lc.exe** to compile the **licenses.licx** file. The command line should look like this:
   ```
   c1lc /target:MyApp.exe /complist:licenses.licx /i:C1.Win.C1FlexGrid.dll
   ```
5. Link the licenses into the project. To do this, go back to Visual Studio, right-click the project, select **Properties**, and go to the **Linker/Command Line** option. Enter the following:
   ```
   /ASSEMBLYRESOURCE:Debug\MyApp.exe.licenses
   ```
6. Rebuild the executable to include the licensing information in the application.

### *Using licensed components with automated testing products*

Automated testing products that load assemblies dynamically may cause them to display license dialog boxes. This is the expected behavior since the test application typically does not contain the necessary licensing information and there is no easy way to add it.

This can be avoided by adding the string "C1CheckForDesignLicenseAtRuntime" to the **AssemblyConfiguration** attribute of the assembly that contains or derives from ComponentOne controls. This attribute value directs the ComponentOne controls to use design-time licenses at run time.

For example:

```
#if AUTOMATED_TESTING
   [AssemblyConfiguration("C1CheckForDesignLicenseAtRuntime")]
#endif
   public class MyDerivedControl : C1LicensedControl
   {
         // ...
   }
```

Note that the **AssemblyConfiguration** string may contain additional text before or after the given string, so the **AssemblyConfiguration** attribute can be used for other purposes as well. For example:

```
[AssemblyConfiguration("C1CheckForDesignLicenseAtRuntime,BetaVersion")]
```

THIS METHOD SHOULD ONLY BE USED UNDER THE SCENARIO DESCRIBED. It requires a design-time license to be installed on the testing machine. Distributing or installing the license on other computers is a violation of the EULA.

## Troubleshooting

We try very hard to make the licensing mechanism as unobtrusive as possible, but problems may occur for a number of reasons.

Below is a description of the most common problems and their solutions.

### *I have a licensed version of a ComponentOne product but I still get the splash screen when I run my project.*

If this happens, there may be a problem with the **licenses.licx** file in the project. It either doesn't exist, contains wrong information, or is not configured correctly.

First, try a full rebuild (**Rebuild All** from the Visual Studio **Build** menu). This will usually rebuild the correct licensing resources.

**If that fails follow these steps:**

1. Open the project and go to the Solution Explorer window.
2. Click the **Show All Files** button on the top of the window.
3. Find the **licenses.licx** file and open it. If prompted, continue to open the file.
4. Change the version number of each component to the appropriate value. If the component does not appear in the file, obtain the appropriate data from another **licenses.licx** file or follow the alternate procedure following.
5. Save the file, then close the **licenses.licx** tab.
6. Rebuild the project using the **Rebuild All** option (not just **Rebuild**).

**Alternatively, follow these steps:**

1. Open the project and go to the Solution Explorer window.
2. Click the **Show All Files** button on the top of the window.
3. Find the **licenses.licx** file and delete it.
4. Close the project and reopen it.
5. Open the main form and add an instance of each licensed control.
6. Check the Solution Explorer window, there should be a **licenses.licx** file there.
7. Rebuild the project using the **Rebuild All** option (not just **Rebuild**).

**For ASP.NET 2.x applications, follow these steps:**

1. Open the project and go to the Solution Explorer window.
2. Find the **licenses.licx** file and right-click it.
3. Select the **Rebuild Licenses** option (this will rebuild the **App_Licenses.licx** file).
4. Rebuild the project using the **Rebuild All** option (not just **Rebuild**).

### *I have a licensed version of a ComponentOne product on my Web server but the components still behave as unlicensed.*

There is no need to install any licenses on machines used as servers and not used for development.

The components must be licensed on the development machine, therefore the licensing information will be saved into the executable (.exe or .dll) when the project is built. After that, the application can be deployed on any machine, including Web servers.

For ASP.NET 2.x applications, be sure that the App_Licenses.dll assembly created during development of the application is deployed to the bin application bin directory on the Web server.

If your ASP.NET application uses WinForms user controls with constituent licensed controls, the runtime license is embedded in the WinForms user control assembly. In this case, you must be sure to rebuild and update the user control whenever the licensed embedded controls are updated.

### *I downloaded a new build of a component that I have purchased, and now I'm getting the splash screen when I build my projects.*

Make sure that the serial number is still valid. If you licensed the component over a year ago, your subscription may have expired. In this case, you have two options:

**Option 1 – Renew your subscription to get a new serial number.**

If you choose this option, you will receive a new serial number that you can use to license the new components (from the installation utility or directly from the **About Box**).

The new subscription will entitle you to a full year of upgrades and to download the latest maintenance builds directly from http://prerelease.componentone.com/.

**Option 2 – Continue to use the components you have.**

Subscriptions expire, products do not. You can continue to use the components you received or downloaded while your subscription was valid.

# Technical Support

ComponentOne offers various support options. For a complete list and a description of each, visit the ComponentOne Web site at http://www.componentone.com/Support.

Some methods for obtaining technical support include:

- **Online Support via HelpCentral**
  ComponentOne HelpCentral provides customers with a comprehensive set of technical resources in the form of FAQs, samples, Version Release History, Articles, searchable Knowledge Base, searchable Online Help and more. We recommend this as the first place to look for answers to your technical questions.
- **Online Support via our Incident Submission Form**
  This online support service provides you with direct access to our Technical Support staff via an online incident submission form. When you submit an incident, you'll immediately receive a response via e-mail confirming that you've successfully created an incident. This email will provide you with an Issue Reference ID and will provide you with a set of possible answers to your question from our Knowledgebase. You will receive a response from one of the ComponentOne staff members via e-mail in 2 business days or less.
- **Peer-to-Peer Product Forums and Newsgroups**
  ComponentOne peer-to-peer product forums and newsgroups are available to exchange information, tips, and techniques regarding ComponentOne products. ComponentOne sponsors these areas as a forum for users to share information. While ComponentOne does not provide direct support in the forums and newsgroups, we periodically monitor them to ensure accuracy of information and provide comments when appropriate. Please note that a ComponentOne User Account is required to participate in the ComponentOne Product Forums.
- **Installation Issues**
  Registered users can obtain help with problems installing ComponentOne products. Contact technical support by using the online incident submission form or by phone (412.681.4738). Please note that this does not include issues related to distributing a product to end users in an application.
- **Documentation**
  ComponentOne documentation is installed with each of our products and is also available online at HelpCentral. If you have suggestions on how we can improve our documentation, please email the Documentation team. Please note that e-mail sent to the Documentation team is for documentation feedback only. Technical Support and Sales issues should be sent directly to their respective departments.

**Note:** You must create a ComponentOne Account and register your product with a valid serial number to obtain support using some of the above methods.

# Redistributable Files

**ComponentOne Grid for WPF** is developed and published by ComponentOne LLC. You may use it to develop applications in conjunction with Microsoft Visual Studio or any other programming environment that enables the user to use and integrate the control(s). You may also distribute, free of royalties, the following Redistributable Files with any such application you develop to the extent that they are used separately on a single CPU on the client/workstation side of the network:

- C1.WPF.C1DataGrid.dll

**Note:** The DLL files are installed to the **C:\Program Files\ComponentOne\Studio for WPF\Bin** directory by default.

Site licenses are available for groups of multiple developers. Please contact Sales@ComponentOne.com for details.

# About this Documentation

You can create your grid applications using Microsoft Expression Blend or Visual Studio, but Blend is currently the only design-time environment that allows users to design XAML documents visually. In this documentation, we will use the **Design** workspace of Blend for most examples.

### Acknowledgements



### ComponentOne

If you have any suggestions or ideas for new features or controls, please call us or write:

*Corporate Headquarters*

**ComponentOne LLC**

201 South Highland Avenue

3rd Floor

Pittsburgh, PA 15206 • USA

412.681.4343

412.681.4384 (Fax)

http://www.componentone.com/

### ComponentOne Doc-To-Help

This documentation was produced using ComponentOne Doc-To-Help® Enterprise.

# XAML and XAML Namespaces

XAML is a declarative XML-based language that is used as a user interface markup language in Windows Presentation Foundation (WPF) and the .NET Framework 3.0. With XAML you can create a graphically rich customized user interface, perform data binding, and much more. For more information on XAML and the .NET Framework 3.0, please see http://www.microsoft.com.

### XAML Namespaces

Namespaces organize the objects defined in an assembly. Assemblies can contain multiple namespaces, which can in turn contain other namespaces. Namespaces prevent ambiguity and simplify references when using large groups of objects such as class libraries.

When you create a Microsoft Expression Blend project, a XAML file is created for you and some initial namespaces are specified:

| Namespace | Description |
|---|---|
| xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" | This is the default Windows Presentation Foundation namespace. |
| xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" | This is a XAML namespace that is mapped to the **x:** prefix. The **x:** prefix provides a quick, easy way to reference the namespace, which defines many commonly-used features necessary for WPF applications. |

When you add a C1DataGrid control to the window in Microsoft Expression Blend or Visual Studio, **Blend** or **Visual Studio** automatically creates an XML namespace for the control. The namespace looks like the following in Microsoft Expression Blend:

```
xmlns:c1grid="http://schemas.componentone.com/wpf/C1DataGrid"
```

The namespace value is **c1grid** and the XML namespace is **http://schemas.componentone.com/wpf/C1DataGrid**.

You can also choose to create your own custom name for the namespace. For example:

```
xmlns:MyGrid="clr-namespace:C1.WPF.C1DataGrid;assembly=C1.WPF.C1DataGrid"
```

You can now use your custom namespace when assigning properties, methods, and events. For example, use the following XAML to add a border around the grid:

```
<MyGrid:C1DataGrid Name="C1DataGrid1" BorderThickness="10,10,10,10">
```

# Creating a Microsoft Blend Project

To create a new Blend project, complete the following steps:

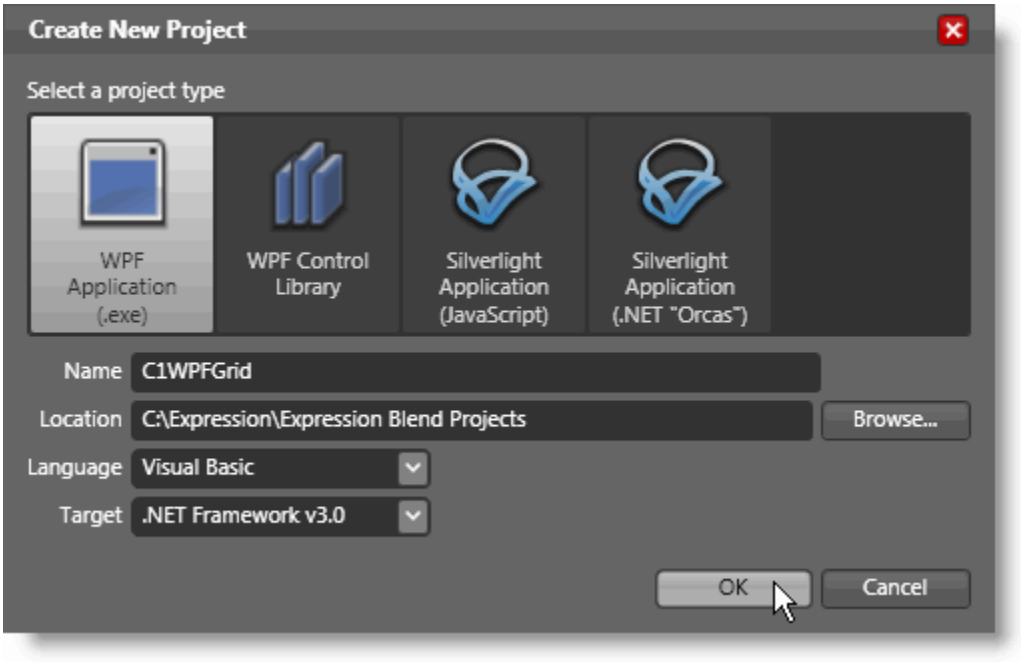
1. From the **File** menu, select **New Project** or click **New Project** in the Blend startup window.

   The **Create New Project** dialog box opens.

2. Make sure **WPF Application (.exe)** is selected and enter a name for the project in the Name text box. The **WPF Application (.exe)** creates a project for a Windows-based application that can be built and run while being designed.

3. Select the **Browse** button to specify a location for the project.

4. Select a language from the **Language** drop-down box and click **OK**.




A new Blend project with a XAML window is created.

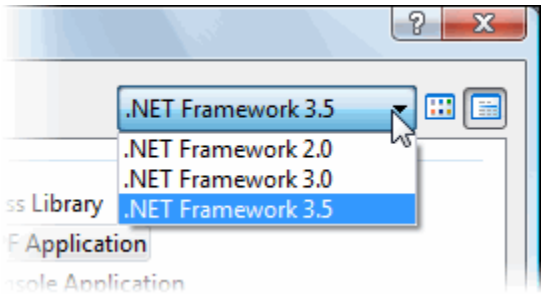
# Creating a .NET Project in Visual Studio

To create a new .NET project in Visual Studio 2008, complete the following steps:

1. From the **File** menu in Microsoft Visual Studio 2008, select **New Project**.

   The **New Project** dialog box opens.

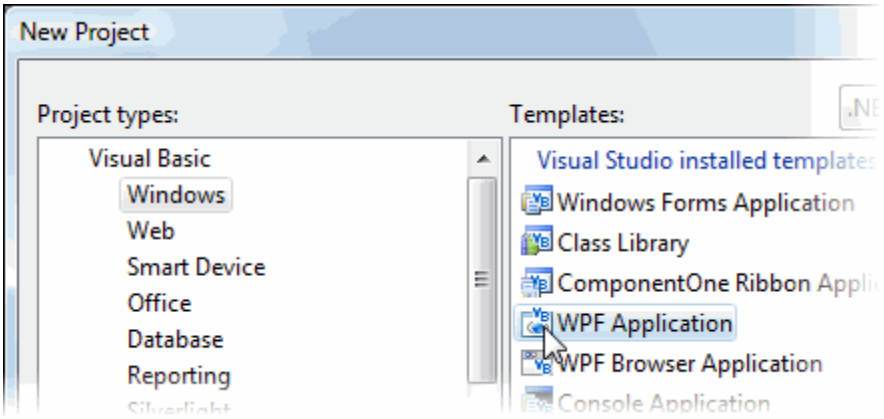
2. Choose the appropriate .NET Framework from the Framework drop-down box in the top-right of the dialog box.



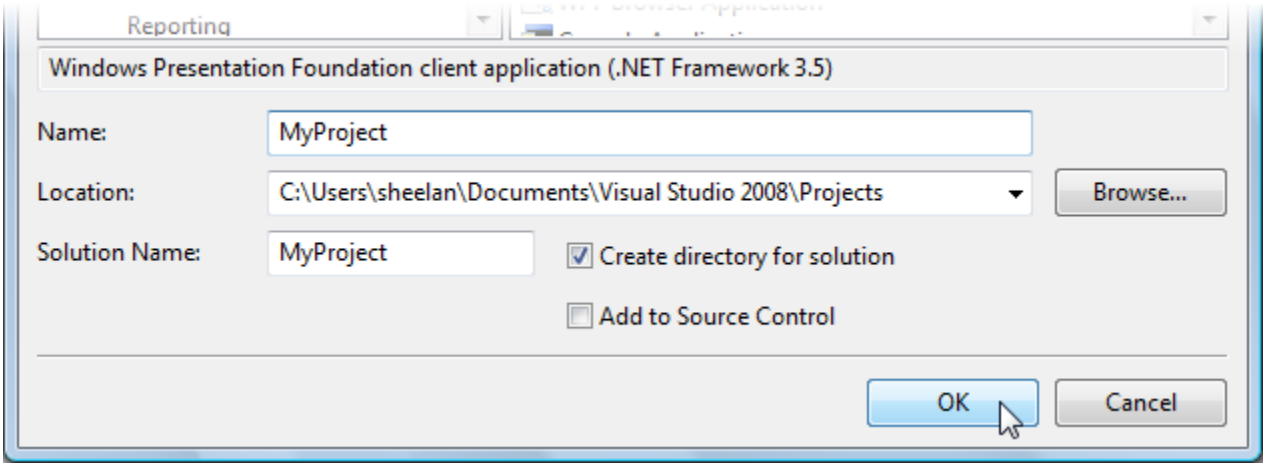

3. Under **Project types**, select either **Visual Basic** or **Visual C#**.

> **Note:** In Visual Studio 2005 select **NET Framework 3.0** under **Visual Basic** or **Visual C#** in the Project types menu.

4. Choose **WPF Application** from the list of **Templates** in the right pane.




5. Enter a name for your application in the **Name** field and click **OK**.



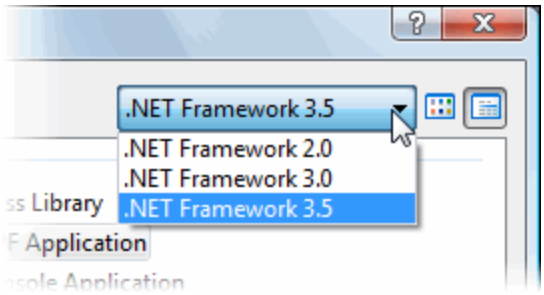

A new Microsoft Visual Studio .NET WPF project is created with a XAML file that will be used to define your user interface and commands in the application.

**Note:** You can create your grid applications using Microsoft Expression Blend or Visual Studio, but Blend is currently the only design-time environment that allows users to design XAML documents visually. In this documentation, Blend will be used for most examples.

# Creating an XAML Browser Application (XBAP) in Visual Studio

To create a new XAML Browser Application (XBAP) in Visual Studio 2008, complete the following steps:

1. From the **File** menu in Microsoft Visual Studio 2008, select **New Project**. The **New Project** dialog box opens.
2. Choose the appropriate .NET Framework from the Framework drop-down box in the top-right of the dialog box.




3. Under Project types, select either **Visual Basic** or **Visual C#**.
4. Choose **WPF Browser Application** from the list of **Templates** in the right pane.

> **Note:** If using Visual Studio 2005, you may need to select **XAML Browser Application (WPF)** after selecting **NET Framework 3.0** under **Visual Basic** or **Visual C#** in the left-side menu.

5. Enter a name for your application in the **Name** field and click **OK**.

   A new Microsoft Visual Studio .NET WPF Browser Application project is created with a XAML file that will be used to define your user interface and commands in the application.

# Adding the Grid for WPF Components to a Blend Project

In order to use C1DataGrid or another **ComponentOne Grid for WPF** component in the Design workspace of Blend, you must first add a reference to the **C1.WPF.C1DataGrid** assembly and then add the component from Blend's **Asset Library**.

**To add a reference to the assembly:**

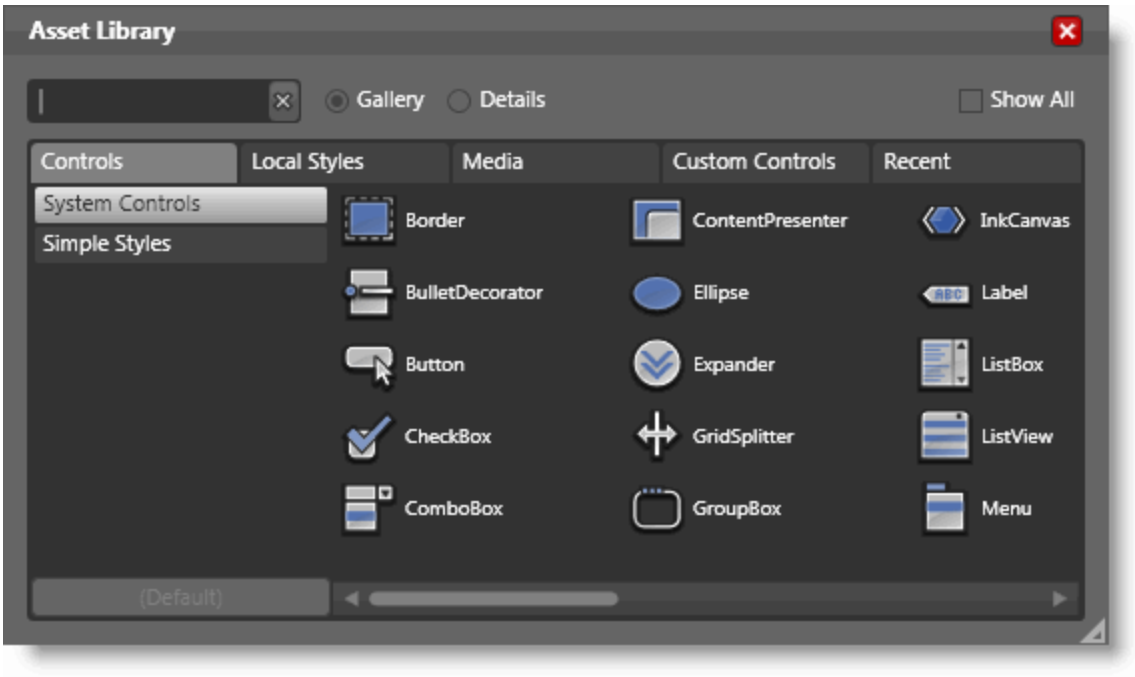
1. Select **Project | Add Reference**.
2. Browse to find the **C1.WPF.C1DataGrid.dll** assembly installed with **Grid for WPF**.

> **Note:** The **C1.WPF.C1DataGrid.dll** file is installed to **C:\Program Files\ComponentOne\Studio for WPF\Bin** by default.

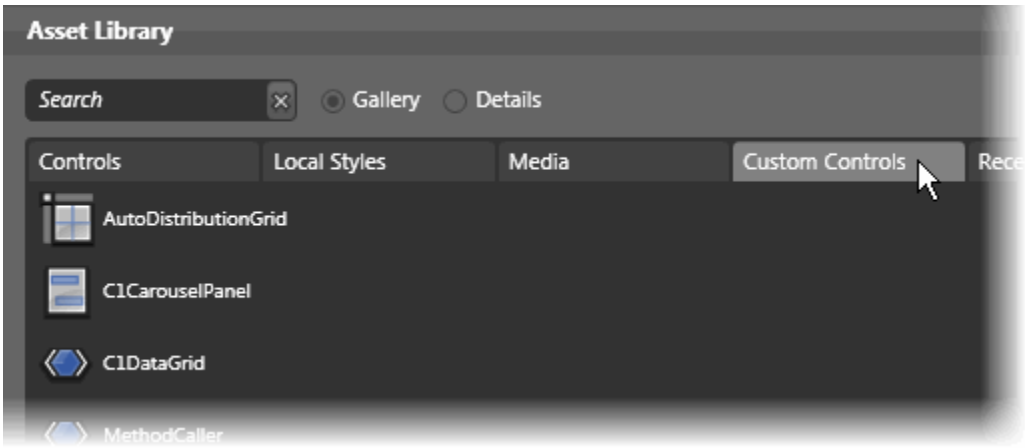
3. Select **C1.WPF.C1DataGrid.dll** and click **Open**. A reference is added to your project.

**To add a component from the Asset Library:**

1. Once you have added a reference to the **C1.WPF.C1DataGrid** assembly, click the **Asset Library** button  in the Blend Toolbox. The **Asset Library** appears:




2. Click the **Custom Controls** tab. All of the **Grid for WPF** main and auxiliary components are listed here.




3. Select **C1DataGrid**. The component will appear in the Toolbox above the **Asset Library** button.

4. Double-click the **C1DataGrid** component in the Toolbox to add it to **Window1.xaml**.

# Adding the Grid for WPF Components to a Visual Studio Project

When you install **ComponentOne Grid for WPF** the C1DataGrid control should be added to your Visual Studio Toolbox. You can also manually add ComponentOne controls to the Toolbox.

**ComponentOne Grid for WPF** provides the following control:

- C1DataGrid

**ComponentOne Grid for WPF** provides the following panel:

- C1CarouselPanel

To use a **Grid for WPF** panel or control, add it to the window or add a reference to the **C1.WPF.C1DataGrid** assembly to your project.

### Manually Adding Grid for WPF to the Toolbox

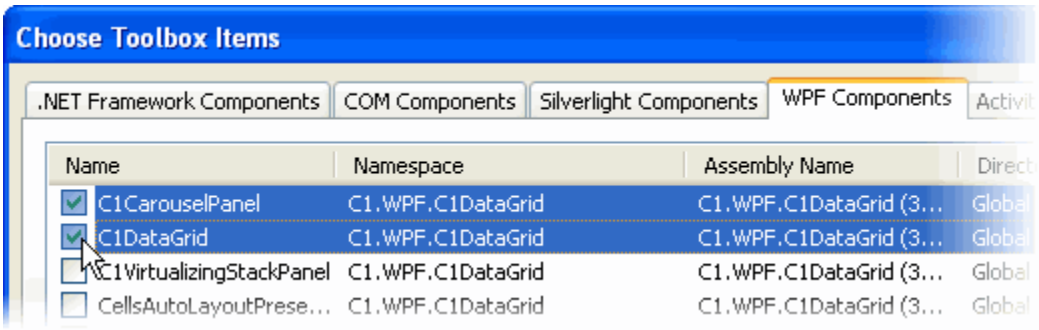
When you install **Grid for WPF**, the following **Grid for WPF** control and panel will appear in the Visual Studio Toolbox customization dialog box:

- C1DataGrid
- C1CarouselPanel

To manually add the C1DataGrid control to the Visual Studio Toolbox, complete the following steps:

1. Open the Visual Studio IDE (Microsoft Development Environment). Make sure the Toolbox is visible (select **Toolbox** in the **View** menu, if necessary) and right-click the Toolbox to open its context menu.
2. To make **Grid for WPF** components appear on its own tab in the Toolbox, select **Add Tab** from the context menu and type in the tab name, **C1WPFGrid**, for example.
3. Right-click the tab where the component is to appear and select **Choose Items** from the context menu.

   The **Choose Toolbox Items** dialog box opens.
4. In the dialog box, select the **WPF Components** tab.
5. Sort the list by Namespace (click the *Namespace* column header) and select the check boxes for components belonging to the **C1.WPF.C1DataGrid** namespace. Note that there may be more than one component for each namespace.




### Adding Grid for WPF to the Window

To add **ComponentOne Grid for WPF** to a window or page, complete the following steps:

1. Add the C1DataGrid control to the Visual Studio Toolbox.
2. Double-click C1DataGrid or drag the control onto the window.

### Adding a Reference to the Assembly

To add a reference to the **Grid for WPF** assembly, complete the following steps:

1. Select the **Add Reference** option from the **Project** menu of your project.
2. Select the **ComponentOne Grid for WPF** assembly from the list on the **.NET** tab or on the **Browse** tab, browse to find the **C1.WPF.C1DataGrid.dll** assembly and click **OK**.
3. Double-click the window caption area to open the code window. At the top of the file, add the following **Imports** statements (**using** in C#):

```
Imports C1.WPF.C1DataGrid
```

This makes the objects defined in the **Grid for WPF** assembly visible to the project.

# Key Features

**ComponentOne Grid for WPF** allows you to create customized, rich grid applications. Make the most of **Grid for WPF** by taking advantage of the following key features:

- **Multiple Data Views**

  Change the appearance of **Grid for WPF** with multiple built-in unique data views or create your own custom data view. For example, a customized carousel data view:


  Built-in data views include a:

  - Tabular view with vertical row distribution
  - Tabular view with horizontal row distribution
  - Card view
  - Carousel view

  You can also create your own views, fully customizing user interface elements. For example, customize the item, header, group by area, and item cells, and define an arbitrary item distribution panel, current item, sort indicators, and much more. For more information, see Using Data Views (page 84).

- **13 Built-In Themes for Easy Customization**

  Customize the appearance of your grid application with built-in Office 2007, Vista, and Office 2003 themes, or create your own theme based on the included themes. For more information about available themes and using themes in your grid, see the Using Themes (page 71) topic.

- **Automatic and Explicit Column Generation**

Automatically generate columns in the grid or explicitly define columns – you can even explicitly define columns in an automatically generated grid. Automatic column generation is based on source list item properties, including support for the ADO.NET DataSet or any **ITypedList** source. By default the AutoGenerateColumns property is set to **True** and all columns, even those undefined in the Columns collection, are generated automatically. If set to **False**, only those columns defined in the Columns collection are shown. For an example of explicitly defining columns in the grid, see Adding Columns to the Grid (page 54).

- **Universal Data Binding**

  Bind the C1DataGrid control to any object that implements the **System.Collections.IEnumerable** or **System.ComponentModel.IListSource** interface. For more information, see Binding Grid for WPF to a Data Source (page 55).

- **Universal Item Templates**

  Define a custom content layout for multiple grid parts (such as data item, header, and filter bar) in a single template with supported universal item templates. For more information, see Using the UniversalItemContentTemplate (page 160).

- **Virtual Grid Item Generation**

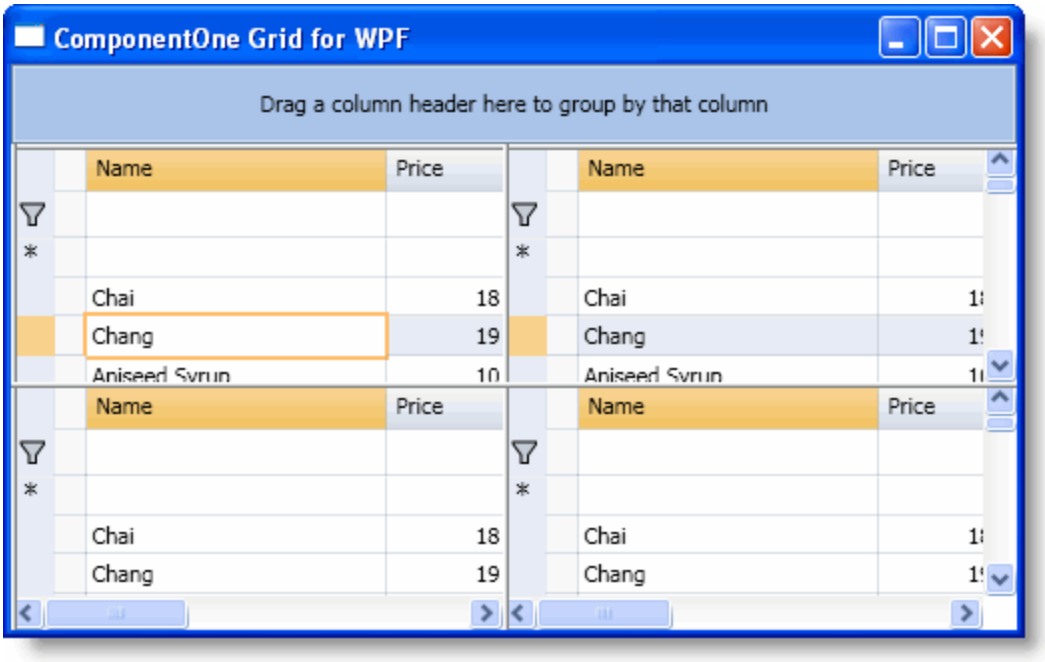
  **Grid for WPF** supports user interface virtualization – by processing only information loaded in the viewable area UI virtualization speeds up the user interface generation when working with a large source list.

- **Run-time Interaction**

  Easily group, sort, filter, resize, and reorder columns at run time all through simple drag-and-drop operations. For more information, see Run-Time Interaction (page 58).

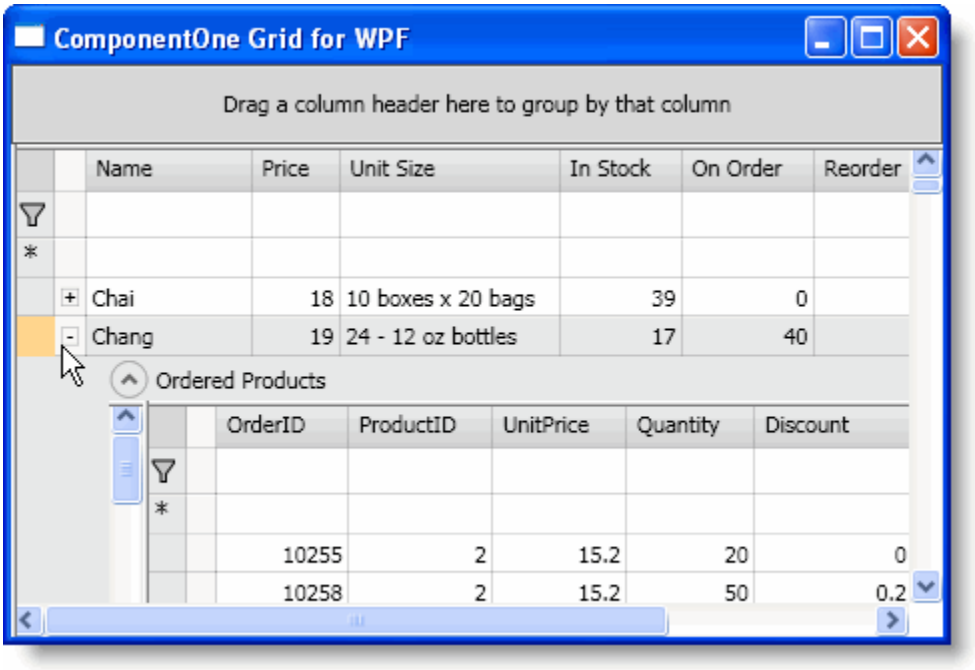
- **Split Creation**

  At run time users can easily create and resize Excel-like horizontal and vertical splits that you can customize at design time. For more information about using splits at run time see Creating Splits (page 66).



- **Hierarchical Data Representation**

  **Grid for WPF** supports automatic hierarchical data representation; so, for example, when the grid is bound to a table with a child table, users can expand the child table under the current table to see related data. See Viewing Child Tables (page 69) for more information.




- **Simplified Layout Customization**

  **Grid for WPF** incorporates simplified grid item and grid header cells layout customization, defining cell placement by specifying panel-specific attached properties on a column. This layout customization can be used in conjunction with **Grid** (**Column**, **Row**, **ColumnSpan**, and **RowSpan** properties), **Canvas** (**Left** and **Top** properties), and other panels that define child element placement via attached properties. See Simplified Cell Layout in a Panel (page 54) for more information.

- **Column Data Type to Cell Template Mapping**

  Define a cell show and edit template for a data type in a single place and set it so it affects all the columns of that type in a specific grid, all of the grids on a Page, or even all of the grids in an application. **Grid for WPF** can declaratively describe a mapping between a data type represented by a grid Column and an item cell template used to define the UI for cell viewing and editing, giving you more control over how data is displayed.

- **Unified Underlying Data Access Model**

  **Grid for WPF** incorporates a unified underlying data access model where an underlying data source is wrapped by a **Rows[i].Cells[j]** object model, which gives unified read-write access to a grid's source list data, without needing to consider data source specifics.

- **Auxiliary XAML Elements**

  Auxiliary XAML elements simplify the creation of an arbitrary user interface. See XAML Elements (page 111) for more information.

- **Unbound Grids and Columns**

  Use C1DataGrid with unbound data, even adding unbound columns to a bound grid. For an example of adding an unbound column to the grid, see Adding Columns to the Grid (page 54).

- **Automatic Totals Calculation**

  You can now add totaling to columns in your grid. Easily total columns with sum, average, minimum, maximum, and count functions, or create your own custom totals function. See the Calculating Column Totals (page 56) topic and the Totals property description for details.

- **Column Formatting**

  You can easily format a column as, for example, date, percent, currency, or so on using the Format property. For an example, see Formatting a Column as Currency (page 155). You can also format how columns appear depending on their DataType – for more information, see Using Type Based Column Styles (page 88).

# Grid for WPF Quick Start

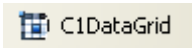
The following quick start guide is intended to get you up and running with **Grid for WPF**. In this quick start you'll start in Visual Studio and create a new project, add **Grid for WPF** to your application, and add a data source. You'll then move to Microsoft Expression Blend to complete binding the grid to the data source, customize the grid, and run the grid application to observe run-time interactions.

**Note:** This quick start guide uses the **C1NWind.mdb** database, installed by default in the **ComponentOne Samples\Common** folder installed in your **MyDocuments** folder (**Documents** in Vista). You could also use the standard Microsoft Northwind database instead, NWind.mdb, and adapt the appropriate steps.

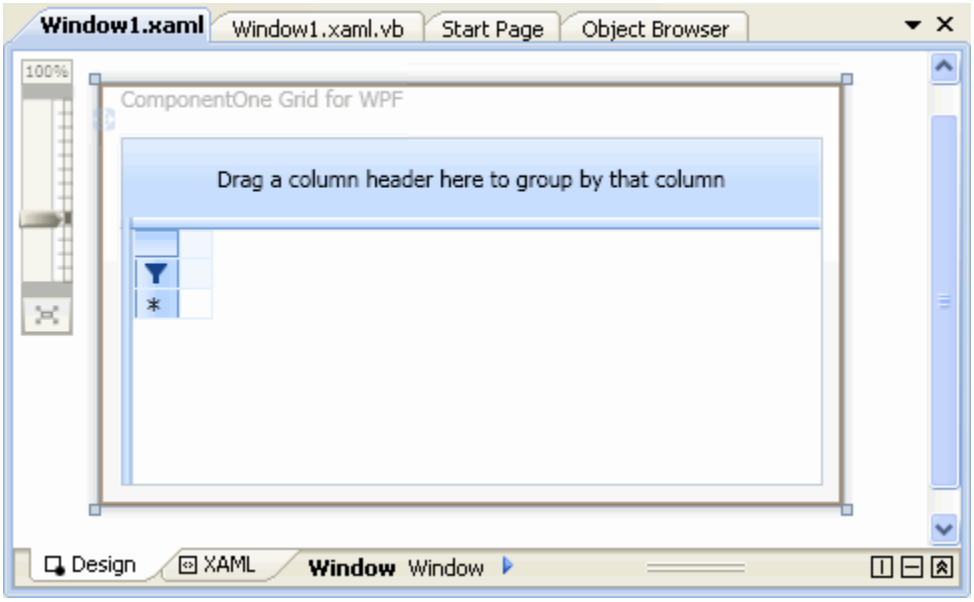
## Step 1 of 4: Adding Grid for WPF to your Project

In this step you'll begin in Visual Studio to create a grid application using **Grid for WPF**. When you add the C1DataGrid control to your application, you'll have a complete, functional grid. You can further customize the grid to your application.

To set up your project and add a C1DataGrid control to your application, complete the following steps:

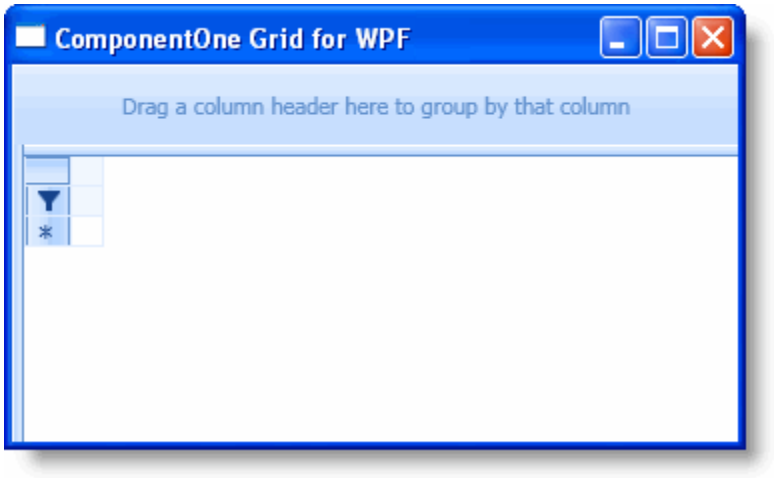
1. Create a new WPF project in Visual Studio. For more information about creating a WPF project, see Creating a .NET Project in Visual Studio (page 15).
2. Navigate to the Toolbox and double-click the **C1DataGrid** icon 

to add the grid control to Window1.
3. Resize the Window and the C1DataGrid within the Window; it should now look similar to the following:




**Run the program and observe:**

The grid application will appear similar to the following:



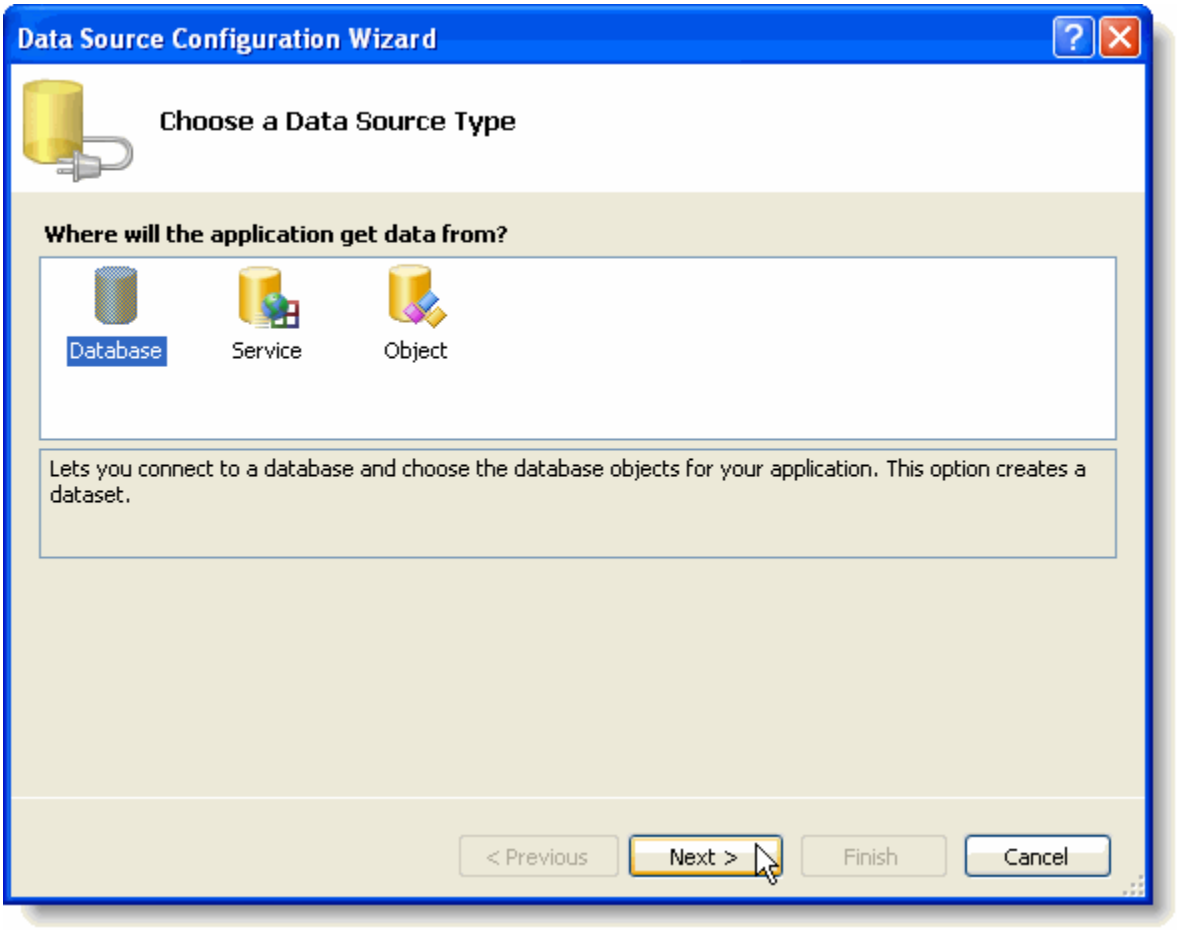

You've successfully created a grid application. In the next step you'll add a data source to your project and bind the grid to a data source.

## Step 2 of 4: Binding the Grid to a Data Source

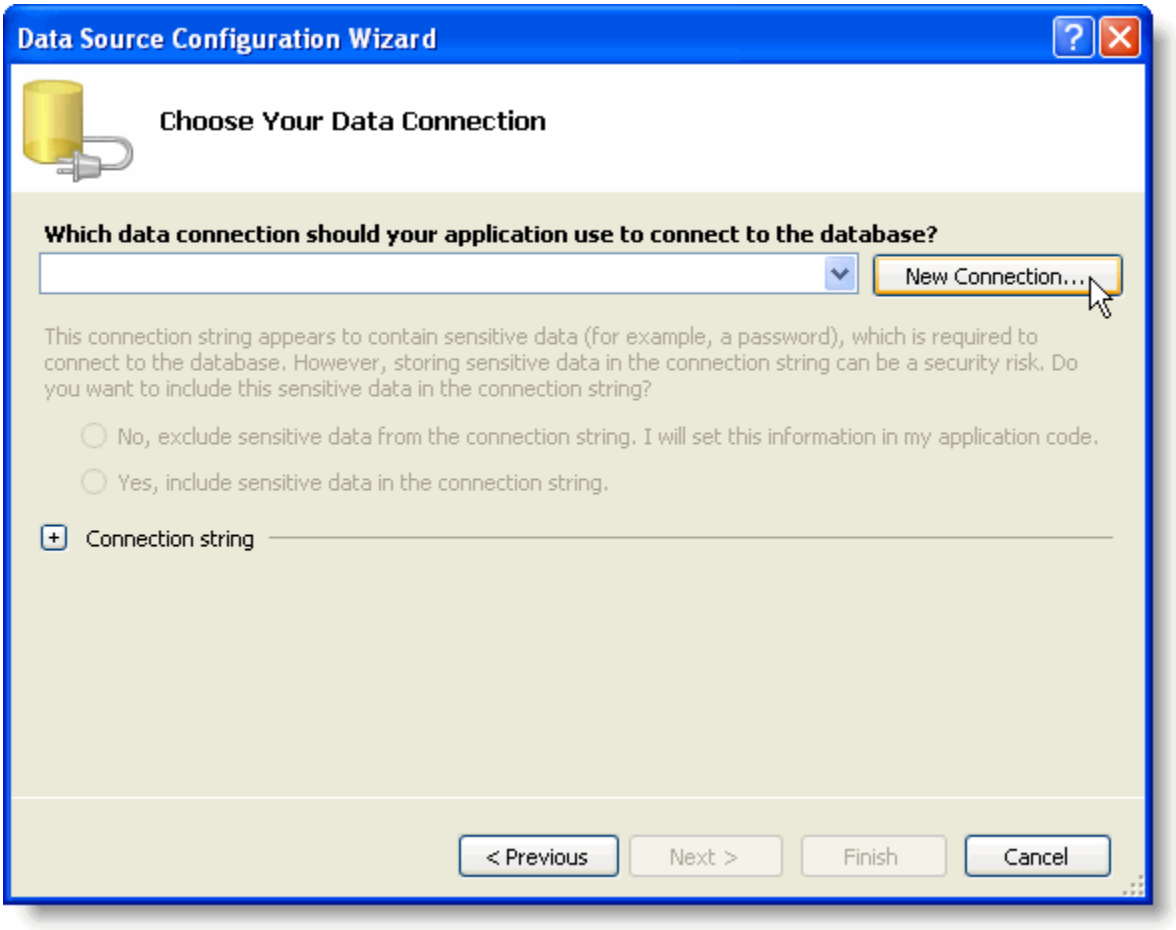
In the last step you set up the grid application – but while the basic grid is functional, it contains no data. In this step you'll continue in Visual Studio by adding a data source to your project. You'll then open the project in Microsoft Expression Blend to complete binding the grid to the data source.

**To add a data source and set up data binding in Visual Studio, complete the following steps:**

1. From the **Data** menu, select **Add New Data Source**. The **Data Source Configuration Wizard** appears.
2. Confirm that **Database** is selected and click **Next**.

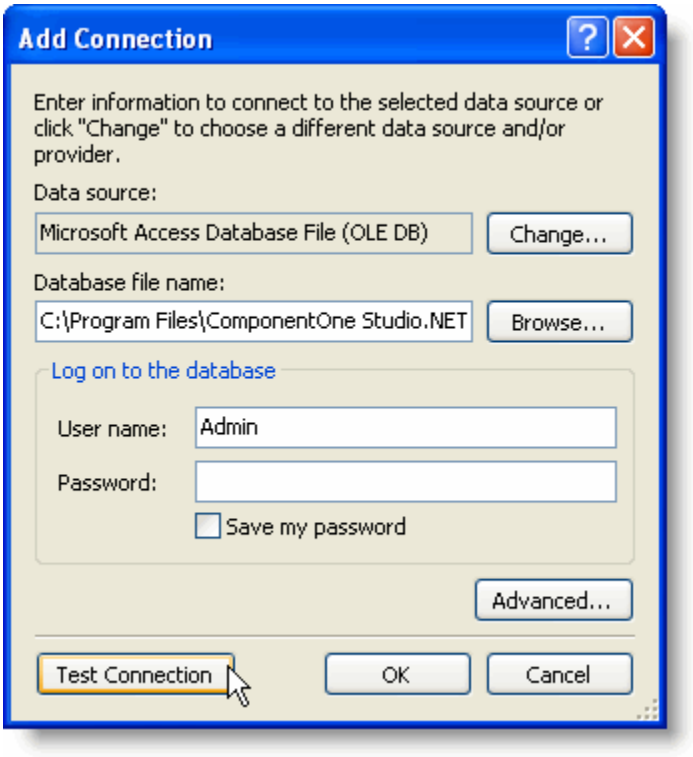



3. Click the **New Connection** button to locate and connect to a database.

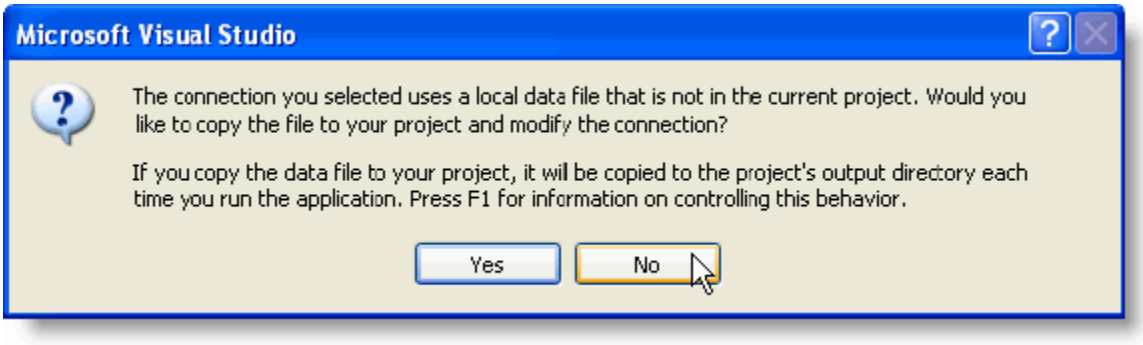



If the **Choose Data Source** dialog box appears, select **Microsoft Access Database File** and click **Continue**. The **Add Connection** dialog box will appear.

4. In the **Add Connection** dialog box, click the **Browse** button and locate **C1NWind.mdb** in the samples installation directory. Select it and click **Open**.
5. Click the **Test Connection** button to make sure that you have successfully connected to the database or server and click **OK**.

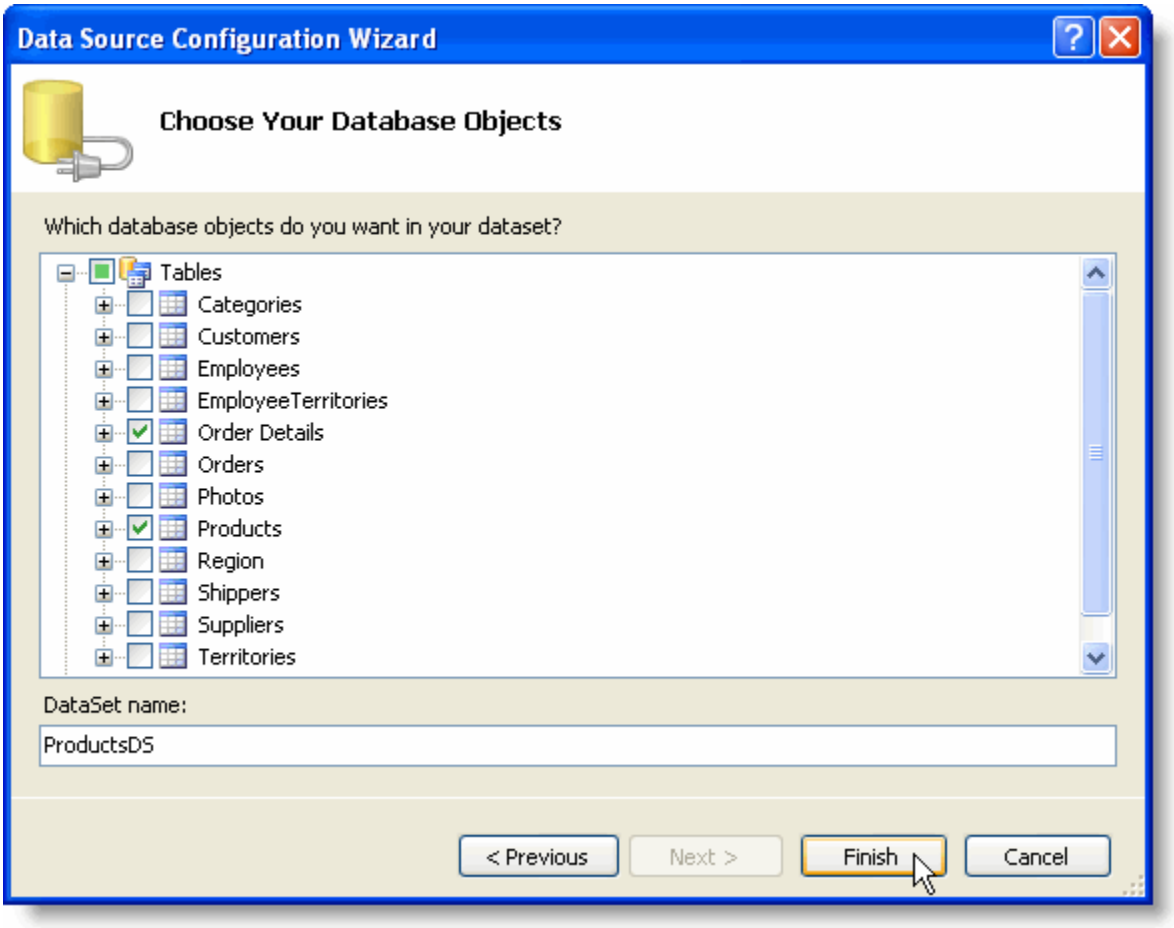



6. Click **OK** to close the **Add Connection** dialog box. The new string appears in the data connection drop-down list on the **Choose Your Data Connection** page.
7. Click the **Next** button to continue. A dialog box will appear asking if you would like to add the data file to your project and modify the connection string. Since it is not necessary to copy the database to your project, click **No**.

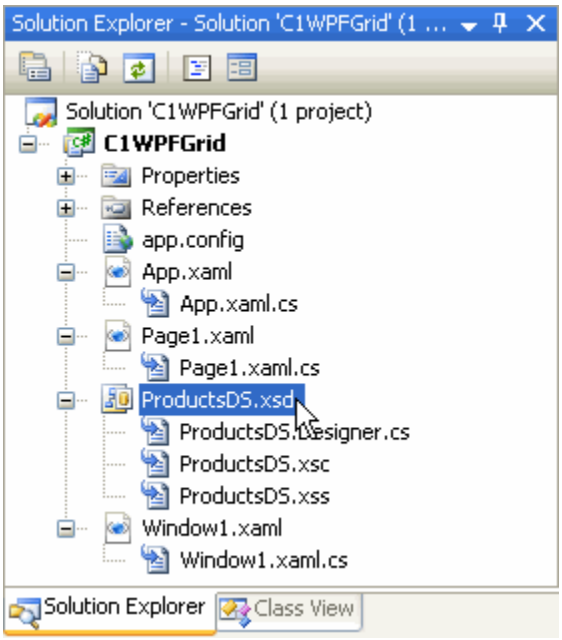



8. In the next window, confirm that the **Yes, save the connection as** checkbox is checked and a name has been automatically entered in the text box. Click **Next** to continue.


9. In the **Choose Your Database Objects** window, you can select the tables and fields that you would like in your dataset. Select the **Products** and **Order Details** tables and change the DataSet name to **ProductsDS**.




10. Click **Finish** to exit the wizard. The **ProductsDS.xsd** files now appear in the Solution Explorer.



11. In the Solution Explorer, double-click the **Window1.xaml.cs** (or Window1.xaml.vb) file to switch to code view.
12. Add the following references to the top of the Window1.xaml.cs (or Window1.xaml.vb) file, replacing *ProjectName* with the name of your project:

    - Visual Basic
    ```
    Imports C1.WPF.C1DataGrid
    Imports ProjectName.ProductsDSTableAdapters
    ```
    - C#
    ```
    using C1.WPF.C1DataGrid;
    using ProjectName.ProductsDSTableAdapters;
    ```
13. Add the following code to the Window1 class to retrieve the products and order details data from the database:

    - Visual Basic
    ```
    Class Window1
        Inherits Window
        Private _productsDataSet As ProductsDS = Nothing
        Public ReadOnly Property ProductsDataSet() As ProductsDS
            Get
                If _productsDataSet Is Nothing Then
                    _productsDataSet = New ProductsDS()
                    Dim prodTA As New ProductsTableAdapter()
                    prodTA.Fill(_productsDataSet.Products)
                    Dim ordDetTA As New Order_DetailsTableAdapter()
                    ordDetTA.Fill(_productsDataSet.Order_Details)
                End If
                Return _productsDataSet
            End Get
        End Property

        Public Sub New()
            InitializeComponent()
        End Sub
    End Class
    ```
    - C#
    ```
    public partial class Window1 : Window
    {
        private ProductsDS _productsDataSet = null;
        public ProductsDS ProductsDataSet
        {
            get
            {
                if (_productsDataSet == null)
                {
                    _productsDataSet = new ProductsDS();
                    ProductsTableAdapter prodTA = new
    ProductsTableAdapter();
                    prodTA.Fill(_productsDataSet.Products);
                    Order_DetailsTableAdapter ordDetTA = new
    Order_DetailsTableAdapter();
                    ordDetTA.Fill(_productsDataSet.Order_Details);
                }
                return _productsDataSet;
            }
    ```

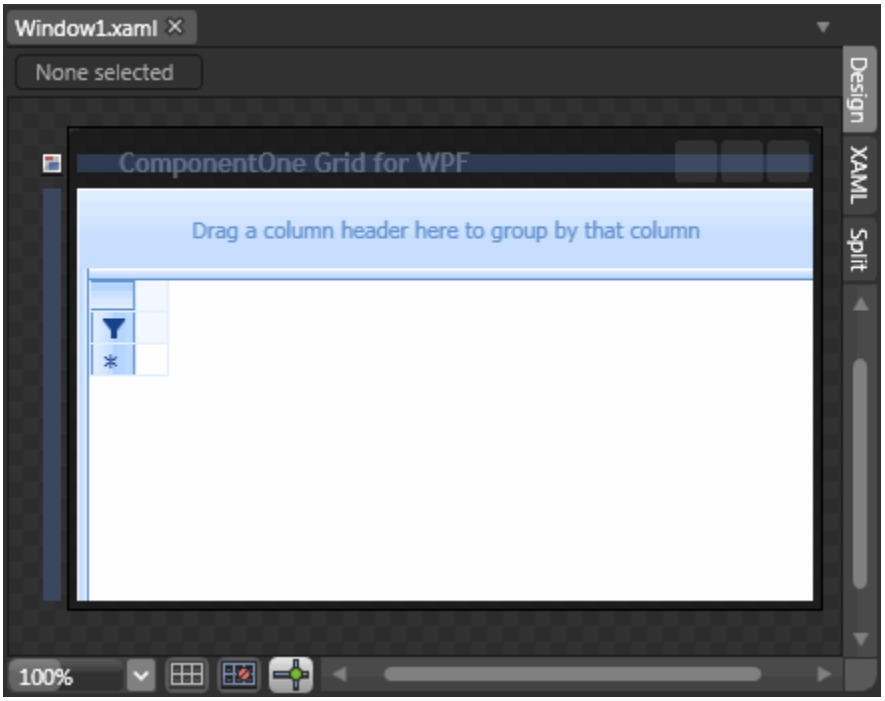
```
    }

    public Window1()
    {
        InitializeComponent();
    }
}
```

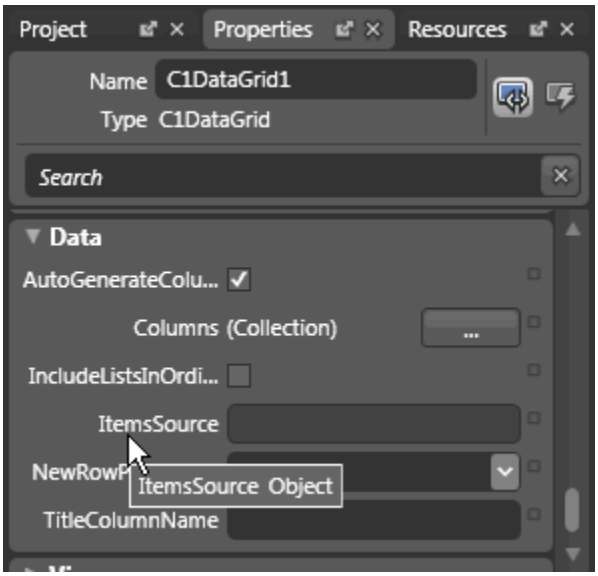
14. Run your project to ensure that everything is working correctly and then close your running application and save and close your project. You'll complete binding the grid to a data source in Microsoft Expression Blend.

**To bind the grid to a data source in Microsoft Expression Blend, complete the following steps:**

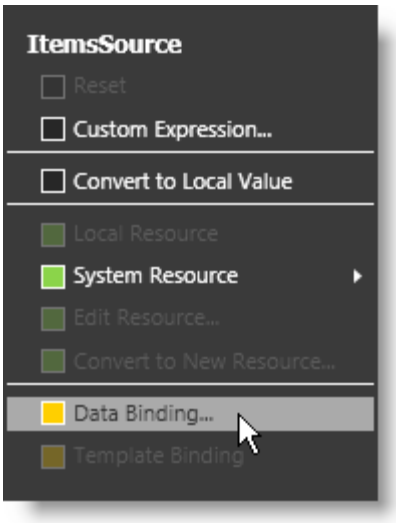
1. Open Microsoft Expression Blend and on the **File** menu click **Open** and then **Project/Solution**.
2. In the **Open Project** dialog box, locate your project file and click **Open**. The project you created in Visual Studio will open in Blend.




3. Click once on the **C1DataGrid** control so that it is selected, use the scroll bar to navigate to the lower part of the C1DataGrid1 Properties window, expand the **Data** group, and locate the ItemsSource property.



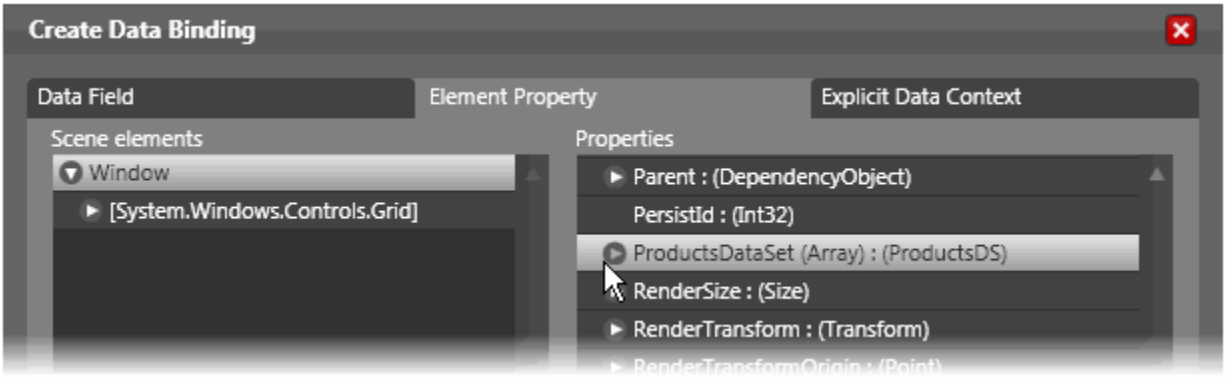

4. Click the square next to the ItemsSource property to access advanced property options and select **Data Binding** from the menu that appears.




5. In the **Create Data Binding** dialog box, select the **Element Property** tab.

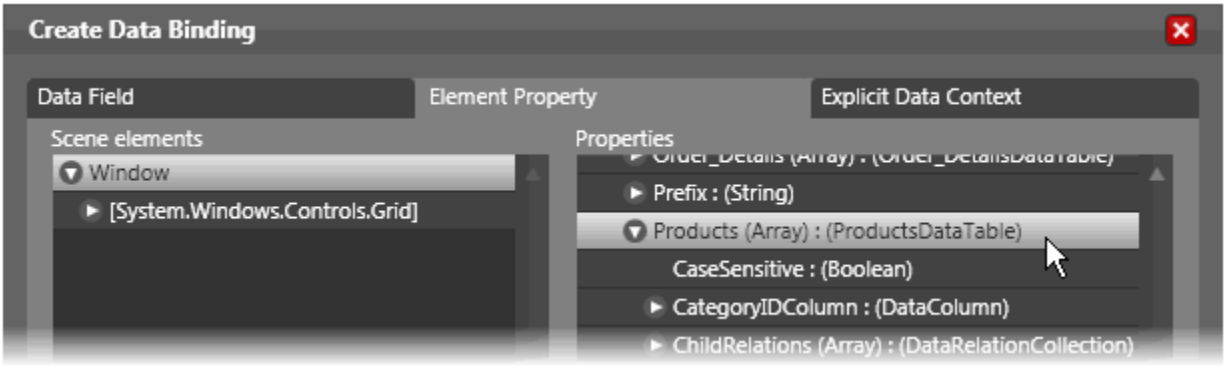

The **Element Property** tab appears with Scene elements in the left pane and Properties in the right pane.

6. In the right **Properties** pane of the Element Property tab, scroll down and click the arrow icon next to the **ProductsDataSet (Array) : (ProductsDS)** item to expand the available tables.




Note that this is the dataset that you added in Visual Studio.

7. Select **Products(Array) : (ProductsDataTable)** to bind the grid to the products table.



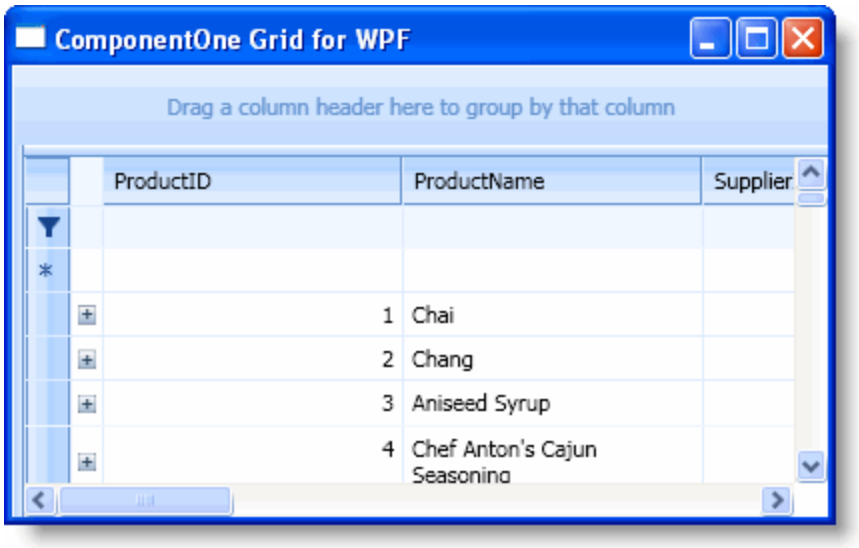

8. Click **Finish** to complete the data binding process and close the **Create Data Binding** dialog box.

   Notice in the XAML view, the C1DataGrid tag now appears as the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=ProductsDataSet.Products, ElementName=window, Mode=Default}" />
```

**Run the program and observe:**

The grid is now populated with data from the **Products** table. Notice the plus sign (+) next to each row – this expands the **Order_Details** table under each product in the **Products** table:




You've successfully bound **Grid for WPF**'s C1DataGrid control to a data source. In the next step you'll customize the look and feel of the C1DataGrid control in Blend.

# Step 3 of 4: Customizing the Grid in Blend

In the previous steps you worked in Visual Studio and Microsoft Expression Blend to create a new project and bind the **Grid for WPF** to a database. In this step you'll continue in Blend to customize the grid application's appearance. Your project should already be opened in Blend. You can customize the project using XAML or by using the C1DataGrid Properties window to change properties.

**To customize Grid for WPF in Blend using XAML, complete the following steps:**

1. Change the NewRowPlacement property to **FirstItem** so that new rows are added to the first line of the grid by adding `NewRowPlacement="FirstItem"` to the `<c1grid:C1DataGrid>` tag so that it looks like the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=ProductsDataSet.Products, ElementName=window, Mode=Default}"
NewRowPlacement="FirstItem">
```

2. Change the Theme property from the Office2007Default theme to the MediaPlayer theme by adding `Theme="{DynamicResource {c1grid:C1ThemeKey ThemeName=MediaPlayer, TypeInTargetAssembly={x:Type c1grid:C1DataGrid}}}"` to the `<c1grid:C1DataGrid>` tag so that it looks like the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=ProductsDataSet.Products, ElementName=window,
Mode=Default}"NewRowPlacement="FirstItem" Theme="{DynamicResource
{c1grid:C1ThemeKey ThemeName=MediaPlayer, TypeInTargetAssembly={x:Type
c1grid:C1DataGrid}}}">
```

3. Set the AutoGenerateColumns property to **False** by adding `AutoGenerateColumns="False"` to the `<c1grid:C1DataGrid>` tag so that it looks like the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=ProductsDataSet.Products, ElementName=window,
Mode=Default}"NewRowPlacement="FirstItem" Theme="{DynamicResource
{c1grid:C1ThemeKey ThemeName=MediaPlayer, TypeInTargetAssembly={x:Type
c1grid:C1DataGrid}}}" AutoGenerateColumns="False">
```

Because you'll customize which columns will appear in the grid, you don't want the columns to be automatically generated.

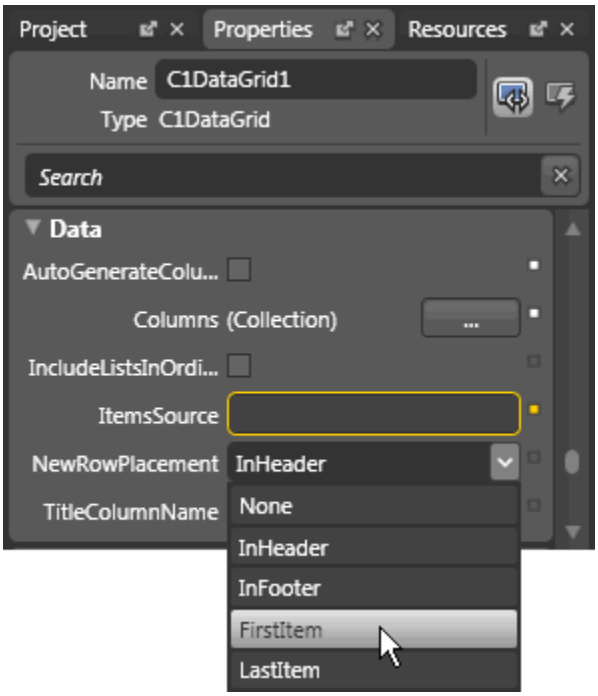
4. Add XAML (the <c1grid:C1DataGrid.Columns> tags) to the <c1grid:C1DataGrid> tags so that it looks like the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=ProductsDataSet.Products, ElementName=window, Mode=Default}"
NewRowPlacement="FirstItem" Theme="{DynamicResource {c1grid:C1ThemeKey
ThemeName=MediaPlayer, TypeInTargetAssembly={x:Type
c1grid:C1DataGrid}}}" AutoGenerateColumns="False">
   <c1grid:C1DataGrid.Columns>
         <c1grid:Column PropertyName="ProductName" Caption="Name"
HeaderCellWidth="150"/>
         <c1grid:Column PropertyName="QuantityPerUnit" Caption="Unit
Size" HeaderCellWidth="125"/>
         <c1grid:Column PropertyName="UnitPrice" Caption="Price"
HeaderCellWidth="60"/>
         <c1grid:Column PropertyName="UnitsInStock" Caption="In Stock"
HeaderCellWidth="75"/>
         <c1grid:Column PropertyName="UnitsOnOrder" Caption="On Order"
HeaderCellWidth="80"/>
         <c1grid:Column PropertyName="ReorderLevel" Caption="Reorder
Level" HeaderCellWidth="100"/>
         <c1grid:Column PropertyName="Discontinued"
Caption="Discontinued" HeaderCellWidth="95"/>
         <c1grid:Column PropertyName="ProductsOrder Details"
Caption="Ordered Products"/>
   </c1grid:C1DataGrid.Columns>
</c1grid:C1DataGrid>
```

   Only the columns specified above will appear in your grid now. Notice that the column tags specify the PropertyName, the Caption, and the HeaderCellWidth. The *ProductsOrder Details* column is an expandable child list that details order information from the **Order_Details** table.

**To customize Grid for WPF in Blend using the Properties window, complete the following steps:**

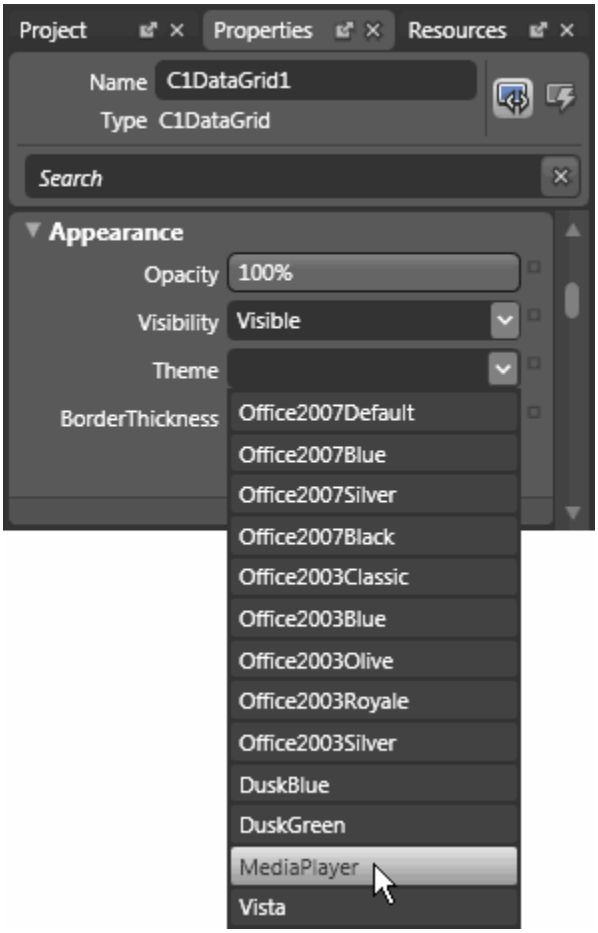
1. In the Properties window, locate the NewRowPlacement property in the **Data** tab and change its value to **FirstItem** so that new rows are added to the first line of the grid:




The XAML will now look similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=ProductsDataSet.Products, ElementName=window, Mode=Default}"
NewRowPlacement="FirstItem">
```

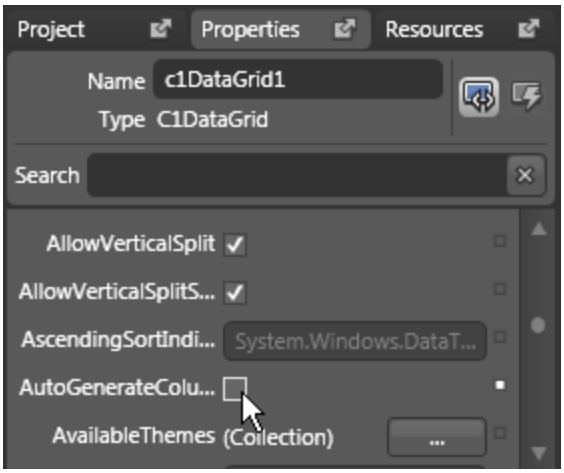
2. In the Properties window, locate the Theme property in the **Appearance** tab and change its value to **MediaPlayer**:




This will change the grid's theme from the default **Office2007Default** theme to the **MediaPlayer** theme. The XAML will now look similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=ProductsDataSet.Products, ElementName=window,
Mode=Default}"NewRowPlacement="FirstItem" Theme="{DynamicResource
{c1grid:C1ThemeKey ThemeName=MediaPlayer, TypeInTargetAssembly={x:Type
c1grid:C1DataGrid}}}">
```

3. Set the AutoGenerateColumns property to **False** by unchecking the box next to the AutoGenerateColumns property:



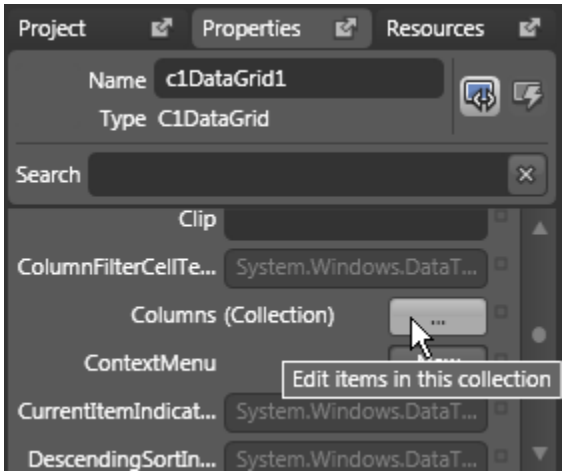

Because you'll customize which columns will appear in the grid, you don't want the columns to be automatically generated.

The XAML will now look similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=ProductsDataSet.Products, ElementName=window,
Mode=Default}"NewRowPlacement="FirstItem" Theme="{DynamicResource
{c1grid:C1ThemeKey ThemeName=MediaPlayer, TypeInTargetAssembly={x:Type
c1grid:C1DataGrid}}}" AutoGenerateColumns="False">
```

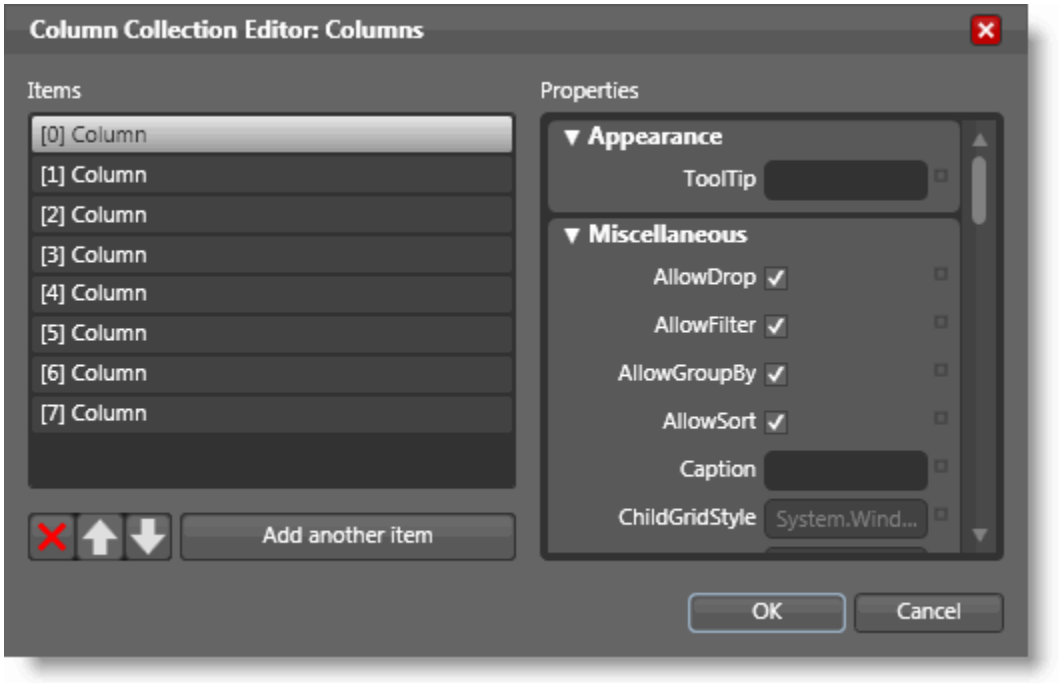
4. Next you'll add columns to the grid using the Column object. You'll specify which columns you want to appear in your grid.

    a. Click the **ellipsis** button next to the Column item.




    This will open the **Column Collection Editor** dialog box.

    b. In the **Column Collection Editor** dialog box, click the **Add another item button** eight times to add eight columns (numbered 0 to 7) to the collection.




c. Select each column on the left side of the **Column Collection Editor** dialog box and change each column's properties on the right side of the dialog box, as follows:

| Column | Property | Setting |
|---|---|---|
| **[0] Column** | PropertyName | ProductName |
| | Caption | Name |
| | HeaderCellWidth | 150 |
| **[1] Column** | PropertyName | QuantityPerUnit |
| | Caption | Unit Size |
| | HeaderCellWidth | 125 |
| **[2] Column** | PropertyName | UnitPrice |
| | Caption | Price |
| | HeaderCellWidth | 60 |
| **[3] Column** | PropertyName | UnitsInStock |
| | Caption | In Stock |
| | HeaderCellWidth | 75 |
| **[4] Column** | PropertyName | UnitsOnOrder |
| | Caption | On Order |
| | HeaderCellWidth | 80 |
| **[5] Column** | PropertyName | ReorderLevel |

| | | |
|---|---|---|
| | Caption | Reorder Level |
| | HeaderCellWidth | 100 |
| **[6] Column** | PropertyName | Discontinued |
| | Caption | Discontinued |
| | HeaderCellWidth | 95 |
| **[7] Column** | PropertyName | ProductsOrder Details |
| | Caption | Ordered Products |

d. Click **OK** to close the **Column Collection Editor** dialog box.

The XAML will now look similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=ProductsDataSet.Products, ElementName=window, Mode=Default}"
NewRowPlacement="FirstItem" Theme="{DynamicResource {c1grid:C1ThemeKey
ThemeName=MediaPlayer, TypeInTargetAssembly={x:Type
c1grid:C1DataGrid}}}" AutoGenerateColumns="False">
   <c1grid:C1DataGrid.Columns>
         <c1grid:Column PropertyName="ProductName" Caption="Name"
HeaderCellWidth="150"/>
         <c1grid:Column PropertyName="QuantityPerUnit" Caption="Unit
Size" HeaderCellWidth="125"/>
         <c1grid:Column PropertyName="UnitPrice" Caption="Price"
HeaderCellWidth="60"/>
         <c1grid:Column PropertyName="UnitsInStock" Caption="In Stock"
HeaderCellWidth="75"/>
         <c1grid:Column PropertyName="UnitsOnOrder" Caption="On Order"
HeaderCellWidth="80"/>
         <c1grid:Column PropertyName="ReorderLevel" Caption="Reorder
Level" HeaderCellWidth="100"/>
         <c1grid:Column PropertyName="Discontinued"
Caption="Discontinued" HeaderCellWidth="95"/>
         <c1grid:Column PropertyName="ProductsOrder Details"
Caption="Ordered Products"/>
   </c1grid:C1DataGrid.Columns>
</c1grid:C1DataGrid>
```

Only the columns specified above will appear in your grid now. Notice that the column tags specify the PropertyName, the Caption, and the HeaderCellWidth. The *ProductsOrder Details* column is an expandable child list that details order information from the **Order_Details** table.

**Run the program and observe:**

You've changed the appearance of the grid and the columns that are displayed:


You've successfully customized the appearance and behavior of your grid. In the next step you'll explore some of the run-time interactions that are possible in your grid application.

## Step 4 of 4: Running the Grid Application

Now that you've created a grid application, bound the grid to a database, and customized the grid's appearance in Blend, the only thing left to do is run your application. To run your grid application and observe **Grid for WPF**'s run-time behavior, complete the following steps:

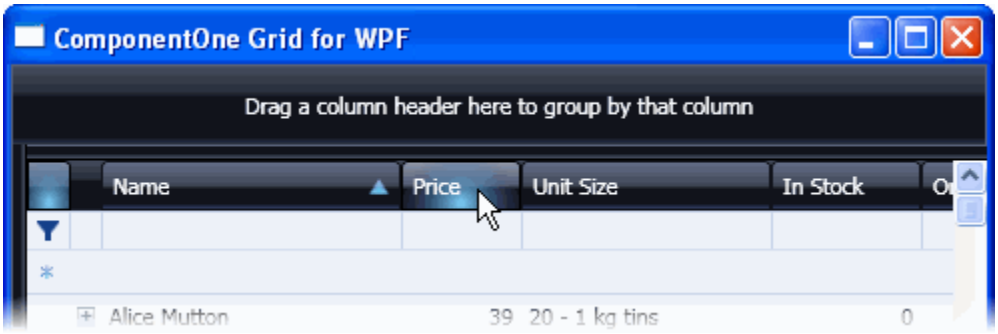
1. From the **Project** menu, select **Test Solution** to view how your grid application will appear at run time.
2. Click the **Name** header to sort the grid by product name. Notice that a sort indicator glyph appears to indicate the column being sorted and the direction of the sort.


You can sort multiple columns by holding the CTRL key while clicking another column.

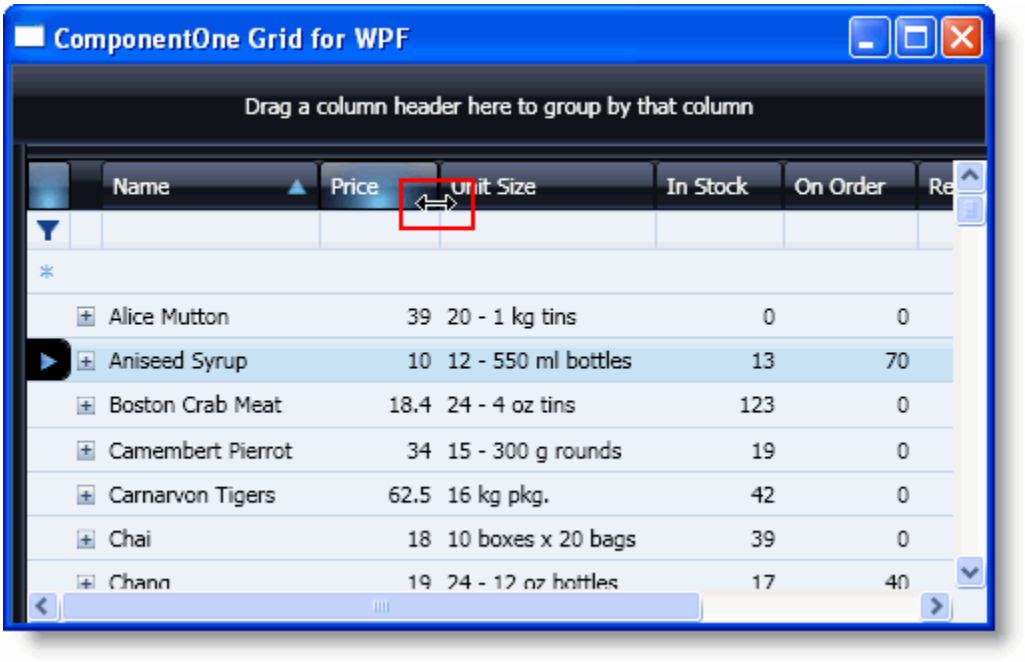
3. Click the plus button (+) next to a product name to expand the child **Order_Details** table for that product.


4. Collapse the child table by clicking the minus button (**-**) next to the expanded product name.
5. Re-order the columns by clicking the *Price* column header and dragging it in front of the *UnitSize* column header:

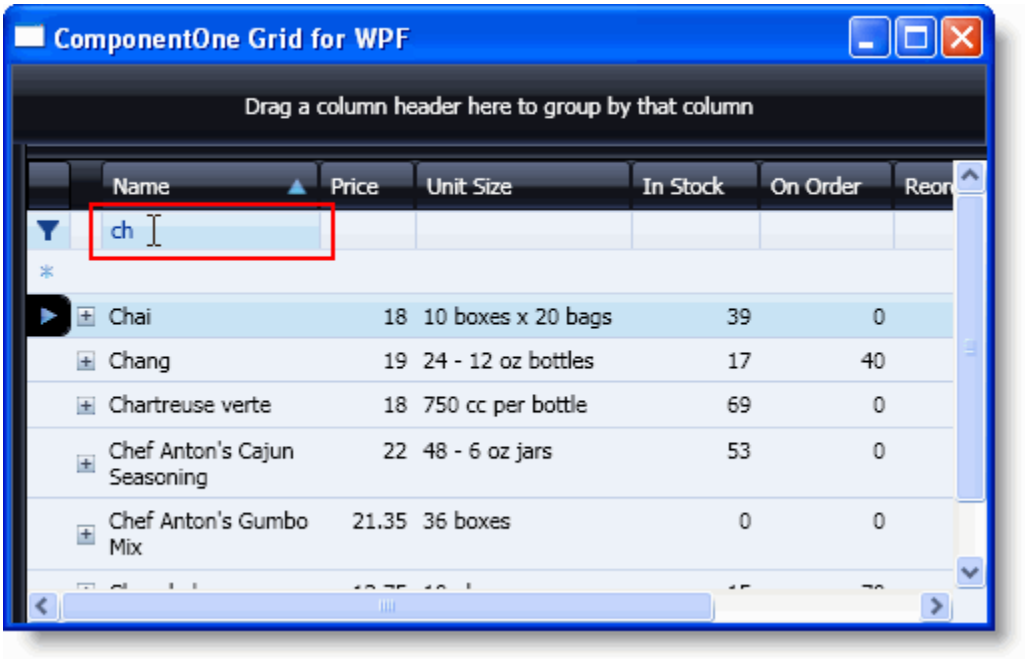

The *Price* column will now appear before the *Unit Size* column:



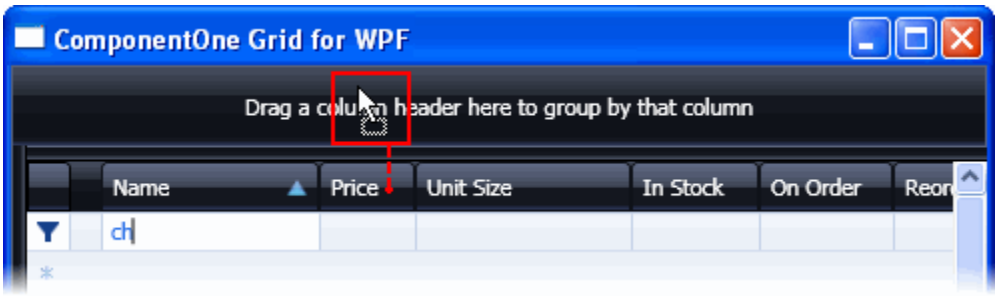

6. Resize a column, here the *Price* column, by clicking the right edge of the column and dragging the edge to a new location.



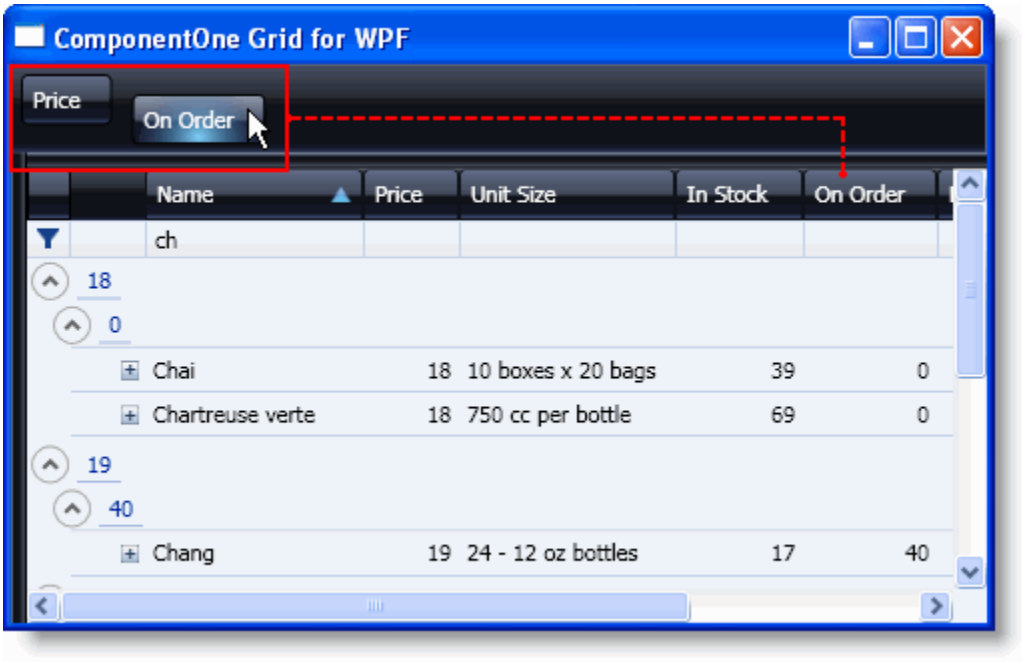

7. Filter the content of a column by entering "ch" in the filter bar in the *Name* column and pressing the ENTER key or clicking away from the column, so that only products beginning with that string appear:




8. Drag the **Price** header to the **Group By** area to group the grid by item price:

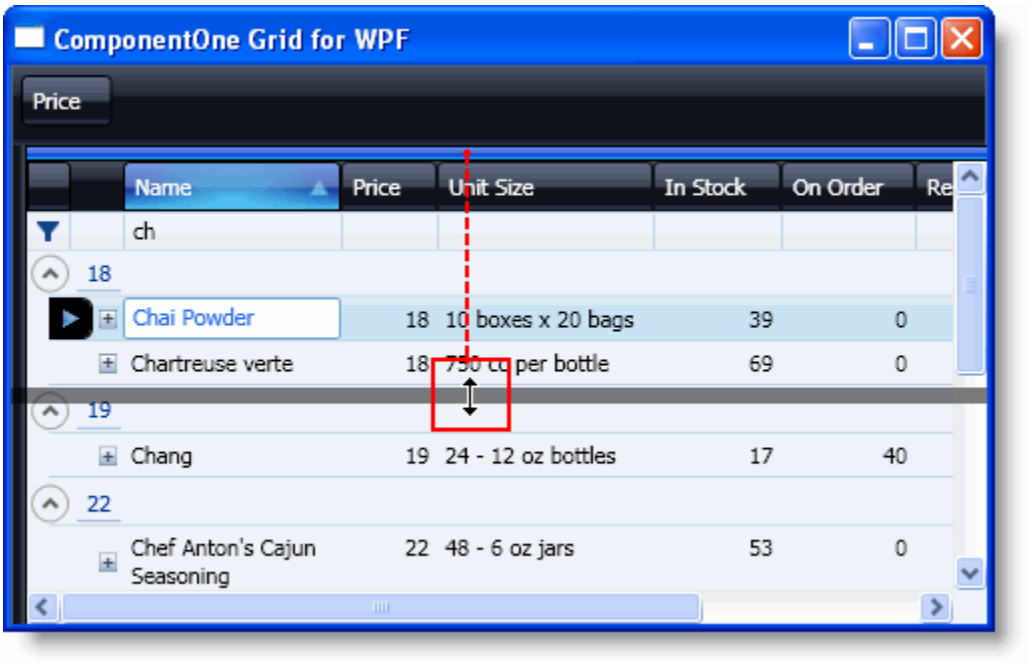



9. Repeat this with the On Order column to group by multiple criteria:

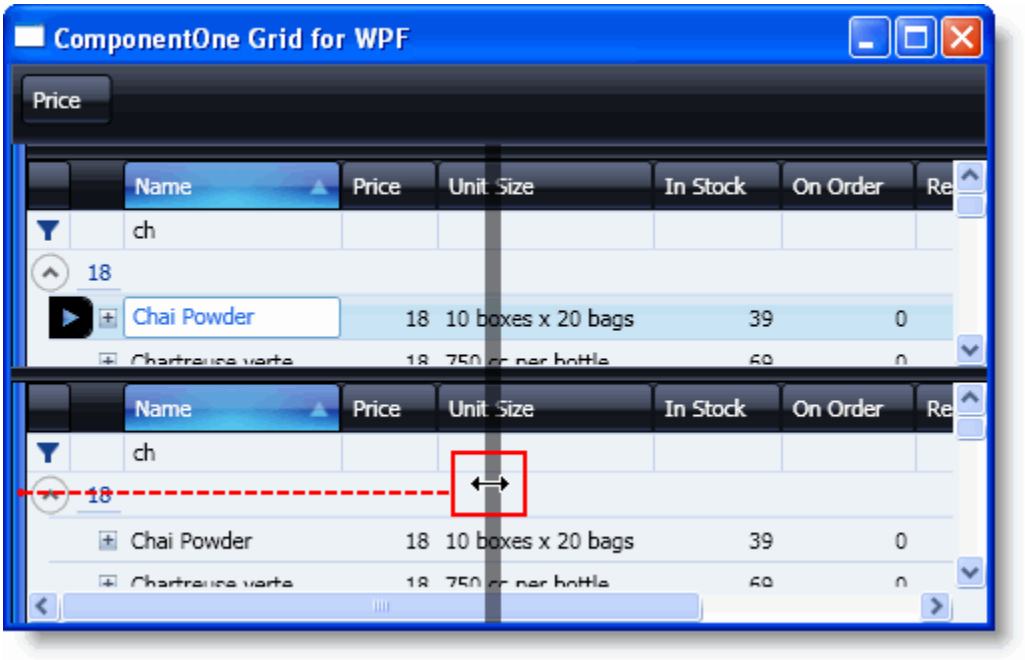



10. Click the *On Order* column header in the Group By area and drag it back onto the grid to no longer group columns by items on order.
11. Click once on a cell to edit the contents of that cell, and press the ENTER key.


12. Click and drag the top splitter bar to create two separate horizontal splits in the document.




13. Click and drag the left splitter bar to create two vertical separate splits in the document.




The grid should now look similar to the following:


Congratulations! You've completed the **Grid for WPF** quick start and created a **Grid for WPF** grid application, bound the grid to a data source, customized the appearance of the grid, and viewed some of the run-time capabilities of your grid application.

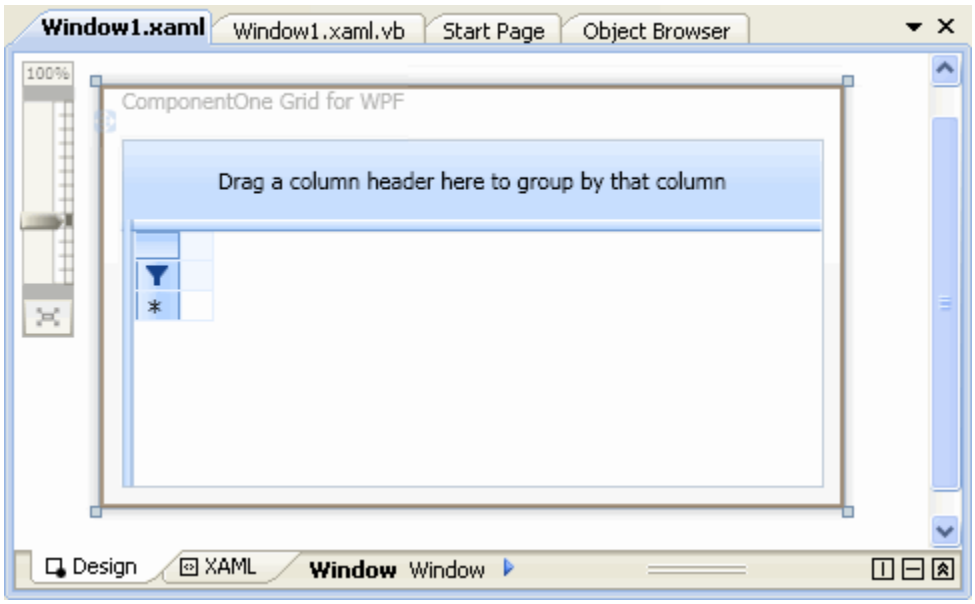
# Using Grid for WPF

**ComponentOne Grid for WPF** consists of the C1DataGrid control, which provides all of the functionality for a grid application and the C1CarouselPanel panel which allows you to add controls, such as Microsoft **ListBox** controls, to a carousel view. The C1DataGrid control is a fully functional grid that allows users to add, edit, and manage their data.

## Using the C1DataGrid Control

**ComponentOne Grid for WPF** includes the C1DataGrid control, which provides a flexible, customizable grid. When you add the C1DataGrid control to a XAML window, it exists as a completely functional grid that allows end users to add, edit and manage data. If you choose to, you can drop the control onto your window, bind the grid, and be done. But while the initial grid is quite flexible, it uses a default interface. You can further customize this interface to fit your users' needs by using themes (page 71), data views (page 84), and by customizing the grid's appearance and behavior with templates (page 108).

The default user interface looks like the following image in **Design** view in Visual Studio 2008:




## Using the C1CarouselPanel Panel

In addition to the C1DataGrid grid control, **ComponentOne Grid for WPF** includes the C1CarouselPanel panel. The **C1CarouselPanel** is a panel allowing you to view content and controls – including standard or 3rd-party controls – in a carousel. The **C1CarouselPanel** is used in the carousel data view (page 84) of the **C1DataGrid** control. Items in the panel revolve on a path when interacted with at run time.

To use the **C1CarouselPanel** to create a carousel-like interactive effect, you can set it as an **ItemsControl** control's panel and assign your visual elements collection to the **ItemsControl.Items** property. So, for example, in the XAML below an ItemsPanelTemplate template is defined in the Window's resources and includes the **C1CarouselPanel** panel. An **ItemsControl** containing arbitrary elements later points to the ItemsPanelTemplate template:

```
<Window x:Class="Window1"
xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:c1grid=http://schemas.componentone.com/wpf/C1DataGrid
Title="Window1" Height="231" Width="375">
    <Window.Resources>
        <!-- An ItemsPanelTemplate template defining the C1CarouselPanel.
-->
        <ItemsPanelTemplate x:Key="carouselPanel">
            <c1grid:C1CarouselPanel />
        </ItemsPanelTemplate>
    </Window.Resources>
    <Grid>
<!-- An ItemsControl with ItemsPanel set to the ItemsPanelTemplate
defining a C1CarouselPanel. -->
<ItemsControl ItemsPanel="{StaticResource carouselPanel}">
<!-- Arbitrary controls or images within the ItemsControl. -->
            <Image Width="51" Height="51" Source="image1.png"/>
            <Image Width="51" Height="51" Source="image2.png"/>
            <Image Width="51" Height="51" Source="image3.png"/>
            <Button Height="23" Name="Button1" Width="75">Button</Button>
</ItemsControl>
    </Grid>
</Window>
```

## Customizing the Carousel

You can customize the **C1CarouselPanel** in the example in the Using the C1CarouselPanel Panel (page 51) topic by setting properties in either the `<c1grid:C1CarouselPanel/>` tag in the ItemsPanelTemplate or in the `<ItemsControl>` tag. Setting properties in the `<ItemsControl>` tag is possible because all of the properties introduced in carousel are attached dependency properties, providing the ability to change carousel's property at run time.

### In the C1CarouselPanel Tag

So, for example, to limit the number of visible elements in the C1CarouselPanel, you can set the PageSize property in the `<c1grid:C1CarouselPanel>` tag:

```
<Window.Resources>
     <!-- An ItemsPanelTemplate template defining the C1CarouselPanel. -->
    <ItemsPanelTemplate x:Key="carouselPanel">
        <!-- Limit the number of visible elements to 3 with the PageSize
property. -->
        <c1grid:C1CarouselPanel PageSize="3" />
    </ItemsPanelTemplate>
</Window.Resources>
```

### In the ItemsControl Tag

To limit the number of visible elements in the C1CarouselPanel, you can also set the PageSize property in the in the `<ItemsControl>` tag:

```
<!-- An ItemsControl with ItemsPanel set to the ItemsPanelTemplate
defining a C1CarouselPanel. The PageSize property limits the number of
visible elements to 3. -->
<ItemsControl ItemsPanel="{StaticResource carouselPanel}"
c1grid:C1CarouselPanel.PageSize="3">
<!-- Arbitrary controls or images within the ItemsControl. -->
            <Image Width="51" Height="51" Source="image1.png"/>
            <Image Width="51" Height="51" Source="image2.png"/>
```

```
            <Image Width="51" Height="51" Source="image3.png"/>
            <Button Height="23" Name="Button1" Width="75">Button</Button>
</ItemsControl>
```

# Using the C1CollectionView Class

The C1CollectionView class was added to **ComponentOne Grid for WPF** in the 2010 v1 release. This class can be used as a replacement for native **CollectionView**, **ListCollectionView** and **BindingListCollectionView** classes and provides performance gains for grouping and sorting operations.

### Performance Benefits

Using the C1CollectionView class rather than the **CollectionView**, **ListCollectionView** and **BindingListCollectionView** classes can provide the following performance benefits:

- **Improved Local Sorting**

  Local sorting performed by the view (this is analogous of sorting operations performed by the Microsoft **ListCollectionView**) completes 5 times faster.

- **Improved Grouping**

  Grouping completes up to ten times as fast as Microsoft's **ListCollectionView or BindingListCollectionView**. Particularly when in a performance-intensive scenario, such as if there is one group per item and a huge number of groups, C1CollectionView improves performance significantly.

### Functional Benefits

There are also some functional benefits regarding these operations in C1CollectionView.

As you may know there are three different types of possible data sources: **IEnumerable** (no indexed access to items), **IList** (with indexed access to items) and **IBindingList**/**IBindingListView** (**IList** with filtering and sorting capabilities, the main implementer of which is the ADO.NET DataTable). Microsoft has separate view class for each of the mentioned types of source: **CollectionView**, **ListCollectionView** and **BindingListCollectionView**, respectively. Sorting and filtering operations can be potentially implemented by the view (local sort/filtering) or by data source (in case of **IBindingList**).

C1CollectionView gathers all this functionality in a single class (three-in-one), it accepts any **IEnumerable**, recognizes more specific interfaces supported by the source (**IList** or **IBindingList**/**IBindingListView**) and utilizes the additional capabilities offered by these interfaces. The table below summarizes the data processing capabilities provided by Micosoft and ComponentOne views:

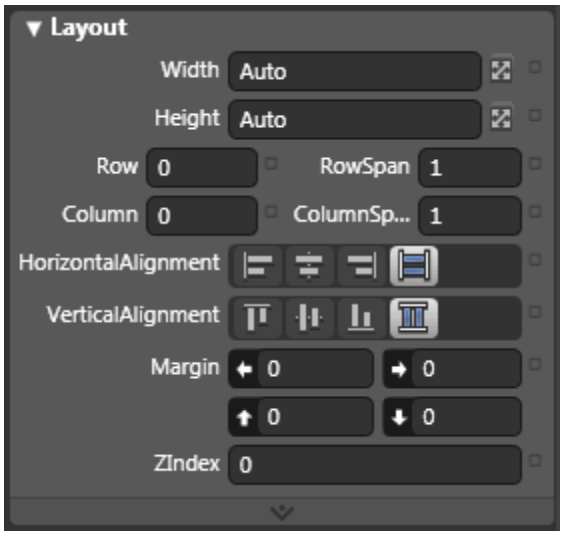
| | **CollectionView (IEnumerable)** | **ListCollectionView (IList)** | **BindingListCollectionView (IBindingList/ IBindingListView)** | **C1CollectionView (any IEnumerable)** |
|---|---|---|---|---|
| **Grouping** | - | + | + | + |
| **Local Sorting** | - | + | - | + |
| **Local Filtering** | + | + | - | + |
| **Source Sorting** | not applicable | not applicable | + | + for IBindingList |
| **Source Filtering** | not applicable | not applicable | + | + for IBindingListView |

As you see, C1CollectionView supports sorting and grouping operations for "just **IEnumerable**" (non-**IList**), which is not provided by MS **CollectionView**, and supports local sorting and filtering (in addition to data source sorting/filtering) for **IBindingList**/**IBindingListView**, as opposite to **BindingListCollectionView** that supports a processing by data source only.

# Simplified Cell Layout in a Panel

**ComponentOne Grid for WPF**'s C1DataGrid control provides a simplified way to lay out item cells in Panels which define placement of their child elements via attached properties. For example, in a **Grid** panel with its **Row**, **ColumnSpan**, and **RowSpan** properties and in a **Canvas** panel with its **Left** and **Top** properties. To define a cell placement, just assign the necessary panel's attached properties to a **Column** object that the cell represents: to lay out cells by means of the **Grid** panel, you can just assign the **Grid** panel with necessary number of column and rows to the CellsAutoLayoutPanelTemplate property, and define the cell placement in the panel by assigning Grid's **Row**, **Column**, **RowSpan**, and **ColumnSpan** attached properties to the **Column** objects from the Columns collection.

For example, the C1DataGrid control includes the following **Layout** properties when located within a **Grid** panel:




You can change the sizing, alignment, and location of the C1DataGrid within the **Grid** panel.

# Adding Columns to the Grid

You can easily add columns, even unbound columns, to a C1DataGrid grid control. To add columns to your grid, use the Columns property to specify and modify columns. For example, the following XAML adds the bound *ProductName* and *UnitPrice* columns to the C1DataGrid control, as well as an unbound Boolean *Organic* column:

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:sys="clr-namespace:System;assembly=mscorlib"
    x:Class="Window1"
    Title="Window1" Height="300" Width="300" xmlns:c1grid="clr-
namespace:C1.WPF.C1DataGrid;assembly=C1.WPF.C1DataGrid">
    <Grid>
```

```
      <c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=ProductsDataSet.Products, ElementName=Window, Mode=Default}">
            <c1grid:C1DataGrid.Columns>
                  <c1grid:Column Caption="Product Name"
PropertyName="ProductName"/>
                  <c1grid:Column Caption="Unit Price"
PropertyName="UnitPrice"/>
<c1grid:Column Caption="Organic" ColumnName="Organic" DataType="{x:Type
sys:Boolean}"/>
            </c1grid:C1DataGrid.Columns>
      </c1grid:C1DataGrid>
     </Grid>
</Window>
```

**Note:** You must add a `xmlns:sys="clr-namespace:System;assembly=mscorlib"` namespace declaration to the `<Window>` tag for the Boolean DataType declaration above to work correctly.

# Binding Grid for WPF to a Data Source

**ComponentOne Grid for WPF**'s C1DataGrid control can be bound to any object that implements the **System.Collections.IEnumerable** interface (such as **XmlDataProvider**, **ObjectDataProvider**, **DataSet**, **DataView**, and so on). Objects that implement the **System.ComponentModel.IListSource** interface are supported as well, for example the **System.Data.DataTable** class instances. You can use the ItemsSource property to bind the C1DataGrid.

For example, the following XAML binds the grid to the *Products* table in the **ProductsDataSet**; the latter is accessible via the **ProductsDataSet** property of the root Window containing the grid:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=ProductsDataSet.Products, ElementName=Window, Mode=Default}"/>
```

The ItemsSource property can be set to any object that implements the **System.Collections.IEnumerable** or **System.ComponentModel.IListSource** interfaces.

For steps on binding a C1DataGrid control to an access database using Visual Studio 2008 and Expression Blend, see Binding Grid for WPF to a Database (page 138).

# Setting Up Hierarchical Data Views

**ComponentOne Grid for WPF** supports automatic hierarchical data representation; so, for example, when the grid is bound to a table with a child table, users can expand the child table under the current table to see related data. See Viewing Child Tables (page 69) for information about interacting with child grids.

In hierarchical data views, the item type of the parent collection should contain a public property returning a child collection. For the grid to see this property and generate a corresponding Column automatically, the parent collection, such as the parent **ObservableCollection** must have a typed indexer, such as **ObservableCollection<MyParentItem>** instead of **ObservableCollection<object>**. If, for some reason, you must use **ObservableCollection<object>**, then a Column representing the child collection property must be added to the Columns collection explicitly.

Child tables will automatically be generated if the AutoGenerateColumns property is set to **True**. If AutoGenerateColumns is set to **False**, you can explicitly define a child grid using the Columns property. Simply define a Column with the **PropertyName** property set to a name of a data source item's property representing a child collection. If dealing with an ADO.NET DataSet, this should be the name of the **DataRelation** joining the parent and the child tables. For example, the following XAML defines the *Order Details* table in the Products Data Set:

```
<c1grid:C1DataGrid Name="c1DataGrid1" ItemsSource="{Binding
Path=ProductsDataSet.Products, ElementName=window, Mode=Default}"
AutoGenerateColumns="False">
   <c1grid:C1DataGrid.Columns>
          <c1grid:Column PropertyName="ProductsOrder Details"
Caption="Ordered Products"/>
   </c1grid:C1DataGrid.Columns>
</c1grid:C1DataGrid>
```

You can hide child tables by setting the AllowHierarchicalData property to **False**. For an example, see Hiding Child Tables (page 152).

You can customize a child grid by assigning a **Style** defining child grid properties to the ChildGridStyle property. For example, for child grid without a filter bar and with only two columns, the column definition in the above example should be changed to the following:

```
<c1grid:Column PropertyName="ProductsOrder Details" Caption="Ordered
Products"/>
      <c1grid:Column.ChildGridStyle>
                  <Style TargetType="c1grid:C1DataGrid">
                        <Setter Property="AutoGenerateColumns"
Value="False"/>
                        <Setter Property="Columns">
                              <Setter.Value>
                                    <c1grid:ColumnCollection>
                                          <c1grid:Column
PropertyName="UnitPrice" Caption="Price"/>
                                          <c1grid:Column
PropertyName="Quantity"/>
                                    </c1grid:ColumnCollection>
                              </Setter.Value>
                        </Setter>
                        <Setter Property="FilterBarVisibility"
Value="Collapsed"/>
                  </Style>
      </c1grid:Column.ChildGridStyle>
</c1grid:Column>
```

You may also define a **Style** that will be applied to all child grids of the parent grid by assigning the **Style** to the ChildGridStyle property of the parent grid.

# Calculating Column Totals

From the 2009 v1 release, **ComponentOne Grid for WPF** supports total calculation in the grid. To use totals, set the total bar to be visible and use the Totals property to set up totals calculation.

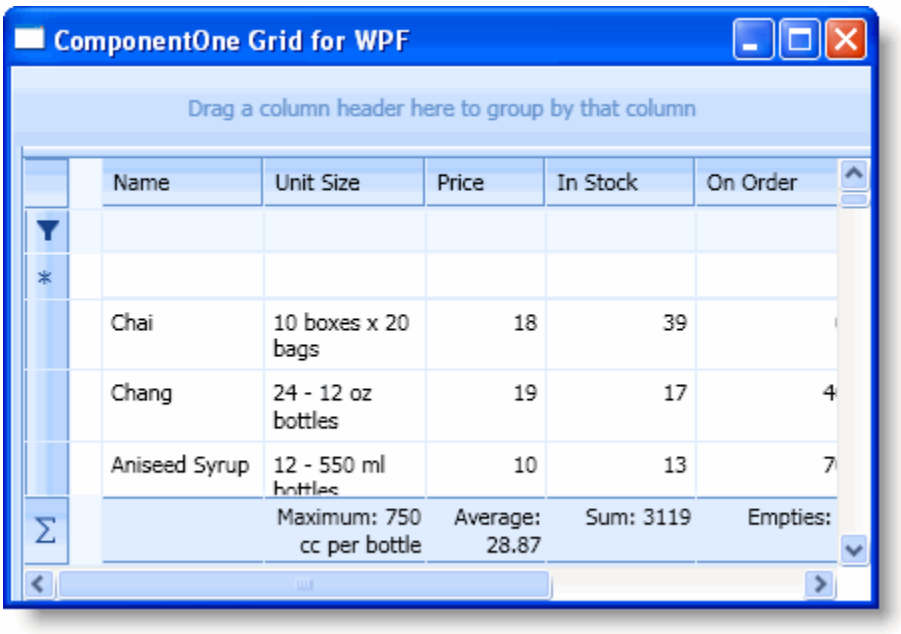
For the total bar to be visible, the TotalBarVisibility and/or GroupTotalBarVisibility property must be set to **Visible**. See Setting Total Bar Visibility (page 101) or Setting Group Total Visibility (page 103) for more information.

To set up totals calculation, include the totals function within the column's definition. You can set up the column definition to calculate the average, count, sum, minimum, or maximum value in a column. In the following example, column definitions have been set up to calculate different totals in several columns:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=ProductsDataSet.Products, ElementName=window, Mode=Default}"
AutoGenerateColumns="False" TotalBarVisibility="Visible">
    <c1grid:C1DataGrid.Columns>
        <c1grid:Column PropertyName="ProductName" Caption="Name"
HeaderCellWidth="150"/>
```

```
        <c1grid:Column PropertyName="QuantityPerUnit" Caption="Unit Size"
HeaderCellWidth="125">
            <c1grid:Column.Totals>
                <c1grid:Total Function="{x:Static
c1grid:MaxFunction.Default}" Format="{}{0:0}"/>
            </c1grid:Column.Totals>
        </c1grid:Column>
        <c1grid:Column PropertyName="UnitPrice" Caption="Price"
HeaderCellWidth="60">
            <c1grid:Column.Totals>
                <c1grid:Total Function="{x:Static
c1grid:AverageFunction.AverageAll}" Format="{}{0:0.00}"/>
            </c1grid:Column.Totals>
        </c1grid:Column>
        <c1grid:Column PropertyName="UnitsInStock" Caption="In Stock"
HeaderCellWidth="75">
            <c1grid:Column.Totals>
                <c1grid:Total Function="{x:Static
c1grid:SumFunction.Default}" Format="{}{0:0}"/>
            </c1grid:Column.Totals>
        </c1grid:Column>
        <c1grid:Column PropertyName="UnitsOnOrder" Caption="On Order"
HeaderCellWidth="80">
            <c1grid:Column.Totals>
                <c1grid:Total Function="{x:Static
c1grid:CountFunction.CountNulls}" Format="{}{0:0}"/>
            </c1grid:Column.Totals>
        </c1grid:Column>
        <c1grid:Column PropertyName="ReorderLevel" Caption="Reorder Level"
HeaderCellWidth="100">
            <c1grid:Column.Totals>
                <c1grid:Total Function="{x:Static
c1grid:AverageFunction.AverageSkipNulls}" Format="{}{0:0.00}"/>
            </c1grid:Column.Totals>
        </c1grid:Column>
        <c1grid:Column PropertyName="Discontinued" Caption="Discontinued"
HeaderCellWidth="95">
            <c1grid:Column.Totals>
                <c1grid:Total Function="{x:Static
c1grid:MinFunction.Default}" Format="{}{0:0}"/>
            </c1grid:Column.Totals>
        </c1grid:Column>
    </c1grid:C1DataGrid.Columns>
</c1grid:C1DataGrid>
```

The grid will appear similar to the following:




Note that if you do not wish to use one of the built-in functions, you can define your own custom function. For more information about custom functions, see the TotalFunction class description.

## Committing Data to a Data Source

From the 2009 v1 release of **ComponentOne Grid for WPF**, you can control when cell values updated by the end user are committed to the underlying data source. You can choose if grid and column items are updated immediately or on cell end edit using the C1DataGrid.CommitItemCellValue, Column.CommitItemCellValue, and ActualCommitItemCellValue properties.

By default, CommitItemCellValue is set to **OnEndEdit** and any changes are committed to the underlying data source when a user completes editing in a cell, for example when clicking away from the cell or pressing the ENTER key. You can set CommitItemCellValue to **Immediate** if you want the data source to be updated after the user makes a change but before leaving edit mode.

For more information about data input is handled by **Grid for WPF**, see Inputting Data (page 131).

## Exporting Data

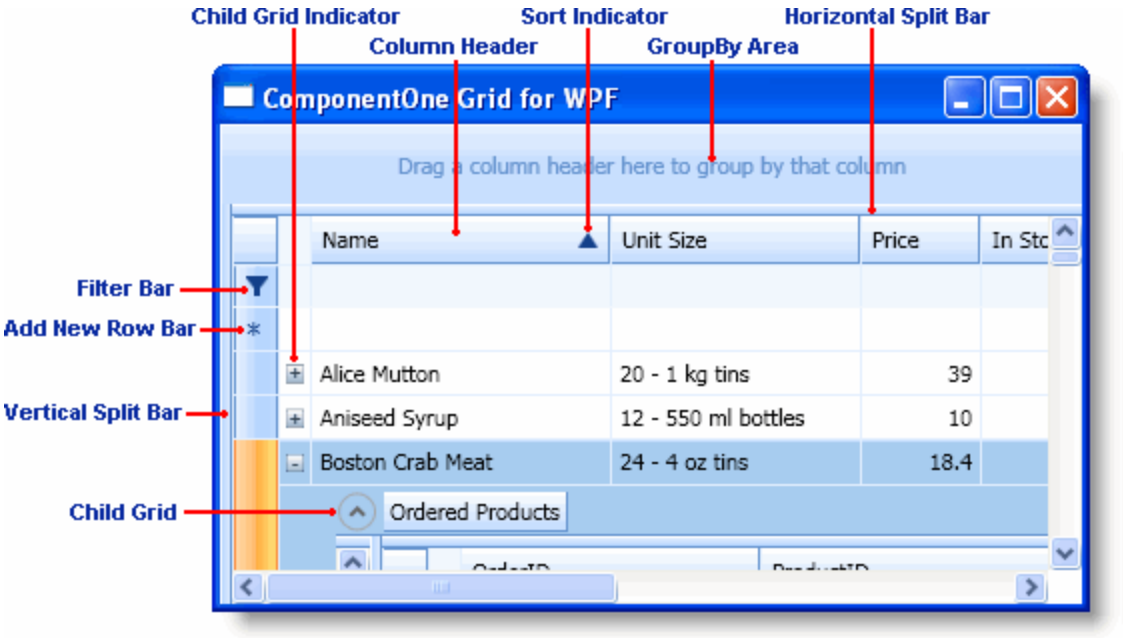
Starting with the 2009 v3 release of **ComponentOne Grid for WPF**, you can export data from a C1DataGrid control to a Microsoft Excel file using the ExportToExcel method. Using the ExportToExcel methods, you can export data to a file or stream. The FileFormat and IncludeColumns properties let you further customize how the file is saved.

Using the FileFormat property you can specify if the file should be saved as a Microsoft Excel 97/2003 file or as a Microsoft Excel 2007 file. The IncludeColumns property lets you specify the specific column names to export, if you do not wish to export all columns. If no columns are specified, all are exported.

For an example, see Exporting Data to an Excel File (page 168).

# Run-Time Interaction

The image below highlights some of the run-time interactions possible in a **ComponentOne Grid for WPF** C1DataGrid. The following topics detail these run-time features including filtering, sorting, and grouping data.




Note that in the images in the following topics the GroupByVisibility and the FilterBarVisibility may be set to **Collapsed** and the NewRowPlacement property may be set to **None** to better illustrate the topic under discussion.

# Keyboard Navigation

**ComponentOne Grid for WPF** supports run-time keyboard navigation. The following table lists several keyboard shortcuts that can be used to navigate and manipulate the grid at run time:

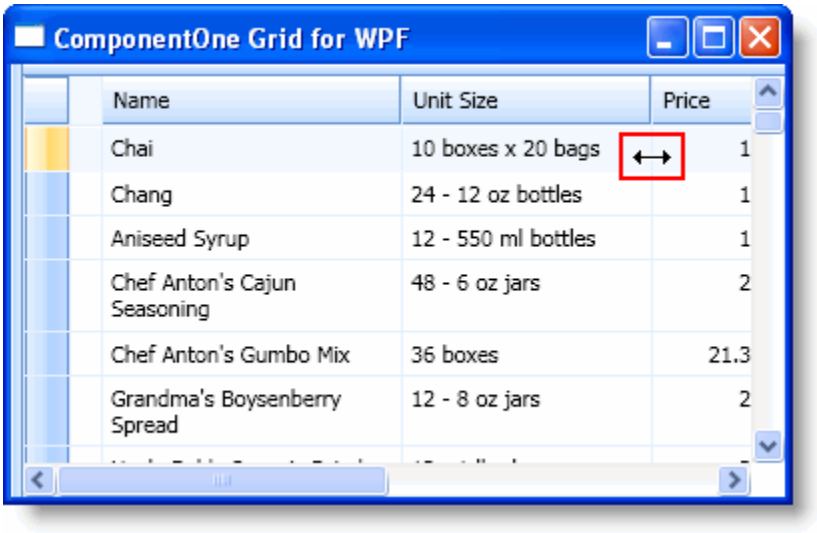
| Key Combination | Description |
|---|---|
| Left Arrow | Select previous cell/row (depending on the grid view). |
| Right Arrow | Select next cell/row (depending on the grid view). |
| Up Arrow | Select previous cell/row (depending on the grid view). |
| Down Arrow | Select next cell/row (depending on the grid view). |
| PAGE UP | Scroll the page up. |
| PAGE DOWN | Scroll the page down. |
| HOME | Select first cell in a selected row. |
| END | Select last cell in a selected row. |
| CTRL+Left Arrow | Select first cell/row (depending on the grid view). |
| CTRL+Right Arrow | Select last cell/row (depending on the grid view). |
| CTRL+Up Arrow | Select first row/cell (depending on the grid view). |
| CTRL+Down Arrow | Select last row/cell (depending on the grid view). |

| | |
|---|---|
| ENTER | Enter/exit edit mode on a selected cell; apply a filter value entered in FilterBar cell. |
| F2 | Enter edit mode on a selected cell. |
| ESC | Cancel editing of a cell or row. |
| TAB | Select next cell. |
| SHIFT+TAB | Select previous cell. |
| CTRL+D | Delete selected row. |

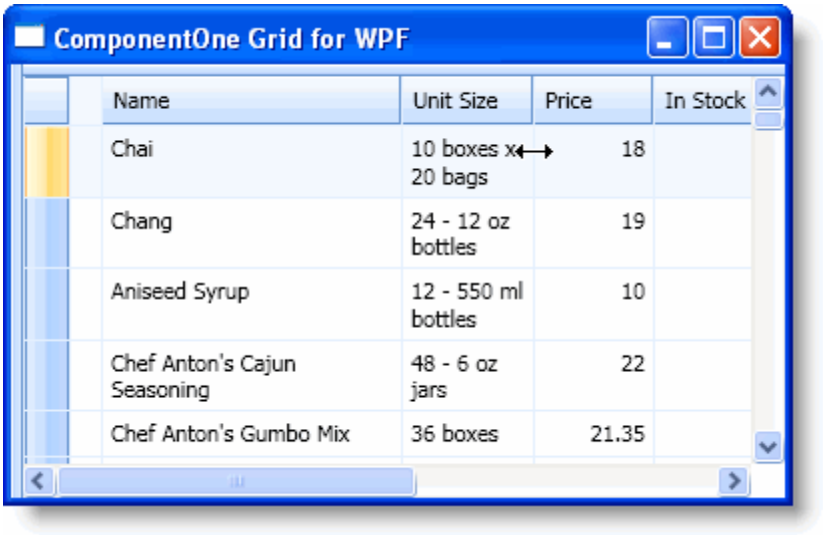
# Resizing Columns

Users can easily resize columns at run time through a drag-and-drop operation. To resize columns at run time, complete the following steps:

1. Navigate the mouse to the right border of a column. The column resizing cursor appears:



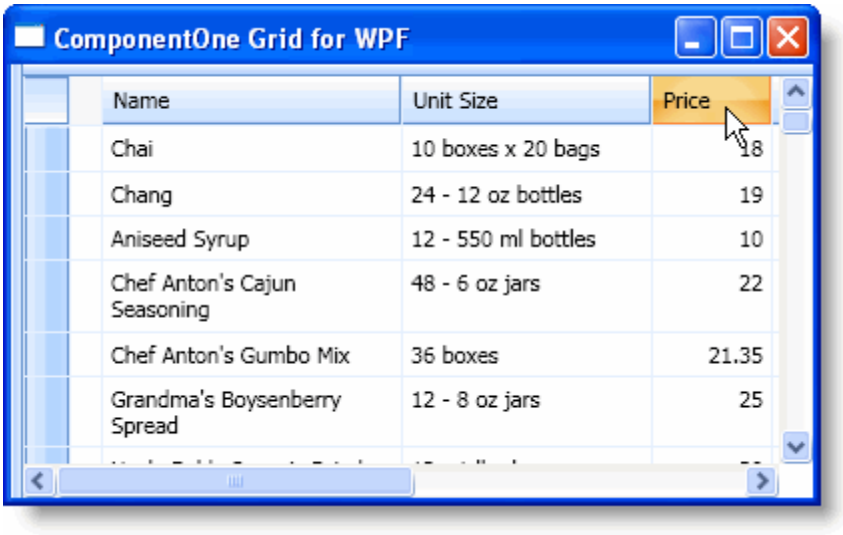

2. Click the mouse and drag the cursor to the left or the right to resize the column.



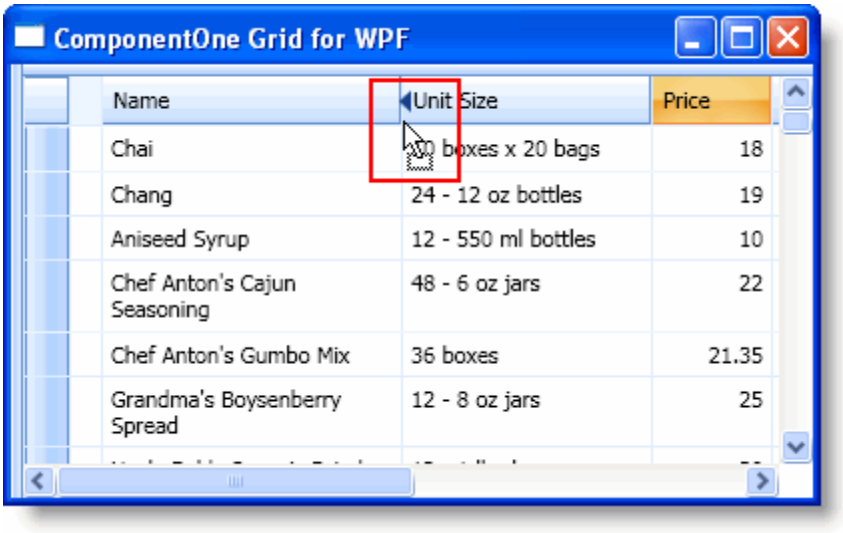

# Reordering Columns

End users can easily reorder columns at run time. To reorder columns at run time, complete the following steps:

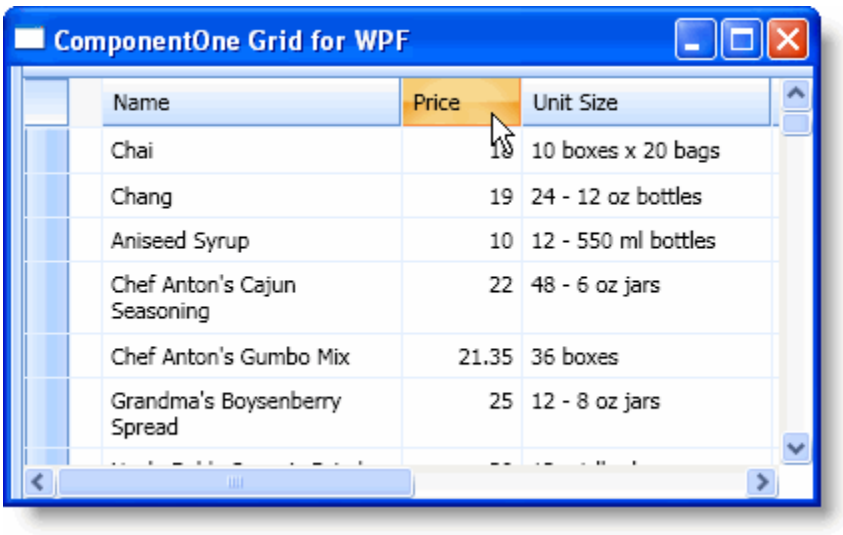
1. Click the column header for the column you wish to reorder. In the following image the *Price* column is selected.




2. Drag the column header to where you wish the column to be ordered. Notice that an arrow icon will appear if you can place the column in that location.



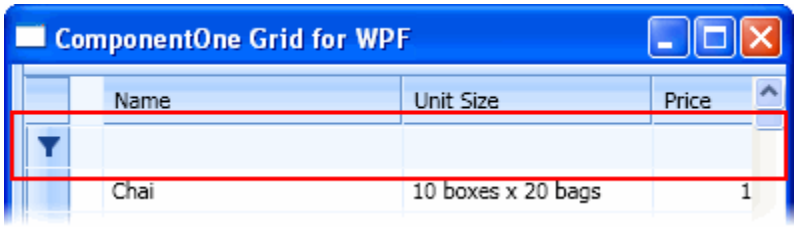

3. Release the mouse to place the column in its new location and reorder the columns.



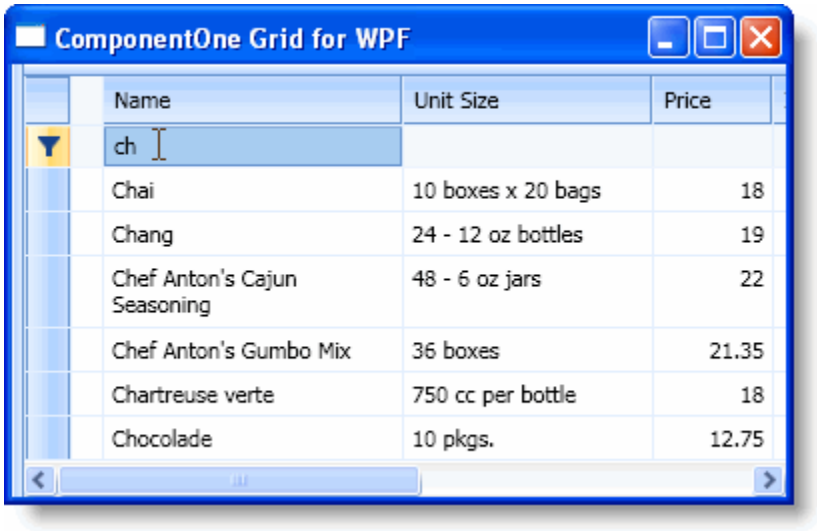

To prevent users from reordering the entire grid or specific columns at run time, see Preventing the Grid from Being Reordered (page 148) and Preventing a Column from Being Reordered (page 149).

# Filtering Columns

**ComponentOne Grid for WPF** incorporates a filter column element in the user interface, allowing users to filter columns by specific criteria at run time.



To filter a column's text at run time, simply enter the text in the filter bar that you want the column to be filtered by and press ENTER or click to another cell. For example, in the following image the *Name* column is filtered by the letters "ch" so that only content beginning with those letters appears.



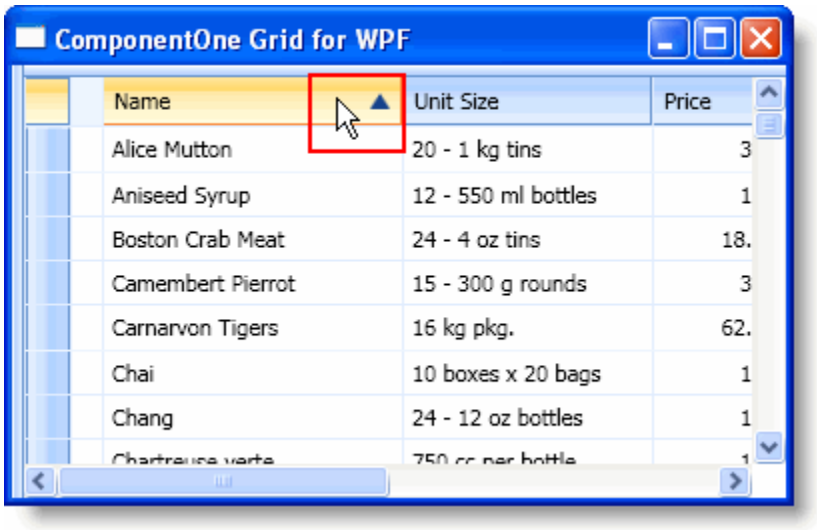

You can prevent users from filtering particular columns at run time by setting the AllowFilter property to **False**. For more information, see Preventing a Column from Being Filtered (page 147).

You can customize the filter bar's templates including the FilterBarContentTemplate, the FilterBarIndicatorTemplate, and the FilterBarTemplate.

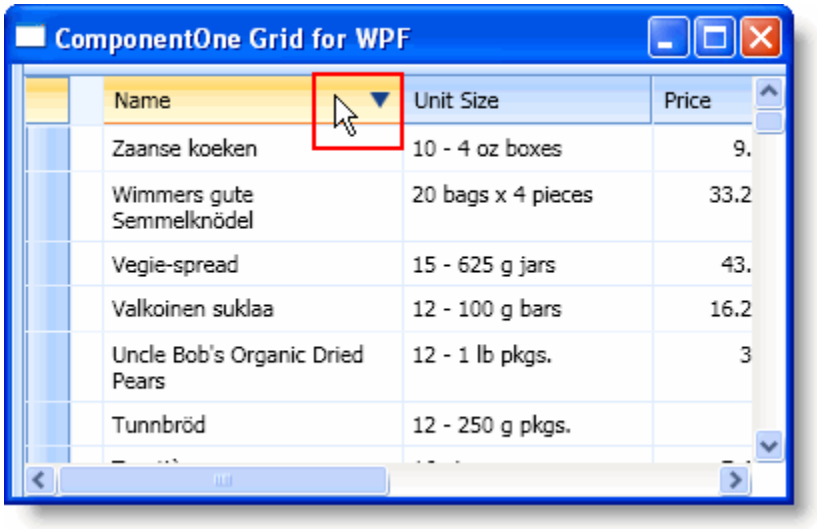
# Sorting Columns

Sorting grid columns at run time is simple in **ComponentOne Grid for WPF**. To sort columns click once on the header of the column that you wish to sort.

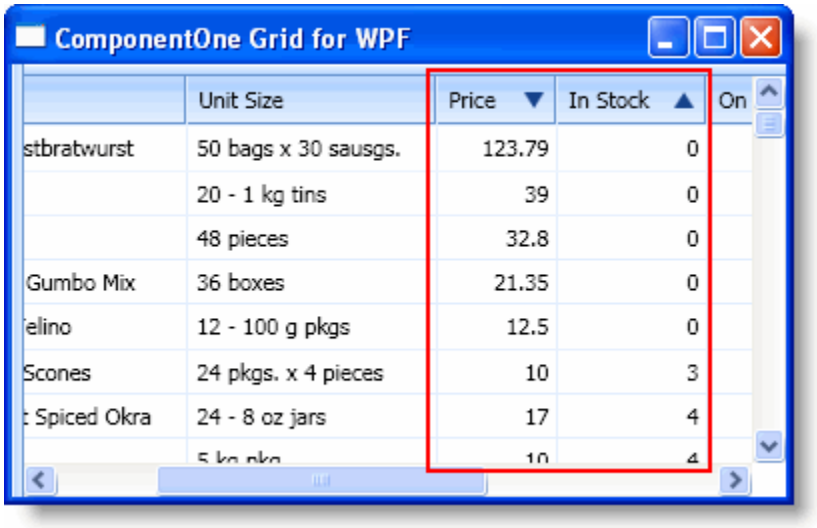
In the image below, the *Name* column is sorted; you will notice that the sort glyph, a sort direction indicator, appears when a column is sorted:



You can click once again on the column header to reverse the sort; notice that the sort glyph changes direction:




Sort multiple columns by sorting one column and then holding the CTRL key while clicking on a second column header to add that column to your sort condition. For example, in the following image the *In Stock* column was first sorted and then the *Price* column was reverse sorted:

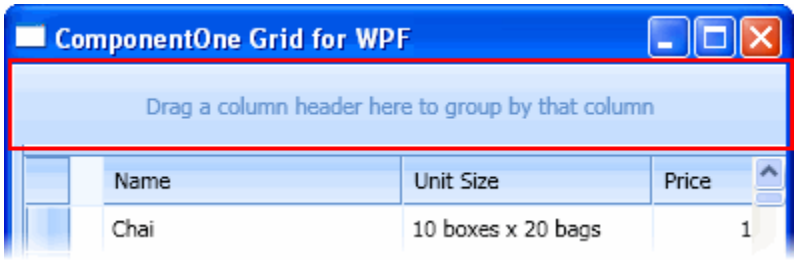



You can easily prevent the grid or a column in the grid from being sorted by setting the AllowSort property to **False**. For more information, see Preventing the Grid from Being Sorted (page 147) and Preventing a Column from Being Sorted (page 148).

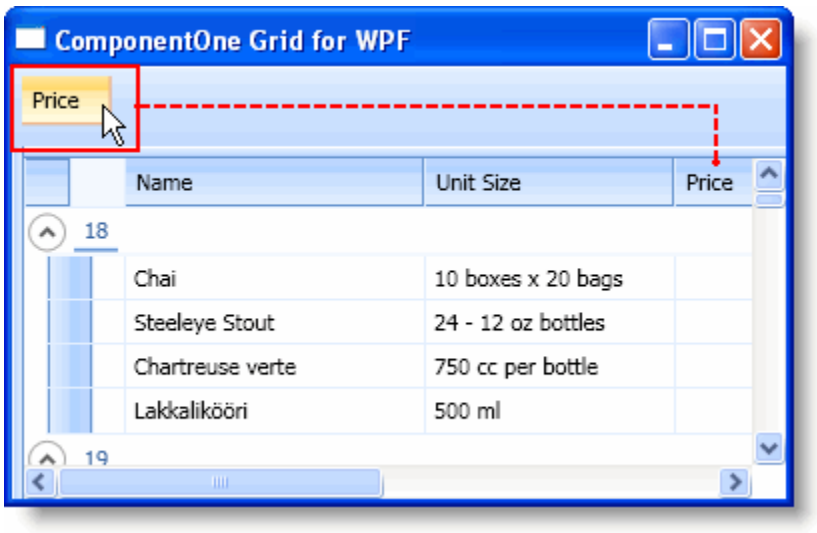
You can customize the sort direction indicators by using the AscendingSortIndicatorTemplate and DescendingSortIndicatorTemplate properties. See Changing the Appearance of the Sort Indicator (page 157) for an example of changing the appearance of the sort indicator.

# Grouping Columns

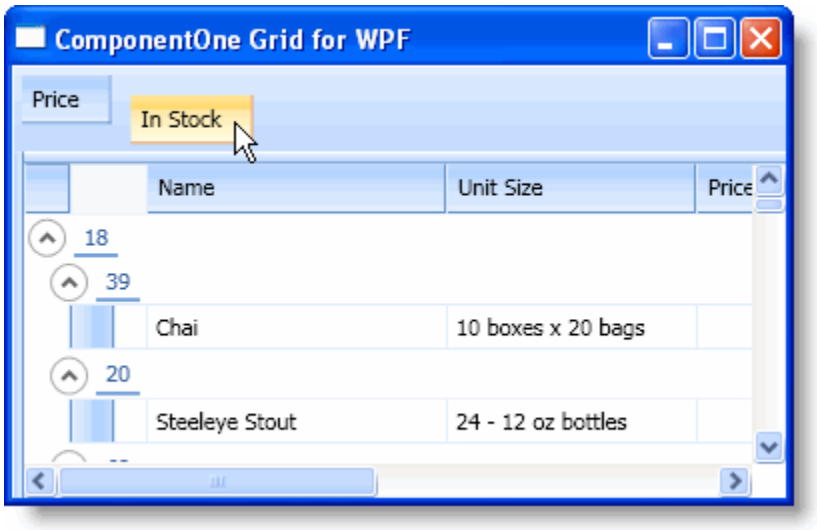
You can group columns in your grid to better organize information. The **GroupBy** area at the top of the grid allows you to easily group columns through a simple drag-and-drop operation.



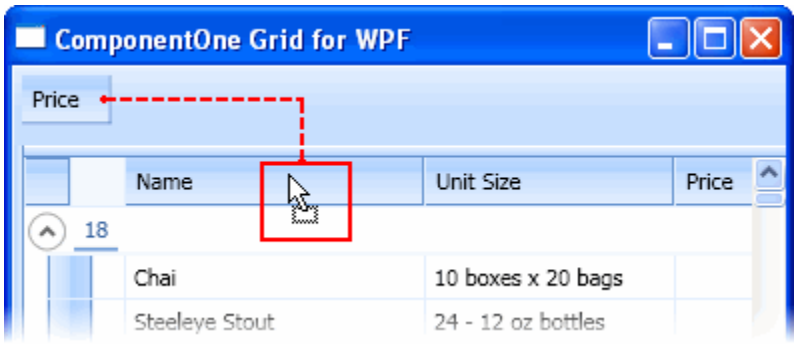

To group a column, drag-and-drop a column header onto the GroupBy area; for example, in the following image the grid is grouped by the *Price* column:




You can group multiple columns by performing a drag-and-drop operation to drag additional columns to the GroupBy area. In the following image, the grid is grouped by both the *Price* and *In Stock* columns:



To remove the grouping, simply click a column name in the GroupBy area and drag it onto the body of the grid:



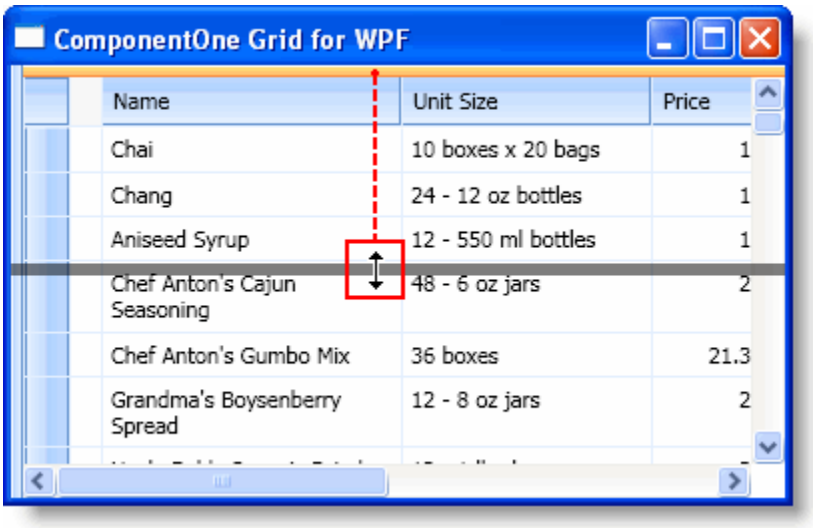

You can prevent users from grouping particular columns at run time by setting the AllowGroupBy property to **False**. For more information, see Preventing a Column from Being Grouped (page 149).

You can customize the GroupBy area through the GroupByCellContentTemplate, GroupByCellsAutoLayoutPanelTemplate, GroupByCellTemplate, and GroupByTemplate templates. You can also use the GroupByColumns and GroupStyles collections to group items at design time and change grouping styles.

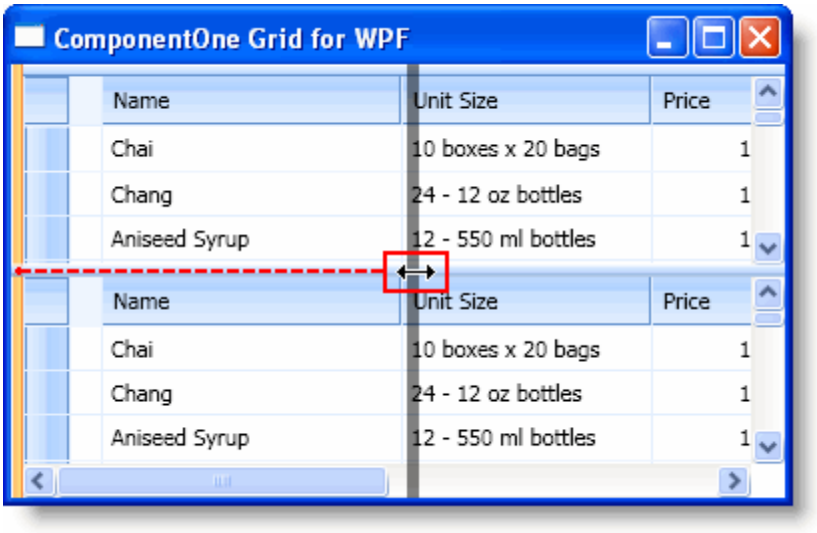
# Creating Splits

Users can easily create splits at run time to better view and organize the grid. Complete the following steps to create four separate areas of the grid by creating both a vertical split and a horizontal split:

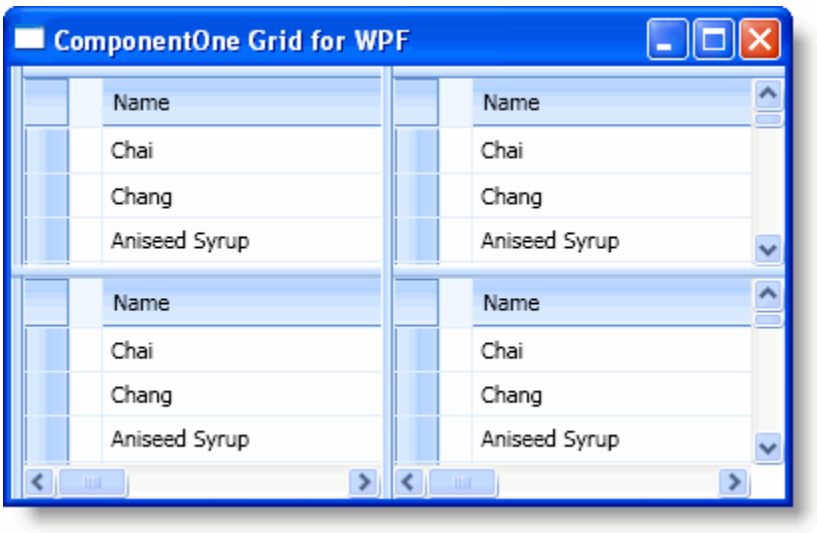
1. Click once on the top splitter bar to select it and drag it down to create two separate horizontal areas in the grid.




2. Click once on the left splitter bar to select it and drag it to the right to create two separate vertical areas in the grid.




The grid should now look similar to the following image and will consist of four separate grid areas:

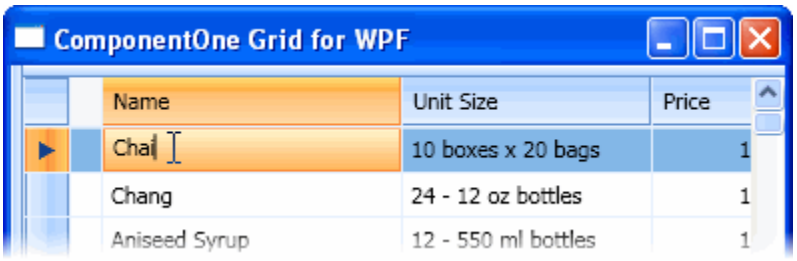



You can prevent users from creating and sizing splits by setting the AllowHorizontalSplit, AllowHorizontalSplitSizing, AllowVerticalSplit, and AllowVerticalSplitSizing properties to **False**. For more information, see Preventing Users from Splitting the Grid (page 150).

# Editing Cells

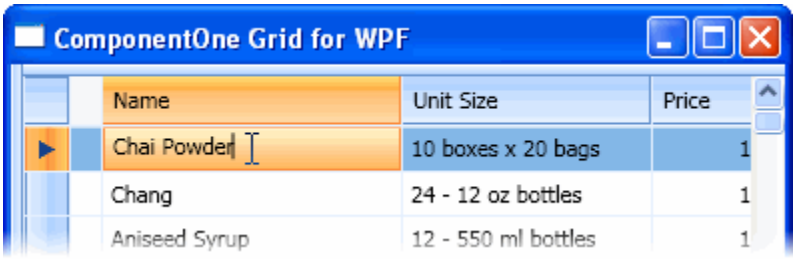
Users can easily edit cell content at run time. Editing content is as simple as selecting a cell and deleting or changing the content in that cell. Complete the following steps to edit cell content:

1. Double-click the cell you would like to edit.

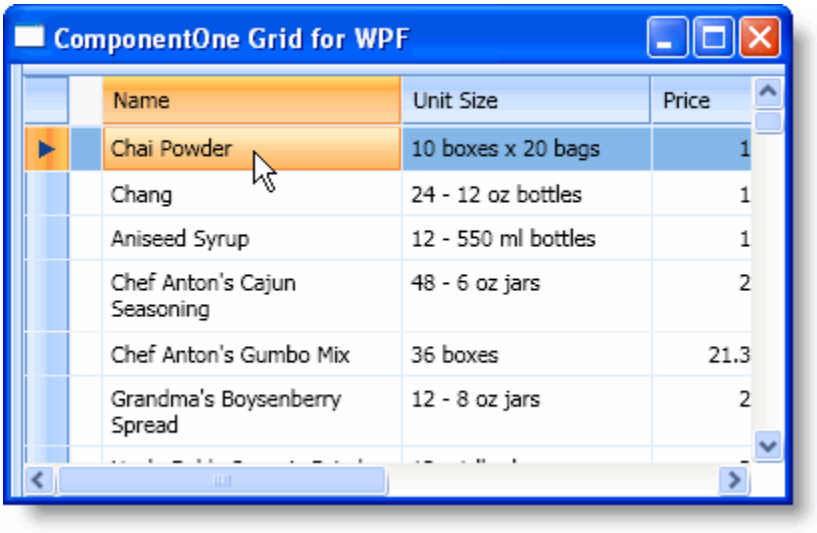



A cursor will appear in that cell indicating that it can be edited.

2. Delete text or type in new or additional text to edit the content of the cell.



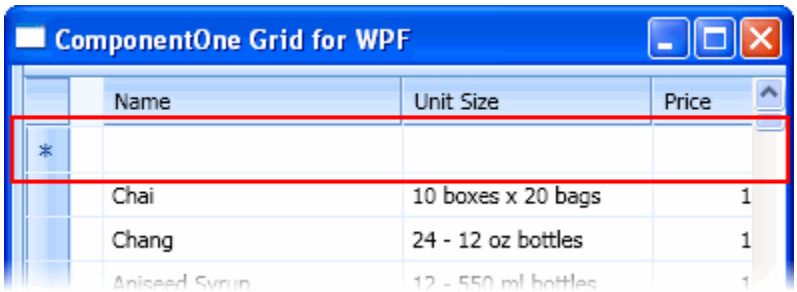

3. Press ENTER or click away from the cell you are editing for the changes you made to take effect.




Note that you can prevent users from altering particular columns by setting a column's **ReadOnly** property to **True**. For more information, see Locking Columns from Being Edited (page 150).

# Adding Rows to the Grid

You can add rows to the grid at run time using the new row bar. The new row bar, indicated by an asterisk symbol (*), allows you to type in new information to add to the grid at run time.



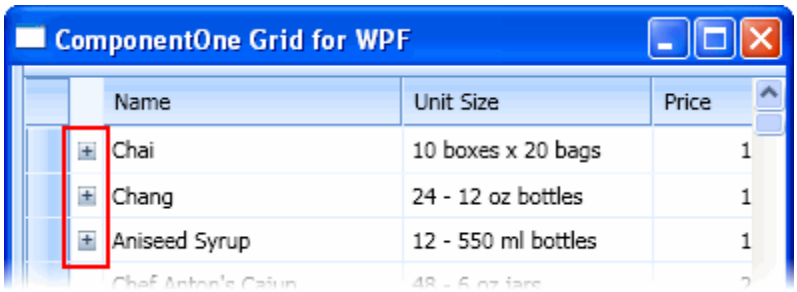

Simply type text into the new row bar and press the UP or DOWN arrow key for text to be added to the grid in a new row. You can cancel adding a new row by pressing the ESC key. Note that you may need to press ESC twice if you're currently editing a cell.

Use the NewItemIndicatorTemplate template to customize the new row bar. You can control the location of the new row bar and prevent users from adding rows to the grid by using the NewRowPlacement property. For more information, see Setting New Row Placement (page 89).

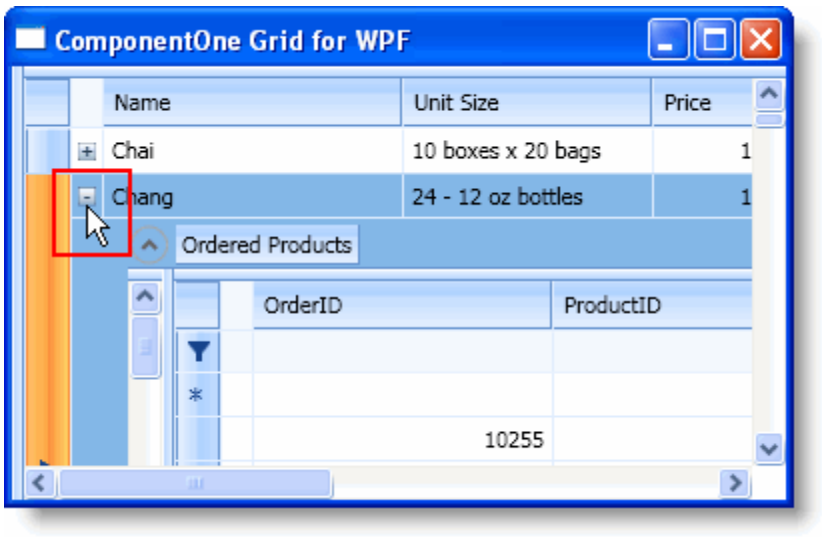
# Viewing Child Tables

**ComponentOne Grid for WPF** supports automatic hierarchical data representation; so, for example, when the grid is bound to a table with a child table, at run time you can expand the child table under the main table to see related data. For information about hierarchical data views, see Setting Up Hierarchical Data Views (page 55).

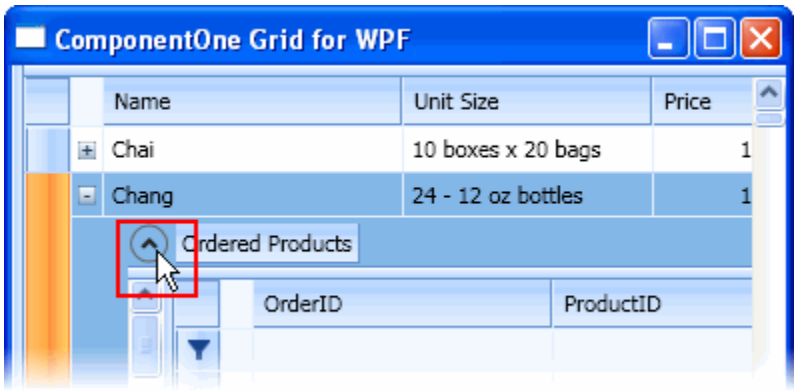
If a child table exists in the current grid, you'll see a plus sign (+) in the child grid indicator column next to each row with a child table.



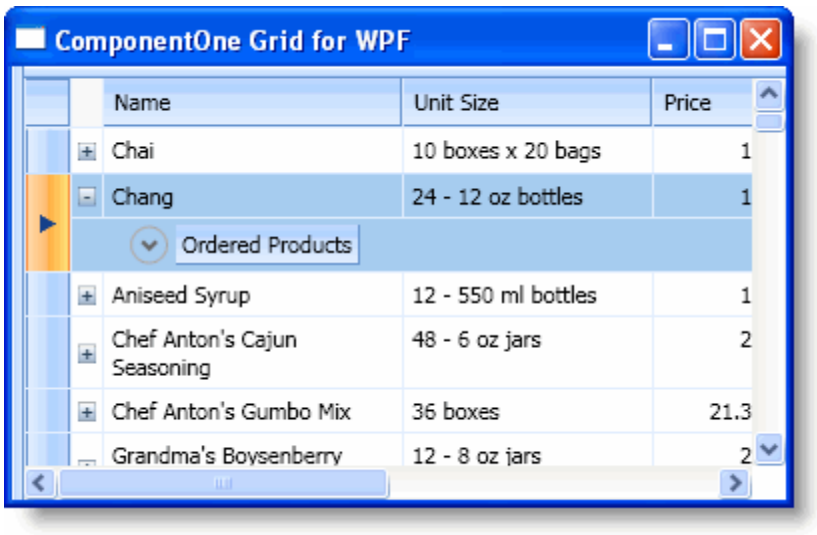

Simply click the plus sign to view the child grid. Notice that the indicator changes from a plus sign to a minus sign (-) to indicate that the child grid can be closed:




You can also collapse an open child grid table by clicking on the arrow next to the child grid's name, which is particularly useful when there are multiple child tables:




When the child table is collapsed, the grid will look similar to the following:




You can prevent the grid from displaying child tables by setting the AllowHierarchicalData property to **False**, for details see Hiding Child Tables (page 152).

# Grid for WPF Appearance

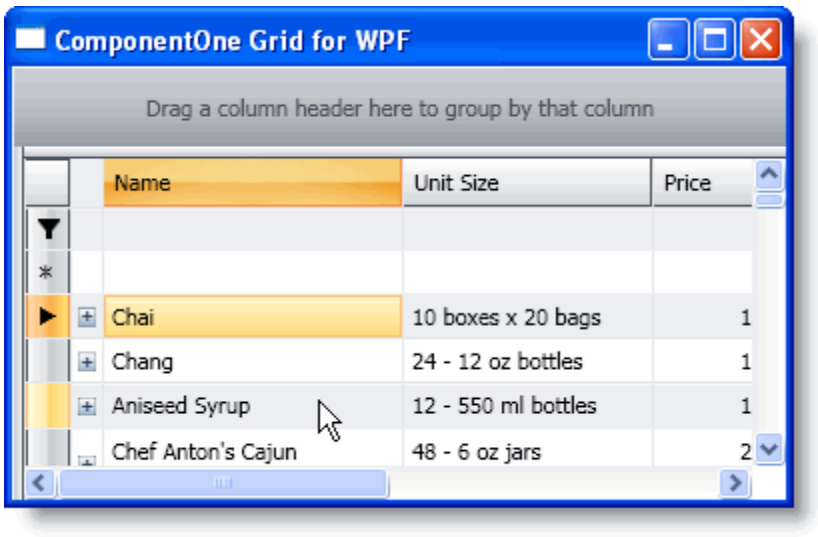
There are several ways you can customize the **ComponentOne Grid for WPF** grid's appearance. The following topics discuss some of these customizations including themes, data views, text, cell styles and highlighting, and templates. Note that in images in the following topics the GroupByVisibility and the FilterBarVisibility properties may have been set to **Collapsed** and the NewRowPlacement property may have been set to **None** to better illustrate the topic under discussion.

## Using Themes

**ComponentOne Grid for WPF** incorporates several themes, including Office 2007, Vista, and Office 2007 themes, that allow you to customize the appearance of your grid. The default theme is the **Office2007Default** theme. The built-in themes are described and pictured below; note that in the images below, a cell has been selected and the mouse is hovering over another cell to show both selected and hover styles:

### Office2007Black Theme

This is the default theme based on the Office 2007 Black style and it appears as a gray-colored grid with orange highlighting.




**In XAML**

To specifically define the **Office2007Black** theme in your grid, add the following `Theme` XAML to the `<c1grid:C1DataGrid>` tag so that it appears similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=GridDataSet.Products, ElementName=Window1, Mode=Default}"
Theme="{DynamicResource {c1grid:C1ThemeKey ThemeName=Office2007Black,
TypeInTargetAssembly=c1grid:C1DataGrid}}">
```

**In Code**

To specifically define the **Office2007Black** theme in your grid, add the following code your project:

- Visual Basic

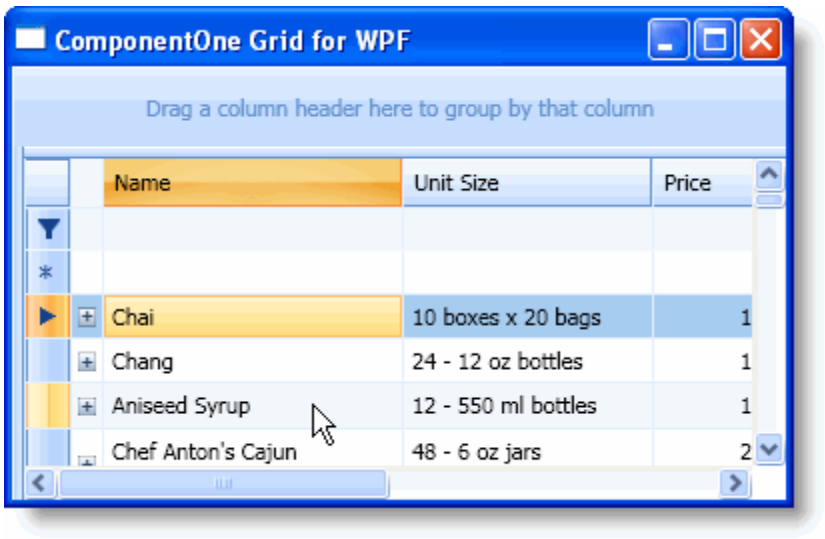
```
C1DataGrid1.Theme = TryCast(c1DataGrid1.FindResource(New
C1ThemeKey("Office2007Black", GetType(C1DataGrid))), ResourceDictionary)
```

- C#

```
c1DataGrid1.Theme = c1DataGrid1.FindResource(new
C1ThemeKey("Office2007Black", typeof(C1DataGrid))) as ResourceDictionary;
```

### Office2007Blue Theme

This theme is based on the Office 2007 Blue style and it appears as a blue-colored grid with orange highlighting.




**In XAML**

To specifically define the **Office2007Blue** theme in your grid, add the following `Theme` XAML to the `<c1grid:C1DataGrid>` tag so that it appears similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=GridDataSet.Products, ElementName=Window1, Mode=Default}"
Theme="{DynamicResource {c1grid:C1ThemeKey ThemeName=Office2007Blue,
TypeInTargetAssembly=c1grid:C1DataGrid}}">
```

**In Code**

To define the **Office2007Blue** theme in your grid, add the following code your project:

- Visual Basic

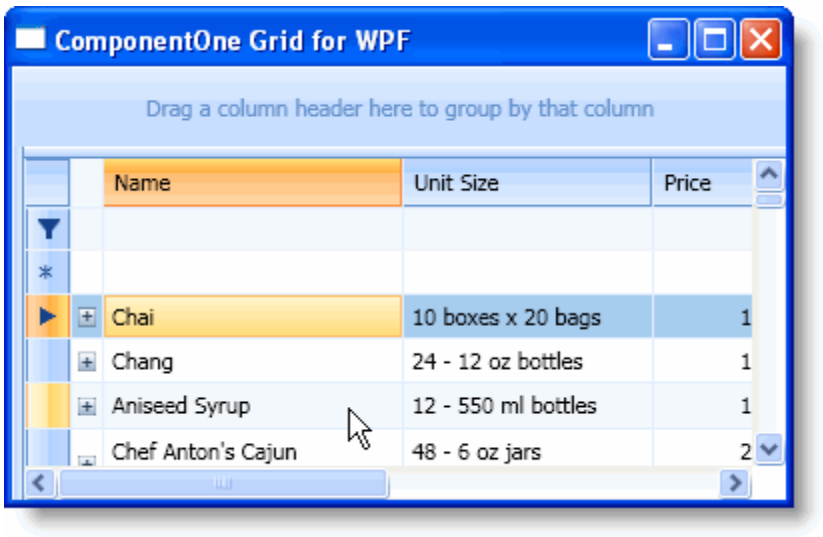
```
C1DataGrid1.Theme = TryCast(c1DataGrid1.FindResource(New
C1ThemeKey("Office2007Blue", GetType(C1DataGrid))), ResourceDictionary)
```

- C#

```
c1DataGrid1.Theme = c1DataGrid1.FindResource(new
C1ThemeKey("Office2007Blue", typeof(C1DataGrid))) as ResourceDictionary;
```

**Office2007Default Theme**

This theme is similar to the Office 2007 Blue style, but does not use Visual Brushes. It appears as a blue-colored grid with orange highlighting. This is the default theme.




**In XAML**

To specifically define the **Office2007Default** theme in your grid, add the following `Theme` XAML to the `<c1grid:C1DataGrid>` tag so that it appears similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=GridDataSet.Products, ElementName=Window1, Mode=Default}"
Theme="{DynamicResource {c1grid:C1ThemeKey ThemeName=Office2007Default,
TypeInTargetAssembly=c1grid:C1DataGrid}}">
```

**In Code**

To define the **Office2007Default** theme in your grid, add the following code your project:

- Visual Basic

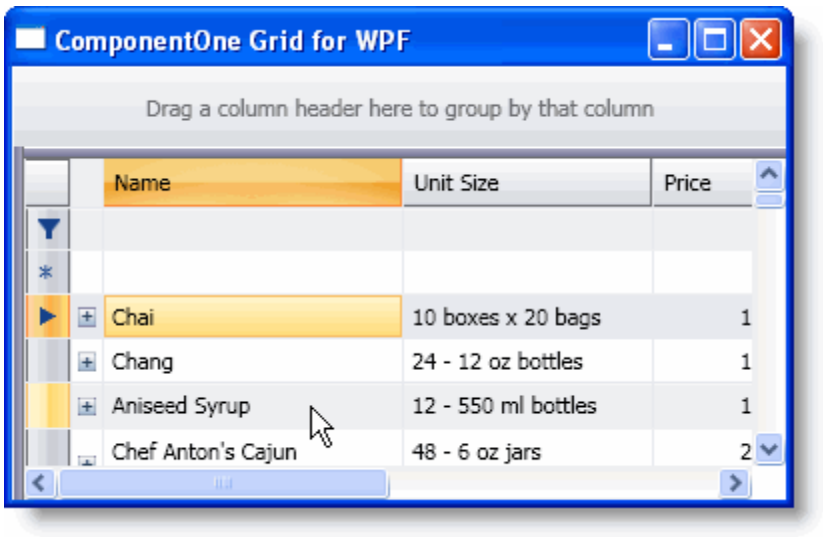
```
C1DataGrid1.Theme = TryCast(c1DataGrid1.FindResource(New
C1ThemeKey("Office2007Default", GetType(C1DataGrid))), ResourceDictionary)
```

- C#

```
c1DataGrid1.Theme = c1DataGrid1.FindResource(new
C1ThemeKey("Office2007Default", typeof(C1DataGrid))) as
ResourceDictionary;
```

**Office2007Silver Theme**

This theme is based on the Office 2007 Silver style and it appears as a silver-colored grid with orange highlighting.




**In XAML**

To specifically define the **Office2007Silver** theme in your grid, add the following `Theme` XAML to the `<c1grid:C1DataGrid>` tag so that it appears similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=GridDataSet.Products, ElementName=Window1, Mode=Default}"
Theme="{DynamicResource {c1grid:C1ThemeKey ThemeName=Office2007Silver,
TypeInTargetAssembly=c1grid:C1DataGrid}}">
```

**In Code**

To specifically define the **Office2007Silver** theme in your grid, add the following code your project:

- Visual Basic

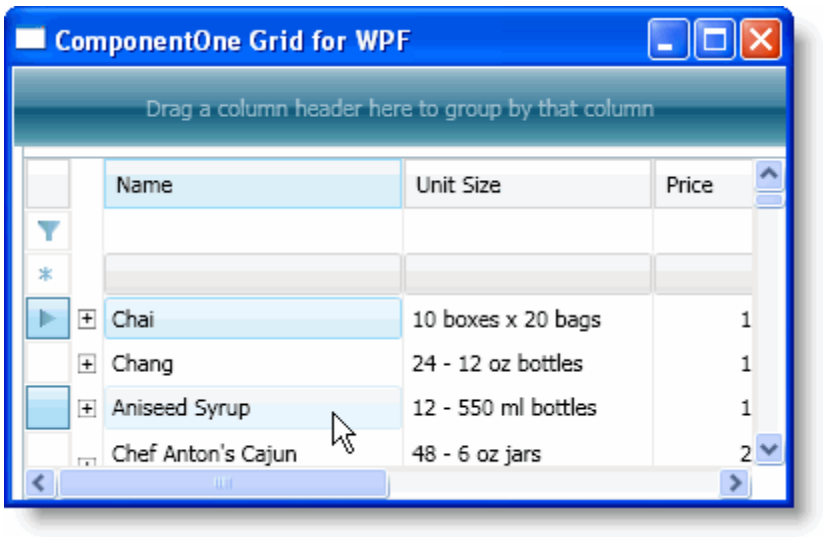
```
C1DataGrid1.Theme = TryCast(c1DataGrid1.FindResource(New
C1ThemeKey("Office2007Silver", GetType(C1DataGrid))), ResourceDictionary)
```

- C#

```
c1DataGrid1.Theme = c1DataGrid1.FindResource(new
C1ThemeKey("Office2007Silver", typeof(C1DataGrid))) as ResourceDictionary;
```

**Vista Theme**

This theme is based on the Vista style and it appears as a teal-colored grid with blue highlighting.




**In XAML**

To specifically define the **Vista** theme in your grid, add the following `Theme` XAML to the `<c1grid:C1DataGrid>` tag so that it appears similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=GridDataSet.Products, ElementName=Window1, Mode=Default}"
Theme="{DynamicResource {c1grid:C1ThemeKey ThemeName=Vista,
TypeInTargetAssembly=c1grid:C1DataGrid}}">
```

**In Code**

To specifically define the **Vista** theme in your grid, add the following code your project:

- Visual Basic

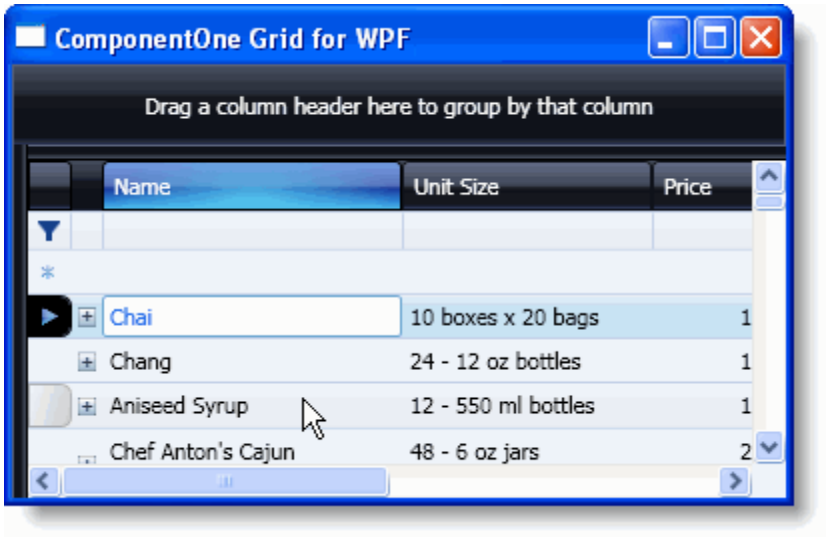
```
C1DataGrid1.Theme = TryCast(c1DataGrid1.FindResource(New
C1ThemeKey("Vista", GetType(C1DataGrid))), ResourceDictionary)
```

- C#

```
c1DataGrid1.Theme = c1DataGrid1.FindResource(new C1ThemeKey("Vista",
typeof(C1DataGrid))) as ResourceDictionary;
```

### MediaPlayer Theme

This theme is based on the Windows Media Player style and it appears as a black-colored grid with blue highlighting.




**In XAML**

To specifically define the **MediaPlayer** theme in your grid, add the following `Theme` XAML to the `<c1grid:C1DataGrid>` tag so that it appears similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=GridDataSet.Products, ElementName=Window1, Mode=Default}"
Theme="{DynamicResource {c1grid:C1ThemeKey ThemeName=MediaPlayer,
TypeInTargetAssembly=c1grid:C1DataGrid}}">
```

**In Code**

To specifically define the **MediaPlayer** theme in your grid, add the following code your project:

- Visual Basic

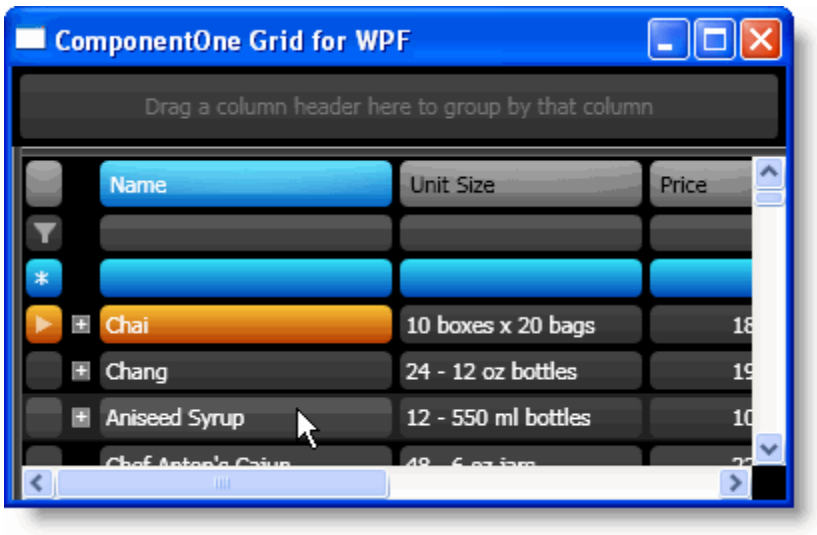
```
C1DataGrid1.Theme = TryCast(c1DataGrid1.FindResource(New
C1ThemeKey("MediaPlayer", GetType(C1DataGrid))), ResourceDictionary)
```

- C#

```
c1DataGrid1.Theme = c1DataGrid1.FindResource(new C1ThemeKey("MediaPlayer",
typeof(C1DataGrid))) as ResourceDictionary;
```

**DuskBlue Theme**

This theme appears as a charcoal-colored grid with electric blue and orange highlighting.




**In XAML**

To specifically define the **DuskBlue** theme in your grid, add the following `Theme` XAML to the `<c1grid:C1DataGrid>` tag so that it appears similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=GridDataSet.Products, ElementName=Window1, Mode=Default}"
Theme="{DynamicResource {c1grid:C1ThemeKey ThemeName=DuskBlue,
TypeInTargetAssembly=c1grid:C1DataGrid}}">
```

**In Code**

To define the **DuskBlue** theme in your grid, add the following code your project:

- Visual Basic

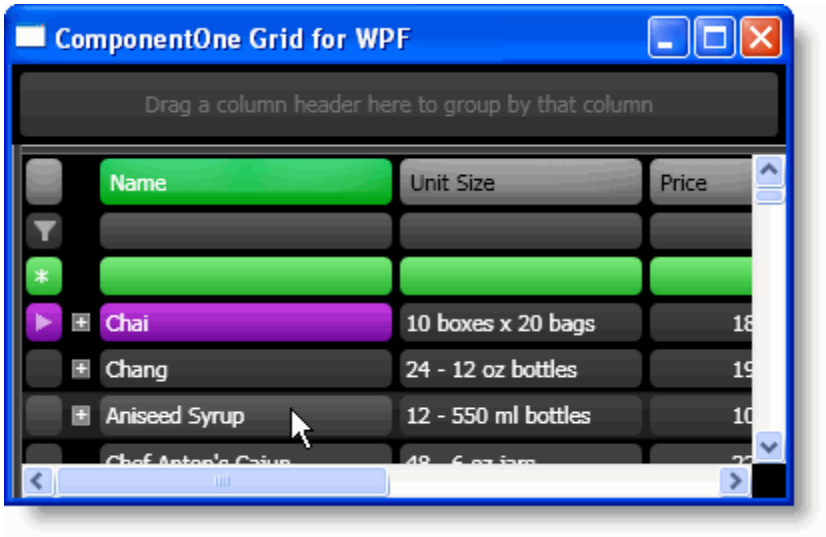
```
C1DataGrid1.Theme = TryCast(c1DataGrid1.FindResource(New
C1ThemeKey("DuskBlue", GetType(C1DataGrid))), ResourceDictionary)
```

- C#

```
c1DataGrid1.Theme = c1DataGrid1.FindResource(new C1ThemeKey("DuskBlue",
typeof(C1DataGrid))) as ResourceDictionary;
```

**DuskGreen Theme**

This theme appears as a charcoal -colored grid with electric green and purple highlighting.




**In XAML**

To specifically define the **DuskGreen** theme in your grid, add the following `Theme` XAML to the `<c1grid:C1DataGrid>` tag so that it appears similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=GridDataSet.Products, ElementName=Window1, Mode=Default}"
Theme="{DynamicResource {c1grid:C1ThemeKey ThemeName=DuskGreen,
TypeInTargetAssembly=c1grid:C1DataGrid}}">
```

**In Code**

To define the **DuskGreen** theme in your grid, add the following code your project:

- Visual Basic

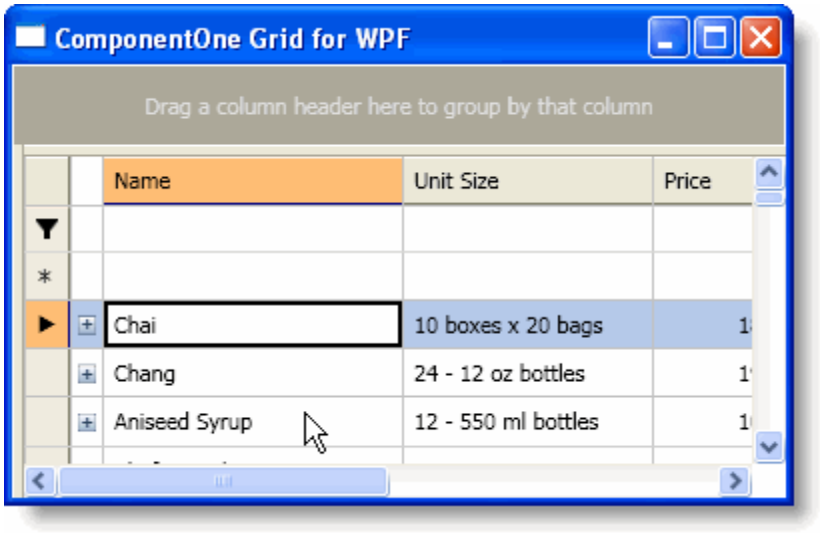
```
C1DataGrid1.Theme = TryCast(c1DataGrid1.FindResource(New
C1ThemeKey("DuskGreen", GetType(C1DataGrid))), ResourceDictionary)
```

- C#

```
c1DataGrid1.Theme = c1DataGrid1.FindResource(new C1ThemeKey("DuskGreen",
typeof(C1DataGrid))) as ResourceDictionary;
```

**Office2003Blue Theme**

This theme is based on the Office 2003 Blue style and it appears as a neutral-colored grid with blue and orange highlighting.




**In XAML**

To specifically define the **Office2003Blue** theme in your grid, add the following `Theme` XAML to the `<c1grid:C1DataGrid>` tag so that it appears similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=GridDataSet.Products, ElementName=Window1, Mode=Default}"
Theme="{DynamicResource {c1grid:C1ThemeKey ThemeName=Office2003Blue,
TypeInTargetAssembly=c1grid:C1DataGrid}}">
```

**In Code**

To define the **Office2003Blue** theme in your grid, add the following code your project:

- Visual Basic

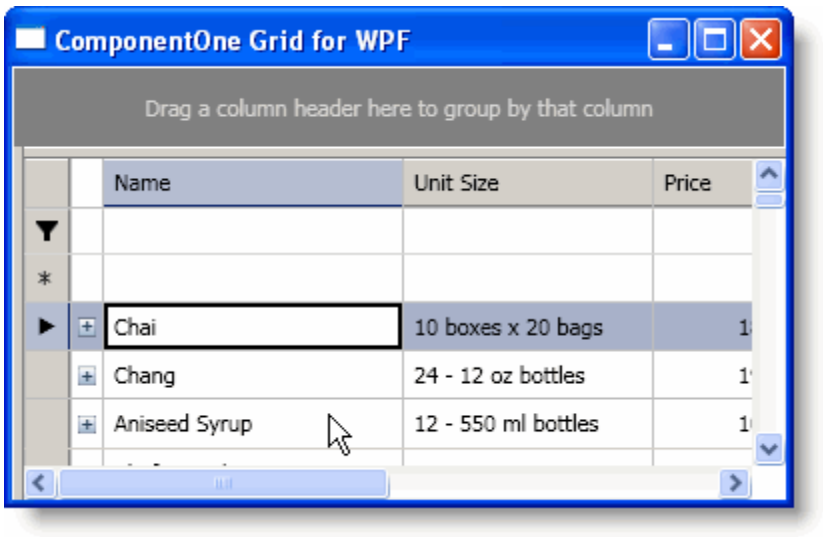
```
C1DataGrid1.Theme = TryCast(c1DataGrid1.FindResource(New
C1ThemeKey("Office2003Blue", GetType(C1DataGrid))), ResourceDictionary)
```

- C#

```
c1DataGrid1.Theme = c1DataGrid1.FindResource(new
C1ThemeKey("Office2003Blue", typeof(C1DataGrid))) as ResourceDictionary;
```

**Office2003Classic Theme**

This theme is based on the Office 2003 Classic style and appears as a gray-colored grid with slate-colored highlighting.




**In XAML**

To specifically define the **Office2003Classic** theme in your grid, add the following `Theme` XAML to the `<c1grid:C1DataGrid>` tag so that it appears similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=GridDataSet.Products, ElementName=Window1, Mode=Default}"
Theme="{DynamicResource {c1grid:C1ThemeKey ThemeName=Office2003Classic,
TypeInTargetAssembly=c1grid:C1DataGrid}}">
```

**In Code**

To define the **Office2003Classic** theme in your grid, add the following code your project:

- Visual Basic

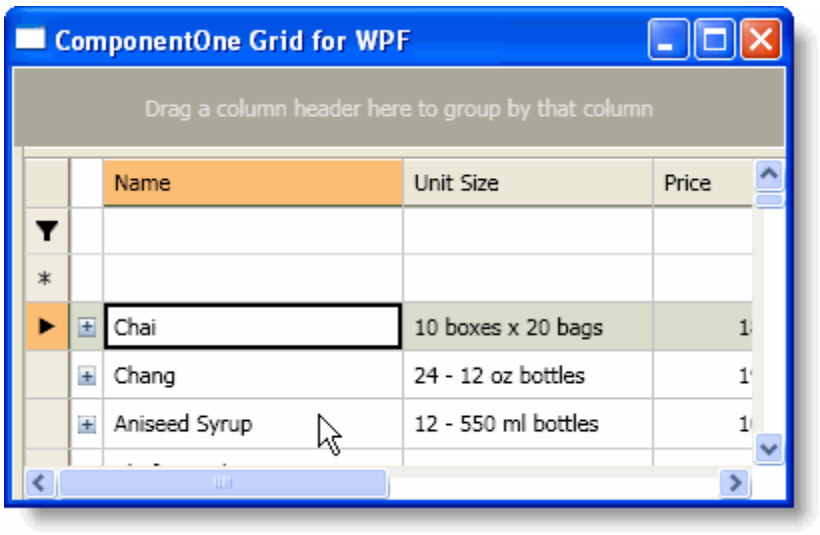
```
C1DataGrid1.Theme = TryCast(c1DataGrid1.FindResource(New
C1ThemeKey("Office2003Classic", GetType(C1DataGrid))), ResourceDictionary)
```

- C#

```
c1DataGrid1.Theme = c1DataGrid1.FindResource(new
C1ThemeKey("Office2003Classic", typeof(C1DataGrid))) as
ResourceDictionary;
```

**Office2003Olive Theme**

This theme is based on the Office 2003 Olive style and it appears as a neutral-colored grid with olive green and orange highlighting.




**In XAML**

To specifically define the **Office2003Olive** theme in your grid, add the following `Theme` XAML to the `<c1grid:C1DataGrid>` tag so that it appears similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=GridDataSet.Products, ElementName=Window1, Mode=Default}"
Theme="{DynamicResource {c1grid:C1ThemeKey ThemeName=Office2003Olive,
TypeInTargetAssembly=c1grid:C1DataGrid}}">
```

**In Code**

To define the **Office2003Olive** theme in your grid, add the following code your project:

- Visual Basic

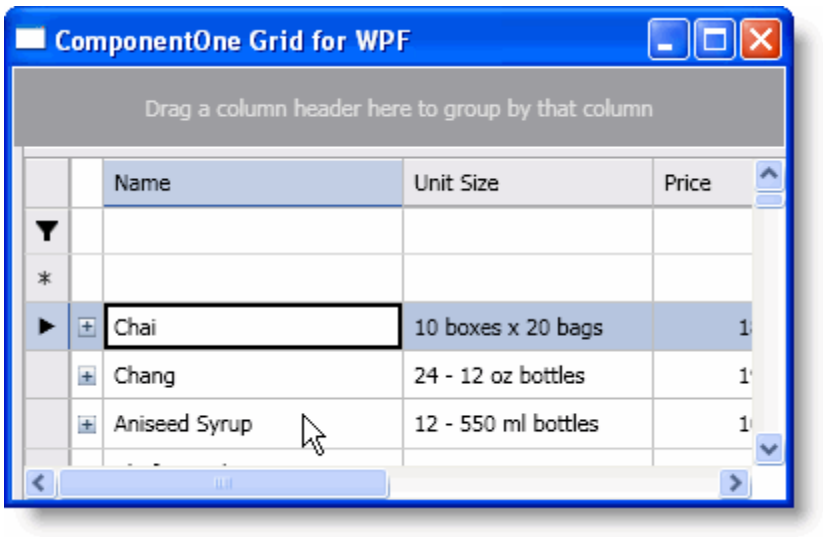
```
C1DataGrid1.Theme = TryCast(c1DataGrid1.FindResource(New
C1ThemeKey("Office2003Olive", GetType(C1DataGrid))), ResourceDictionary)
```

- C#

```
c1DataGrid1.Theme = c1DataGrid1.FindResource(new
C1ThemeKey("Office2003Olive", typeof(C1DataGrid))) as ResourceDictionary;
```

**Office2003Royale Theme**

This theme is similar to the Office 2003 Royale style and appears as a silver-colored grid with blue highlighting.




**In XAML**

To specifically define the **Office2003Royale** theme in your grid, add the following `Theme` XAML to the `<c1grid:C1DataGrid>` tag so that it appears similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=GridDataSet.Products, ElementName=Window1, Mode=Default}"
Theme="{DynamicResource {c1grid:C1ThemeKey ThemeName=Office2003Royale,
TypeInTargetAssembly=c1grid:C1DataGrid}}">
```

**In Code**

To define the **Office2003Royale** theme in your grid, add the following code your project:

- Visual Basic

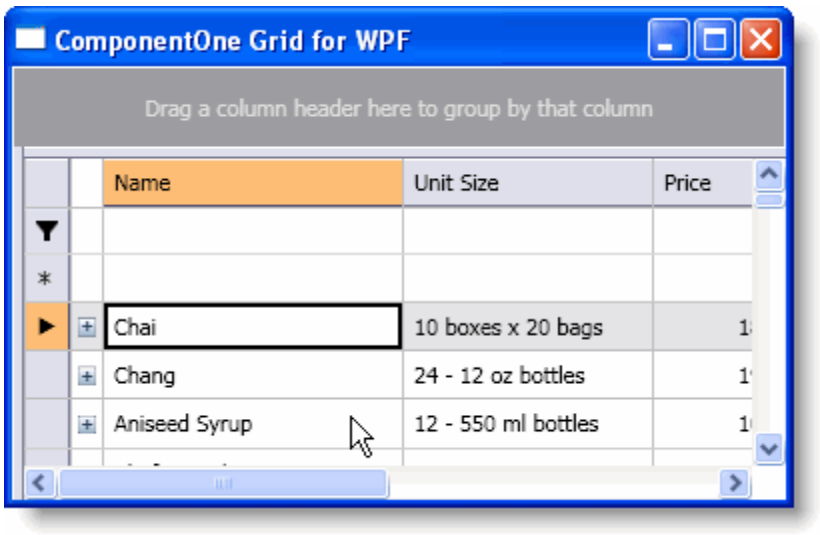
```
C1DataGrid1.Theme = TryCast(c1DataGrid1.FindResource(New
C1ThemeKey("Office2003Royale", GetType(C1DataGrid))), ResourceDictionary)
```

- C#

```
c1DataGrid1.Theme = c1DataGrid1.FindResource(new
C1ThemeKey("Office2003Royale", typeof(C1DataGrid))) as ResourceDictionary;
```

**Office2003Silver Theme**

This theme is based on the Office 2003 Silver style and it appears as a silver-colored grid with gray and orange highlighting.




**In XAML**

To specifically define the **Office2003Silver** theme in your grid, add the following `Theme` XAML to the `<c1grid:C1DataGrid>` tag so that it appears similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=GridDataSet.Products, ElementName=Window1, Mode=Default}"
Theme="{DynamicResource {c1grid:C1ThemeKey ThemeName=Office2003Silver,
TypeInTargetAssembly=c1grid:C1DataGrid}}">
```

**In Code**

To specifically define the **Office2003Silver** theme in your grid, add the following code your project:

- Visual Basic

```
C1DataGrid1.Theme = TryCast(c1DataGrid1.FindResource(New
C1ThemeKey("Office2003Silver", GetType(C1DataGrid))), ResourceDictionary)
```

- C#

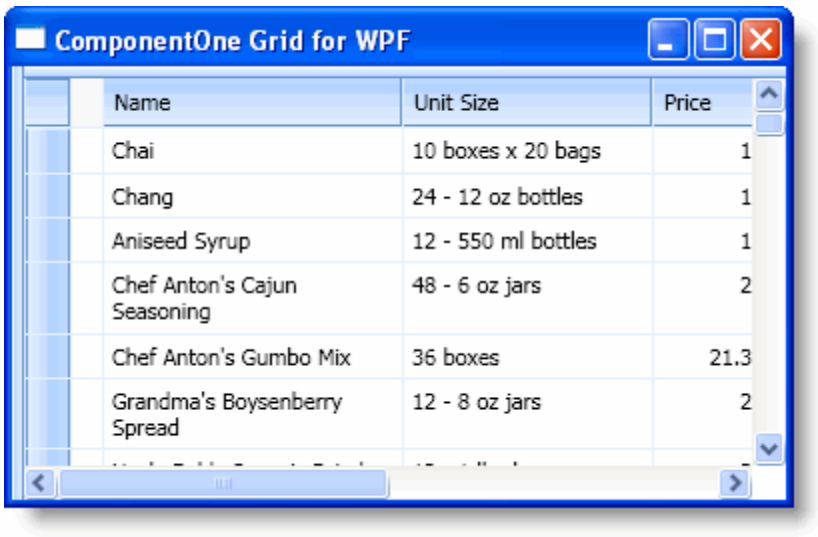
```
c1DataGrid1.Theme = c1DataGrid1.FindResource(new
C1ThemeKey("Office2003Silver", typeof(C1DataGrid))) as ResourceDictionary;
```

# Using Data Views

**ComponentOne Grid for WPF** incorporates various data views that allow you to customize the appearance of your grid. These views include a tabular view with vertical row distribution (the default view), a tabular view with horizontal row distribution, a card view, and a carousel view. In addition you can create your own custom views. Each built-in view is pictured and described below.

### Vertical Tabular Grid

This is the default data view and it appears like a traditional grid with headers at the top of the grid and rows and columns of content below the header.




**In XAML**

To specifically define the tabular view with vertical row distribution in your grid, add the following `Style` XAML to the `<c1grid:C1DataGrid>` tag so that it appears similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=GridDataSet.Products, ElementName=Window1, Mode=Default}"
Style="{StaticResource {ComponentResourceKey
ResourceId=TabularVerticalViewStyle, TypeInTargetAssembly={x:Type
c1grid:C1DataGrid}}}">
```

**In Code**

To specifically define the tabular view with vertical row distribution in your grid, add the following code your project:

- Visual Basic

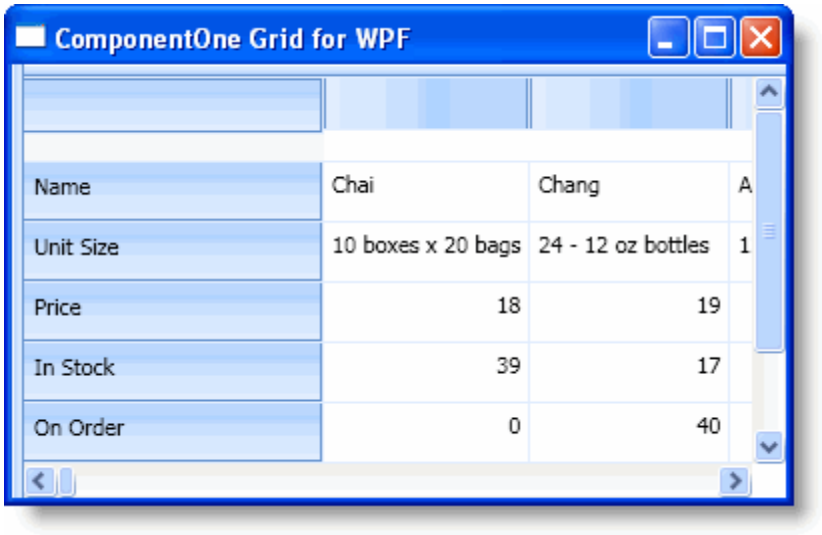
```
C1DataGrid1.Style = TryCast(C1DataGrid1.FindResource(New
ComponentResourceKey(GetType(C1DataGrid), "TabularVerticalViewStyle")),
Style)
```

- C#

```
c1DataGrid1.Style = c1DataGrid1.FindResource(new
ComponentResourceKey(typeof(C1DataGrid), "TabularVerticalViewStyle")) as
Style;
```

**Horizontal Tabular Grid**

This view is similar to the tabular view with vertical row distribution, but in this case the grid is organized horizontally with the headers to the left and content to the right of the grid.




**In XAML**

To specify the tabular view with horizontal row distribution in your grid, add the following `Style` XAML to the `<c1grid:C1DataGrid>` tag so that it appears similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=GridDataSet.Products, ElementName=Window1, Mode=Default}"
Style="{StaticResource {ComponentResourceKey
ResourceId=TabularHorizontalViewStyle, TypeInTargetAssembly={x:Type
c1grid:C1DataGrid}}}">
```

**In Code**

To define the tabular view with horizontal row distribution in your grid, add the following code your project:

- Visual Basic

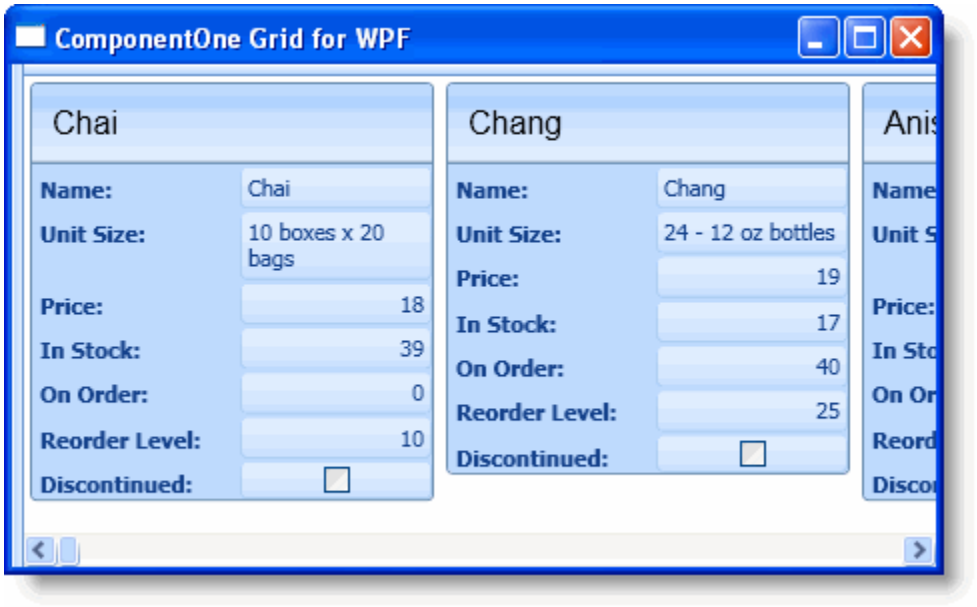
```
C1DataGrid1.Style = TryCast(C1DataGrid1.FindResource(New
ComponentResourceKey(GetType(C1DataGrid), "TabularHorizontalViewStyle")),
Style)
```

- C#

```
c1DataGrid1.Style = c1DataGrid1.FindResource(new
ComponentResourceKey(typeof(C1DataGrid), "TabularHorizontalViewStyle")) as
Style;
```

### Card View

This is a card view that appears similar to a traditional rolodex or card catalog in which each row appears in its own 'card'.




**In XAML**

To specify the card view in your grid, add the following `Style` XAML to the `<c1grid:C1DataGrid>` tag so that it appears similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=GridDataSet.Products, ElementName=Window1, Mode=Default}"
Style="{StaticResource {ComponentResourceKey ResourceId=CardViewStyle,
TypeInTargetAssembly={x:Type c1grid:C1DataGrid}}}">
```

**In Code**

To define the card view in your grid, add the following code your project:

- Visual Basic

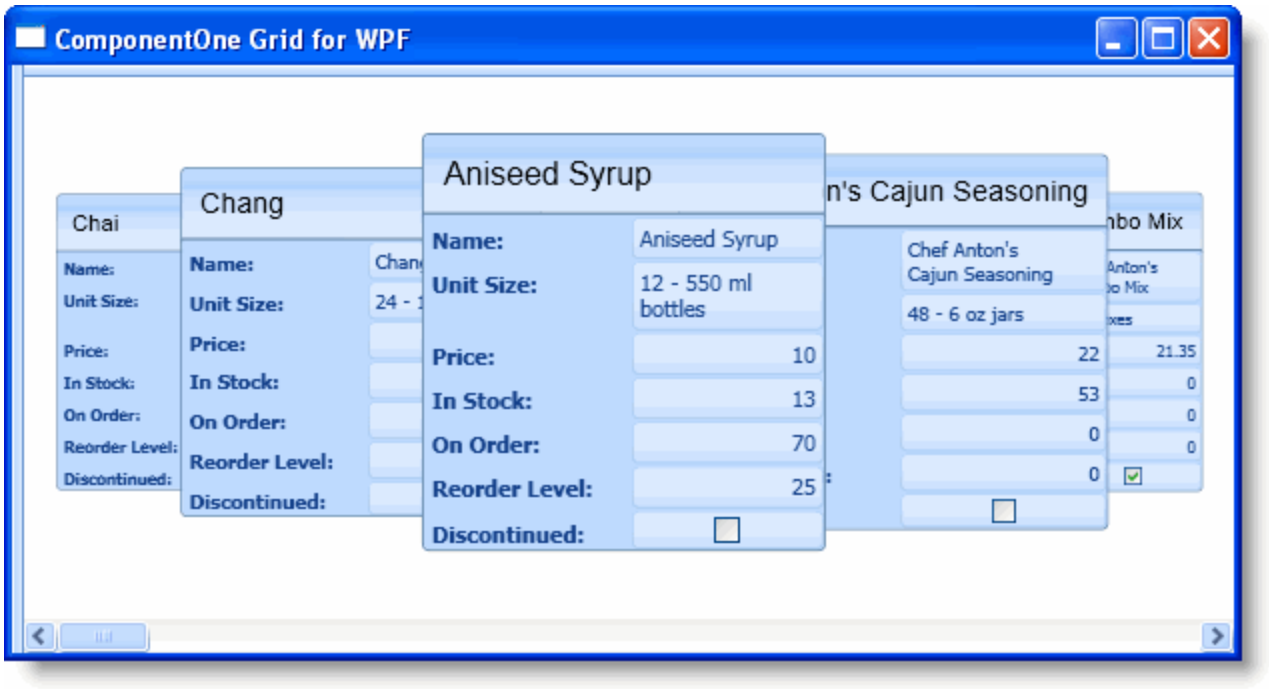
```
C1DataGrid1.Style = TryCast(C1DataGrid1.FindResource(New
ComponentResourceKey(GetType(C1DataGrid), "CardViewStyle")), Style)
```

- C#

```
c1DataGrid1.Style = c1DataGrid1.FindResource(new
ComponentResourceKey(typeof(C1DataGrid), "CardViewStyle")) as Style;
```

### Carousel View

The carousel view is similar to the card view, but each card in this case overlaps and rotates in and out of the main focus at run time though a drag-and-drop operation.




**In XAML**

To specify the carousel view in your grid, add the following `Style` XAML to the `<c1grid:C1DataGrid>` tag so that it appears similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=GridDataSet.Products, ElementName=Window1, Mode=Default}"
Style="{StaticResource {ComponentResourceKey ResourceId=CarouselViewStyle,
TypeInTargetAssembly={x:Type c1grid:C1DataGrid}}}">
```

**In Code**

To specify the carousel view in your grid, add the following code your project:

- Visual Basic

```
C1DataGrid1.Style = TryCast(C1DataGrid1.FindResource(New
ComponentResourceKey(GetType(C1DataGrid), "CarouselViewStyle")), Style)
```

- C#

```
c1DataGrid1.Style = c1DataGrid1.FindResource(new
ComponentResourceKey(typeof(C1DataGrid), "CarouselViewStyle")) as Style;
```

**Custom Views**

You can also create your own views, fully customizing user interface elements including customizing item, header, group by area, and cells of the items, and defining an arbitrary item distribution panel, current item, sort indicators, and much more.

# Using Type Based Column Styles

In the 2009 v1 release of **ComponentOne Grid for WPF**, the ability to format columns by type was added. The TypeBasedColumnStyles and DefaultTypeBasedColumnStyles properties allow you to format how columns appear depending on their DataType – for example, you can format columns differently depending on if the data they contain are strings, Boolean values, integers, and so on.

To format columns, by type, add XAML defining TypeBasedColumnStyles to your grid. The following example maps three styles to columns representing the **System.String**, **System.Double**, and **System.DateTime** data types:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=ProductsDataSet.Products, ElementName=window, Mode=Default}">
    <c1grid:C1DataGrid.TypeBasedColumnStyles>
        <Style TargetType="c1grid:Column" x:Key="{x:Type sys:String}">
            <Setter Property="HeaderCellWidth" Value="100"/>
            <Setter Property="Caption" Value="Description"/>
        </Style>
        <Style TargetType="c1grid:Column" x:Key="{x:Type sys:Double}">
            <Setter Property="HeaderCellWidth" Value="50"/>
            <Setter Property="Format" Value="0.00"/>
        </Style>
        <Style TargetType="c1grid:Column" x:Key="{x:Type sys:DateTime}">
            <Setter Property="HeaderCellWidth" Value="70"/>
            <Setter Property="Format" Value="d"/>
        </Style>
    </c1grid:C1DataGrid.TypeBasedColumnStyles>
</c1grid:C1DataGrid>
```

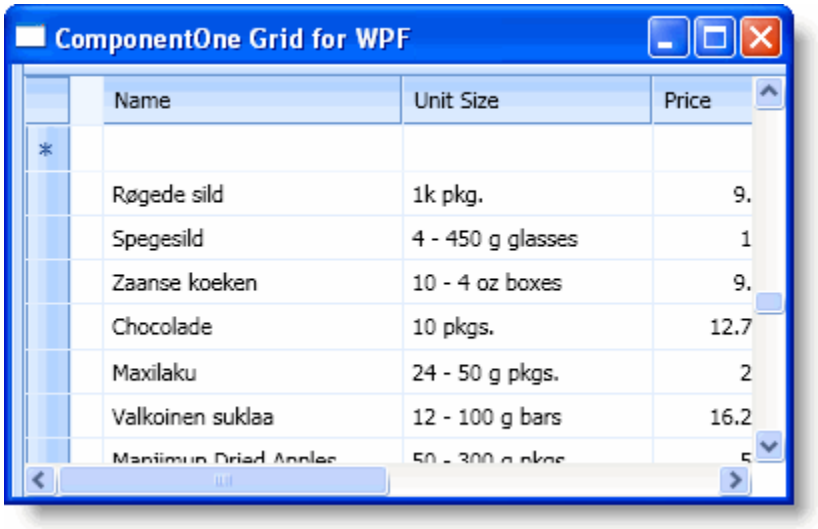
**Note:** You must add a `xmlns:sys="clr-namespace:System;assembly=mscorlib"` namespace declaration to the `<Window>` tag for the above example to work correctly.

# Setting New Row Placement

The new row bar allows end users to add rows to the grid at run time. For more information, see Adding Rows to the Grid (page 68). By default the new row bar is visible and appears in the header of the grid, but if desired you can control where the new row bar is located through the NewRowPlacement property. You can set the NewRowPlacement property to **None**, **InHeader** (default), **InFooter**, **FirstItem**, and **LastItem**.

### NewRowPlacement is set to InHeader

This is the default placement of the new row bar. When the NewRowPlacement property is set to **InHeader**, the new row bar is always visible at the top of the grid even when the grid is scrolled. Notice that the grid below is scrolled but that the new row bar still appears at the top of the grid.




### In XAML

To explicitly set the NewRowPlacement property to **InHeader** in XAML add `NewRowPlacement="InHeader"` to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" NewRowPlacement="InHeader">
```

**In Code**

To explicitly set the NewRowPlacement property to **InHeader** add the following code to your project:

- Visual Basic

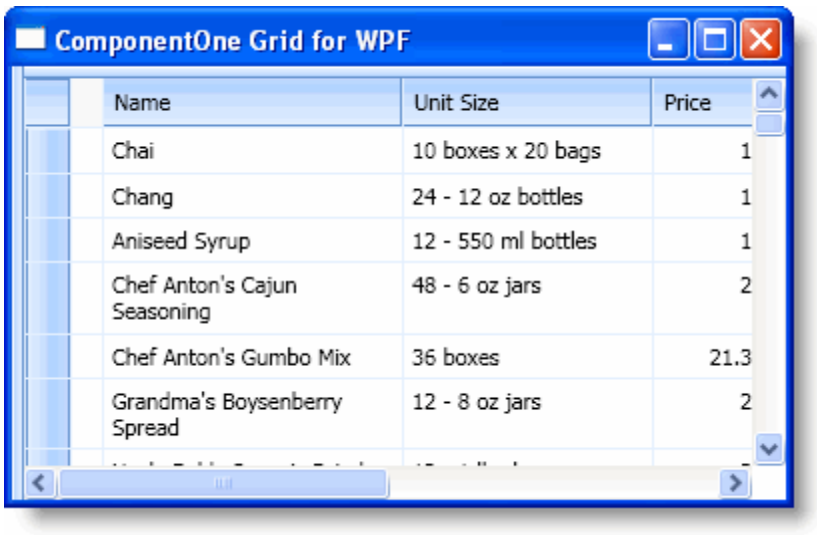
```
C1DataGrid1.NewRowPlacement = NewRowPlacement.InHeader
```

- C#

```
c1DataGrid1.NewRowPlacement = NewRowPlacement.InHeader;
```

### NewRowPlacement is set to None

When the NewRowPlacement property is set to **None**, the new row bar is not visible in the grid and new rows cannot be added at run time through the new row bar.




**In XAML**

To set the NewRowPlacement property to **None** in XAML add `NewRowPlacement="None"` to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" NewRowPlacement="None">
```

**In Code**

To explicitly set the NewRowPlacement property to **None** add the following code to your project:

- Visual Basic

```
C1DataGrid1.NewRowPlacement = NewRowPlacement.None
```

- C#

```
c1DataGrid1.NewRowPlacement = NewRowPlacement.None;
```

### NewRowPlacement is set to InFooter

When the NewRowPlacement property is set to **InFooter**, the new row bar is always visible at the bottom of the grid even when the grid is scrolled.

**In XAML**

To set the NewRowPlacement property to **InFooter** in XAML add `NewRowPlacement="InFooter"` to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" NewRowPlacement="InFooter">
```

**In Code**

To set the NewRowPlacement property to **InFooter** add the following code to your project:

- Visual Basic

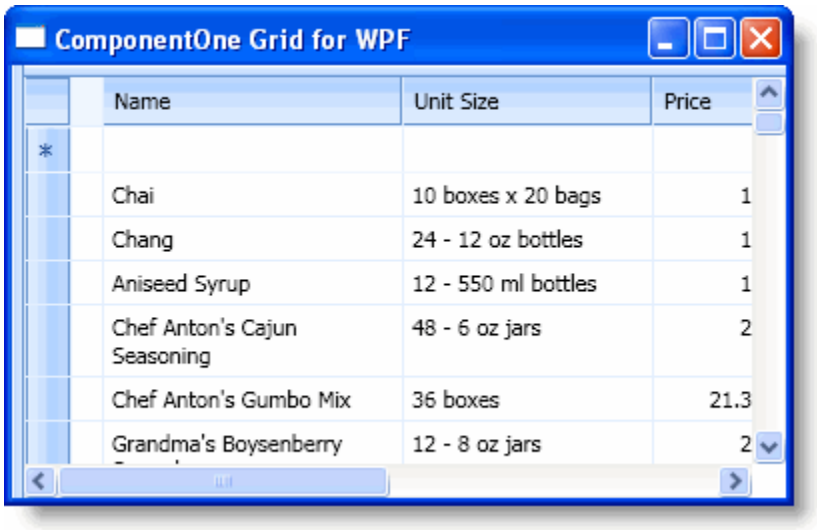
```
C1DataGrid1.NewRowPlacement = NewRowPlacement.InFooter
```

- C#

```
c1DataGrid1.NewRowPlacement = NewRowPlacement.InFooter;
```

### NewRowPlacement is set to FirstItem

When the NewRowPlacement property is set to **FirstItem**, the new row bar appears as the first row in the grid. Because it appears as the first item, when the grid is scrolled it may no longer be visible.




**In XAML**

To set the NewRowPlacement property to **FirstItem** in XAML add `NewRowPlacement="FirstItem"` to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" NewRowPlacement="FirstItem">
```

**In Code**

To set the NewRowPlacement property to **FirstItem** add the following code to your project:

- Visual Basic

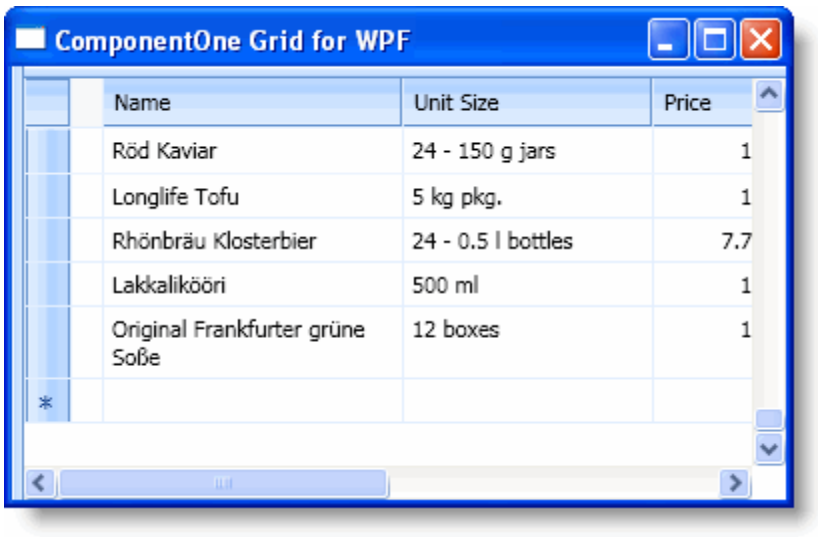
```
C1DataGrid1.NewRowPlacement = NewRowPlacement.FirstItem
```

- C#

```
c1DataGrid1.NewRowPlacement = NewRowPlacement.FirstItem;
```

### NewRowPlacement is set to LastItem

When the NewRowPlacement property is set to **LastItem**, the new row bar appears as the last row in the grid. Because it appears as the last item, when the grid is scrolled it may no longer be visible.




**In XAML**

To set the NewRowPlacement property to **LastItem** in XAML add `NewRowPlacement="LastItem"` to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" NewRowPlacement="LastItem">
```

**In Code**

To explicitly set the NewRowPlacement property to **LastItem** add the following code to your project:

- Visual Basic

```
C1DataGrid1.NewRowPlacement = NewRowPlacement.LastItem
```

- C#

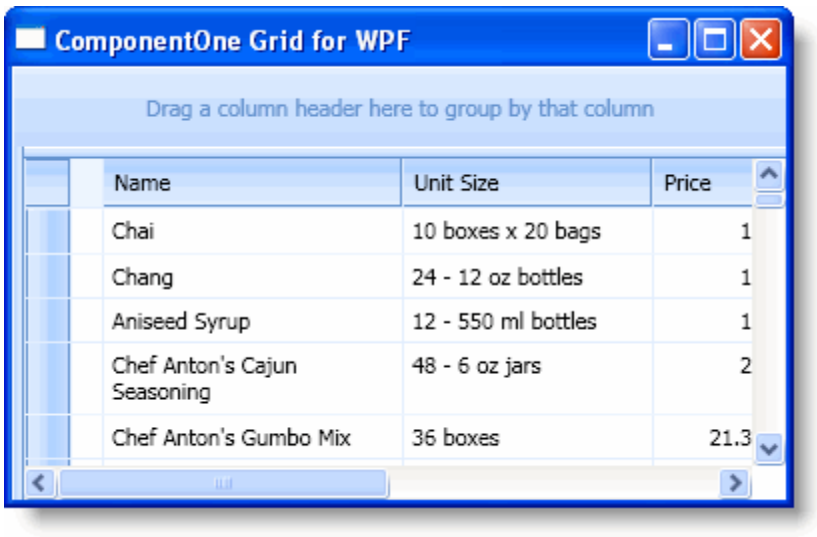
```
c1DataGrid1.NewRowPlacement = NewRowPlacement.LastItem;
```

# Setting GroupBy Area Visibility

The GroupBy area allows end users to group the grid by columns headers. For more information, see Grouping Columns (page 64). By default the GroupBy area is visible, but you can also set the GroupBy area to be hidden or collapsed by using the GroupByVisibility property.

**GroupByVisibility is set to Visible**

When the GroupByVisibility property is set to **Visible**, content can be grouped. This is the default setting.




**In XAML**

You can set the GroupByVisibility property to **Visible** in the Properties window or by adding `GroupByVisibility="Visible"` to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" GroupByVisibility="Visible">
```

**In Code**

To explicitly set the GroupByVisibility property to **Visible** add the following code to your project:

- Visual Basic

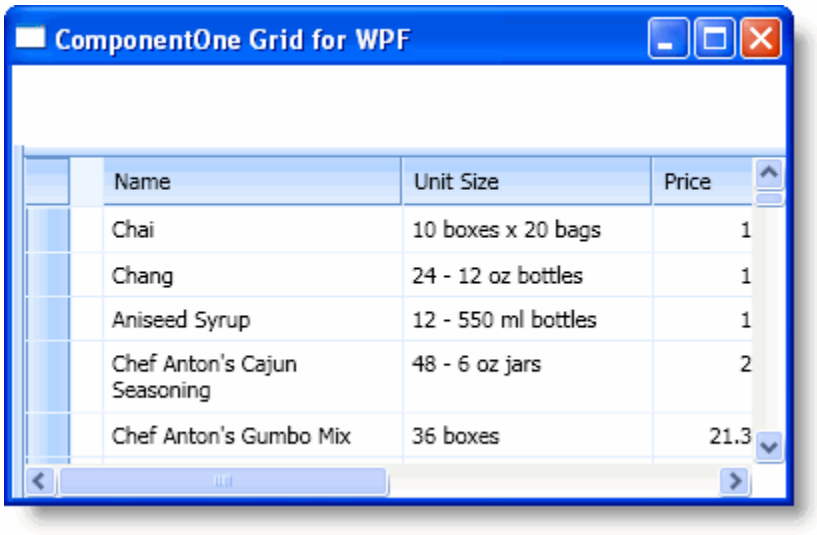
```
C1DataGrid1.GroupByVisibility = Windows.Visibility.Visible
```

- C#

```
c1DataGrid1.GroupByVisibility = Windows.Visibility.Visible;
```

**GroupByVisibility is set to Hidden**

When the GroupByVisibility property is set to **Hidden**, the space where the GroupBy area should appear appears blank and columns cannot be grouped.



**In XAML**

You can set the GroupByVisibility property to **Hidden** in the Properties window or by adding `GroupByVisibility="Hidden"` to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" GroupByVisibility="Hidden">
```

**In Code**

To set the GroupByVisibility property to **Hidden** add the following code to your project:

- Visual Basic

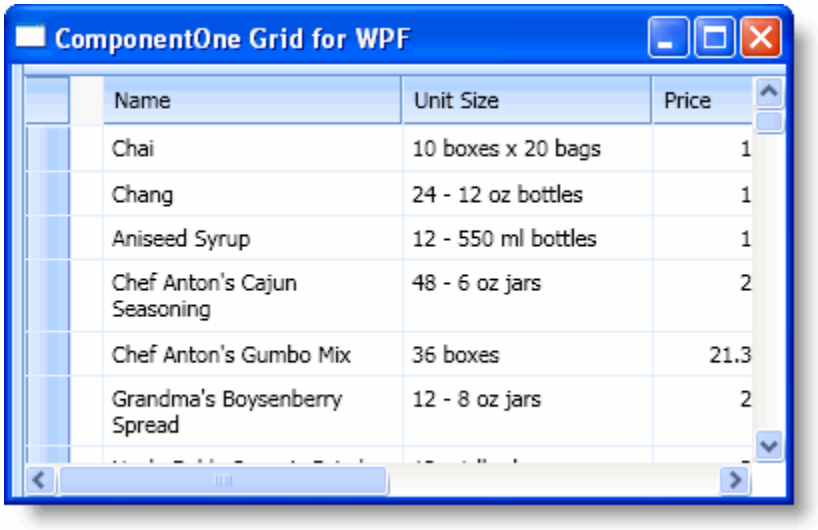
```
C1DataGrid1.GroupByVisibility = Windows.Visibility.Hidden
```

- C#

```
c1DataGrid1.GroupByVisibility = Windows.Visibility.Hidden;
```

**GroupByVisibility is set to Collapsed**

When the GroupByVisibility property is set to **Collapsed**, the GroupBy area and the space where it should appear are collapsed and not visible and columns cannot be grouped.




**In XAML**

You can set the GroupByVisibility property to **Collapsed** in the Properties window or by adding `GroupByVisibility="Collapsed"` to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" GroupByVisibility="Collapsed">
```

**In Code**

To set the GroupByVisibility property to **Collapsed** add the following code to your project:

- Visual Basic

```
C1DataGrid1.GroupByVisibility = Windows.Visibility.Collapsed
```

- C#

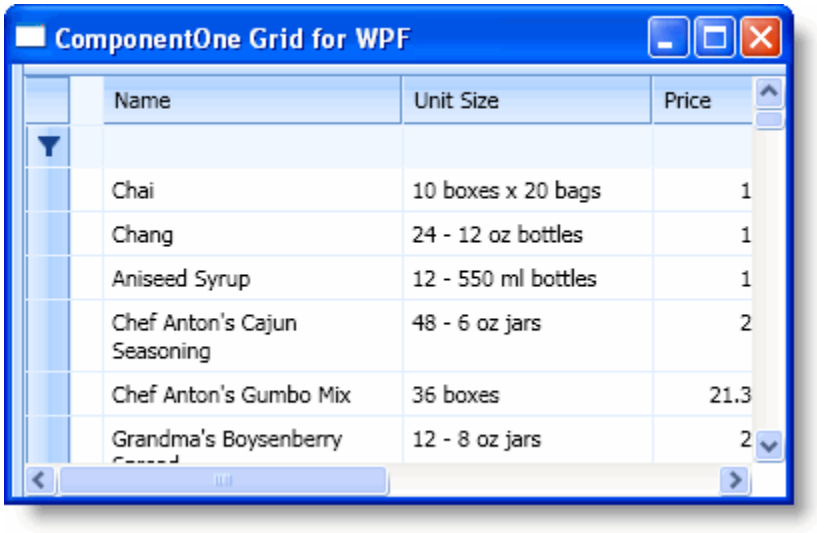
```
c1DataGrid1.GroupByVisibility = Windows.Visibility.Collapsed;
```

# Setting Filter Bar Visibility

The Filter bar allows end users to filter grid content at run time. For more information, see Filtering Columns (page 62). By default the Filter bar is visible, but you can also set the Filter bar to be hidden or collapsed through the FilterBarVisibility property.

**FilterBarVisibility is set to Visible**

When the FilterBarVisibility property is set to **Visible**, content can be filtered. This is the default setting.




**In XAML**

You can set the FilterBarVisibility property to **Visible** in the Properties window or by adding `FilterBarVisibility="Visible"` to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" FilterBarVisibility="Visible">
```

**In Code**

To explicitly set the FilterBarVisibility property to **Visible** add the following code to your project:

- Visual Basic

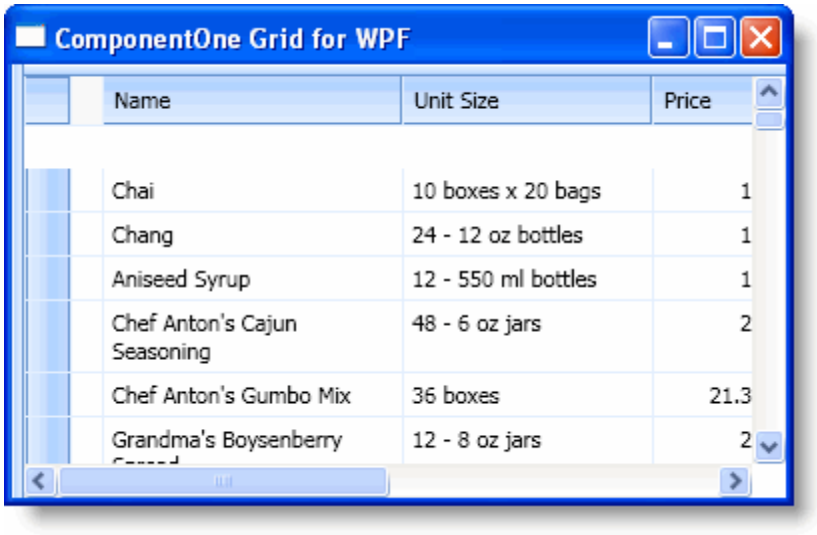
```
C1DataGrid1.FilterBarVisibility = Windows.Visibility.Visible
```

- C#

```
c1DataGrid1.FilterBarVisibility = Windows.Visibility.Visible;
```

**FilterBarVisibility is set to Hidden**

When the FilterBarVisibility property is set to **Hidden**, the space where the filter bar should appear appears blank and content cannot be filtered.




**In XAML**

You can set the FilterBarVisibility property to **Hidden** in the Properties window or by adding `FilterBarVisibility="Hidden"` to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" FilterBarVisibility="Hidden">
```

**In Code**

To set the FilterBarVisibility property to **Hidden** add the following code to your project:

- Visual Basic

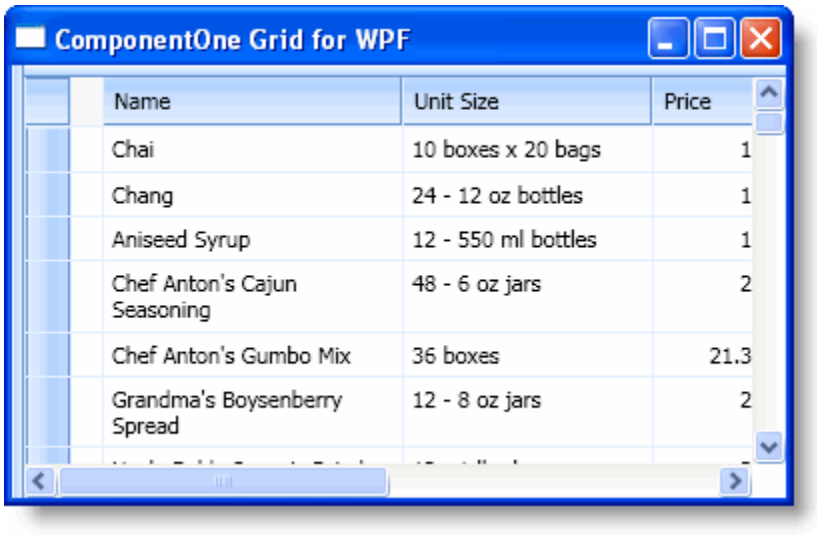
```
C1DataGrid1.FilterBarVisibility = Windows.Visibility.Hidden
```

- C#

```
c1DataGrid1.FilterBarVisibility = Windows.Visibility.Hidden;
```

**FilterBarVisibility is set to Collapsed**

When the FilterBarVisibility property is set to **Collapsed**, the filter bar and the space where it should appear are collapsed and not visible and grid content cannot be filtered.



You can set the FilterBarVisibility property to **Collapsed** in the Properties window or by adding `FilterBarVisibility="Collapsed"` to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" FilterBarVisibility="Collapsed">
```

**In Code**

To set the FilterBarVisibility property to **Collapsed** add the following code to your project:

- Visual Basic

```
C1DataGrid1.FilterBarVisibility = Windows.Visibility.Collapsed
```

- C#

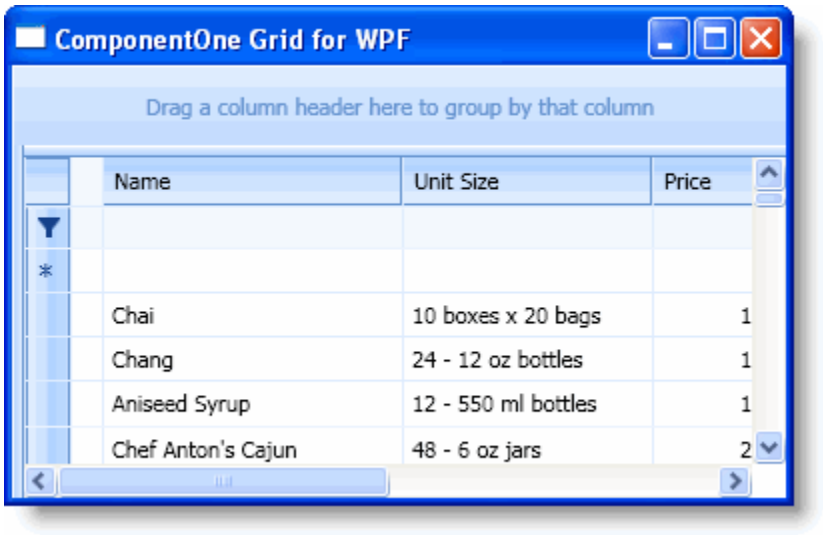
```
c1DataGrid1.FilterBarVisibility = Windows.Visibility.Collapsed;
```

# Setting Header Visibility

By default the column headers are visible, but you can also set the column headers to be hidden or collapsed through the HeaderVisibility property.

**HeaderVisibility is set to Visible**

When the HeaderVisibility property is set to **Visible**, columns can be filtered and reordered via the column headers. This is the default setting.




**In XAML**

You can set the HeaderVisibility property to **Visible** in the Properties window or by adding `HeaderVisibility="Visible"` to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" HeaderVisibility="Visible">
```

**In Code**

To explicitly set the HeaderVisibility property to **Visible**, add the following code to your project:

- Visual Basic

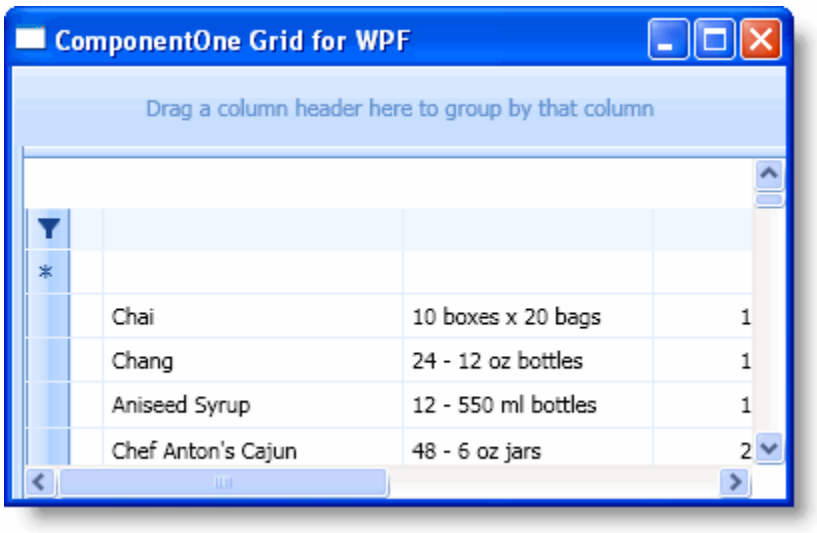
```
C1DataGrid1.HeaderVisibility = Windows.Visibility.Visible
```

- C#

```
c1DataGrid1.HeaderVisibility = Windows.Visibility.Visible;
```

### HeaderVisibility is set to Hidden

When the HeaderVisibility property is set to **Hidden**, the space where the column headers should appear appears blank.




#### In XAML

You can set the HeaderVisibility property to **Hidden** in the Properties window or by adding `HeaderVisibility="Hidden"` to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" HeaderVisibility="Hidden">
```

#### In Code

To set the HeaderVisibility property to **Hidden**, add the following code to your project:

- Visual Basic

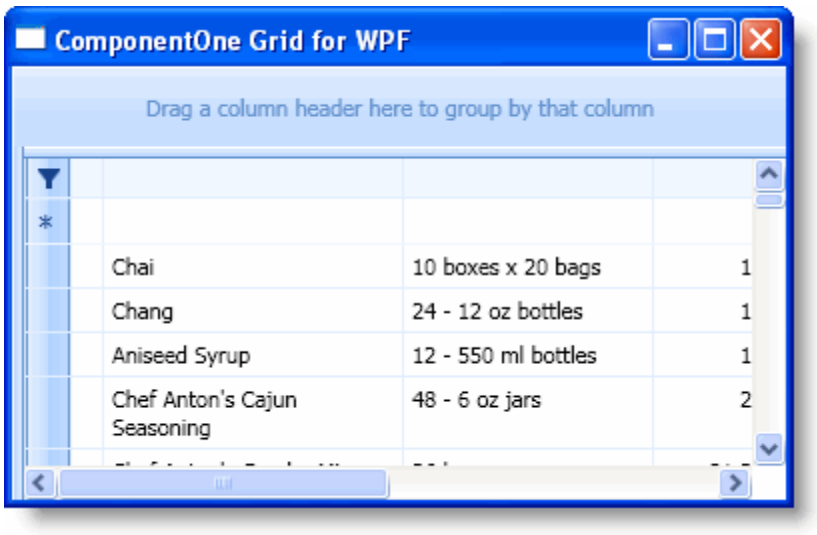
```
C1DataGrid1.HeaderVisibility = Windows.Visibility.Hidden
```

- C#

```
c1DataGrid1.HeaderVisibility = Windows.Visibility.Hidden;
```

### HeaderVisibility is set to Collapsed

When the HeaderVisibility property is set to **Collapsed**, the column headers appear collapsed and not visible and columns cannot be sorted and re-ordered.




**In XAML**

You can set the HeaderVisibility property to **Collapsed** in the Properties window or by adding `HeaderVisibility="Collapsed"` to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" HeaderVisibility="Collapsed">
```

**In Code**

To set the HeaderVisibility property to **Collapsed**, add the following code to your project:

- Visual Basic

```
C1DataGrid1.HeaderVisibility = Windows.Visibility.Collapsed
```

- C#

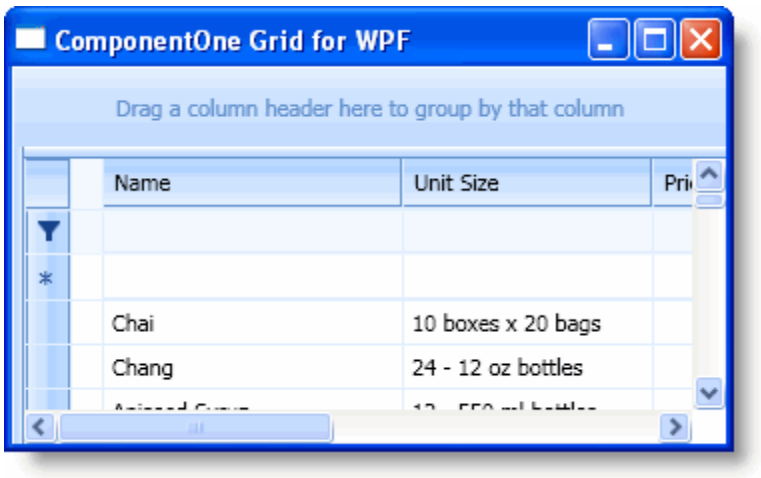
```
c1DataGrid1.HeaderVisibility = Windows.Visibility.Collapsed;
```

# Setting Row Header Visibility

Row headers appear by default on the left side of the grid and include icons for the filter and new row rows as well as an indicator for the current row. By default the row headers are visible, but you can also set the row headers to be hidden or collapsed through the RowHeaderVisibility property.

**RowHeaderVisibility is set to Visible**

When the RowHeaderVisibility property is set to **Visible**, row headers are visible on the grid. This is the default setting.




**In XAML**

You can set the RowHeaderVisibility property to **Visible** in the Properties window or by adding Row`HeaderVisibility="Visible"` to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" RowHeaderVisibility="Visible">
```

**In Code**

To explicitly set the RowHeaderVisibility property to **Visible**, add the following code to your project:

- Visual Basic

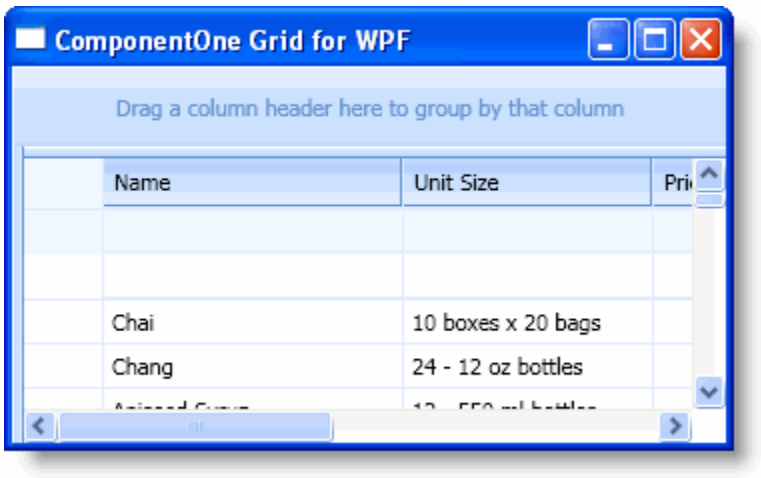
```
C1DataGrid1.RowHeaderVisibility = Windows.Visibility.Visible
```

- C#

```
c1DataGrid1.RowHeaderVisibility = Windows.Visibility.Visible;
```

**RowHeaderVisibility is set to Hidden**

When the RowHeaderVisibility property is set to **Hidden**, the space where the row headers should appear appears blank.




**In XAML**

You can set the RowHeaderVisibility property to **Hidden** in the Properties window or by adding `RowHeaderVisibility="Hidden"` to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" RowHeaderVisibility="Hidden">
```

**In Code**

To set the RowHeaderVisibility property to **Hidden**, add the following code to your project:

- Visual Basic

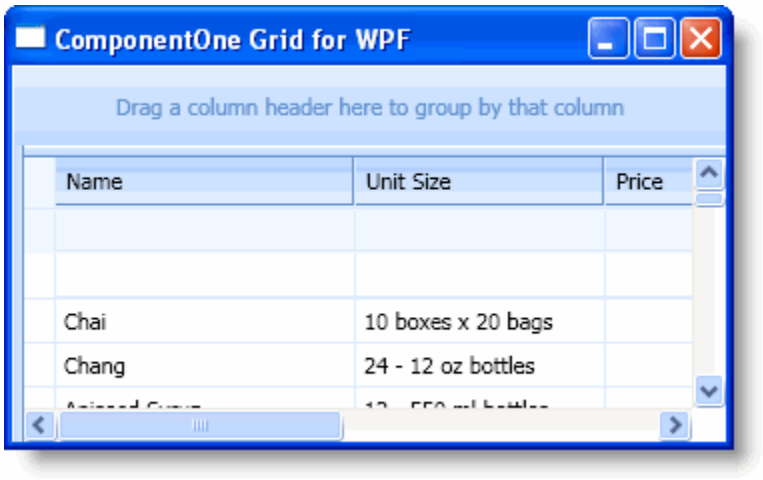
```
C1DataGrid1.RowHeaderVisibility = Windows.Visibility.Hidden
```

- C#

```
c1DataGrid1.RowHeaderVisibility = Windows.Visibility.Hidden;
```

**RowHeaderVisibility is set to Collapsed**

When the RowHeaderVisibility property is set to **Collapsed**, the row headers appear collapsed and not visible.




**In XAML**

You can set the RowHeaderVisibility property to **Collapsed** in the Properties window or by adding `RowHeaderVisibility="Collapsed"` to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" RowHeaderVisibility="Collapsed">
```

**In Code**

To set the RowHeaderVisibility property to **Collapsed**, add the following code to your project:

- Visual Basic

```
C1DataGrid1.RowHeaderVisibility = Windows.Visibility.Collapsed
```

- C#

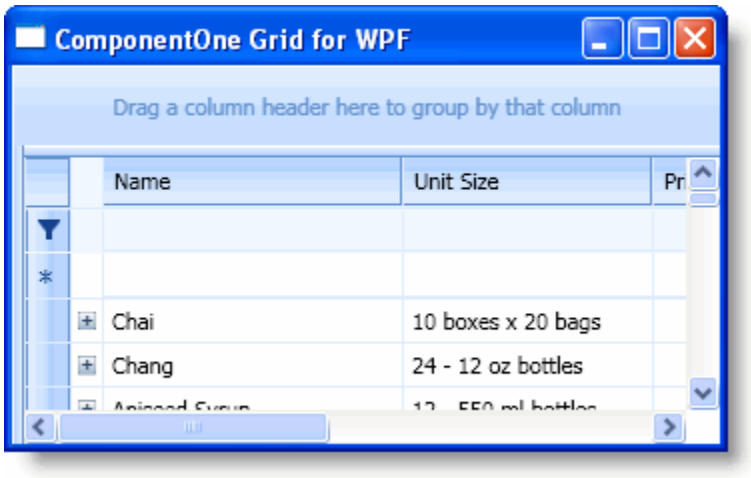
```
c1DataGrid1.RowHeaderVisibility = Windows.Visibility.Collapsed;
```

# Setting Total Bar Visibility

The total bar appears at the bottom of the grid and can be used to display sums, averages, and so on. By default the total bar is collapsed and not visible, but you can set the total bar to be visible or hidden through the TotalBarVisibility property.

**TotalBarVisibility is set to Collapsed**

When the TotalBarVisibility property is set to **Collapsed**, the total bar appears collapsed and not visible. This is the default setting.




**In XAML**

You can set the TotalBarVisibility property to **Collapsed** in the Properties window or by adding `TotalBarVisibility="Collapsed"` to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" TotalBarVisibility="Collapsed">
```

**In Code**

To set the TotalBarVisibility property to **Collapsed**, add the following code to your project:

- Visual Basic

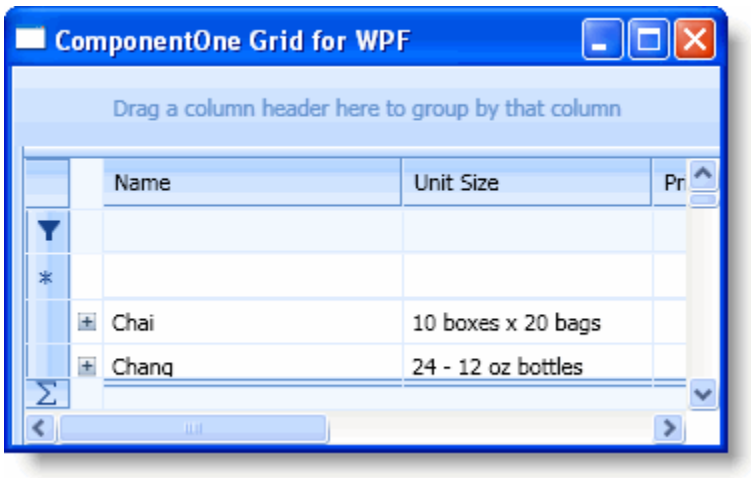
```
C1DataGrid1.TotalBarVisibility = Windows.Visibility.Collapsed
```

- C#

```
c1DataGrid1.TotalBarVisibility = Windows.Visibility.Collapsed;
```

**TotalBarVisibility is set to Visible**

When the TotalBarVisibility property is set to **Visible**, the total bar is visible on the grid. If you're including totals calculation in your grid, set TotalBarVisibility to **Visible**.



**In XAML**

You can set the TotalBarVisibility property to **Visible** in the Properties window or by adding `TotalBarVisibility="Visible"` to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" TotalBarVisibility="Visible">
```

**In Code**

To explicitly set the TotalBarVisibility property to **Visible**, add the following code to your project:

- Visual Basic

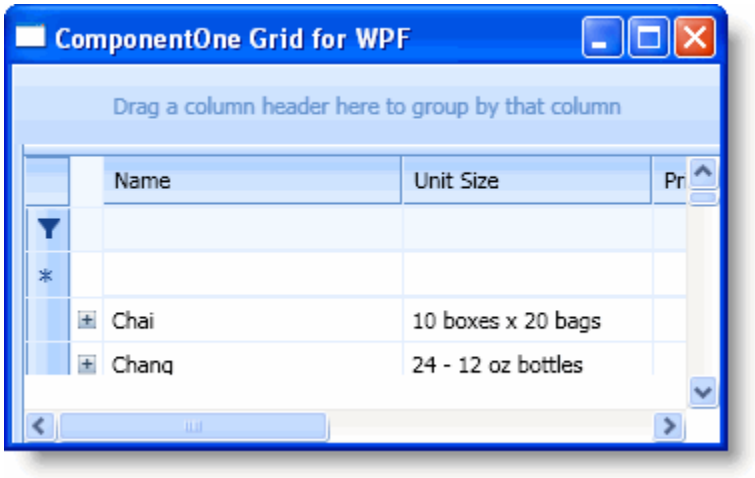
```
C1DataGrid1.TotalBarVisibility = Windows.Visibility.Visible
```

- C#

```
c1DataGrid1.TotalBarVisibility = Windows.Visibility.Visible;
```

**TotalBarVisibility is set to Hidden**

When the TotalBarVisibility property is set to **Hidden**, the space where the total bar should appear appears blank.




**In XAML**

You can set the TotalBarVisibility property to **Hidden** in the Properties window or by adding `TotalBarVisibility="Hidden"` to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" TotalBarVisibility="Hidden">
```

**In Code**

To set the TotalBarVisibility property to **Hidden**, add the following code to your project:

- Visual Basic

```
C1DataGrid1.TotalBarVisibility = Windows.Visibility.Hidden
```

- C#

```
c1DataGrid1.TotalBarVisibility = Windows.Visibility.Hidden;
```

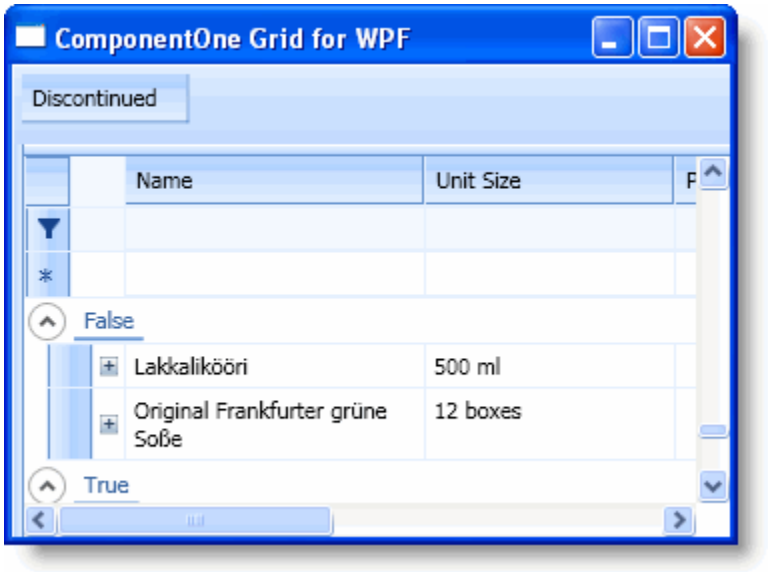
# Setting Group Total Visibility

The group total bar appears at the a group of columns, when the grid is in GroupBy mode and items are grouped together. It can be used to display sums, averages, and so on. By default the group total bar is collapsed and not visible, but you can set the group total bar to be visible or hidden through the

GroupTotalBarVisibility property. Note that in the examples below, the grid has been grouped so that the group total bar is viewable.

### GroupTotalBarVisibility is set to Collapsed

When the GroupTotalBarVisibility property is set to **Collapsed**, the group total bar appears collapsed and not visible. This is the default setting.




#### In XAML

You can set the GroupTotalBarVisibility property to **Collapsed** in the Properties window or by adding `GroupTotalBarVisibility="Collapsed"` to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" GroupTotalBarVisibility="Collapsed">
```

#### In Code

To set the GroupTotalBarVisibility property to **Collapsed**, add the following code to your project:

- Visual Basic

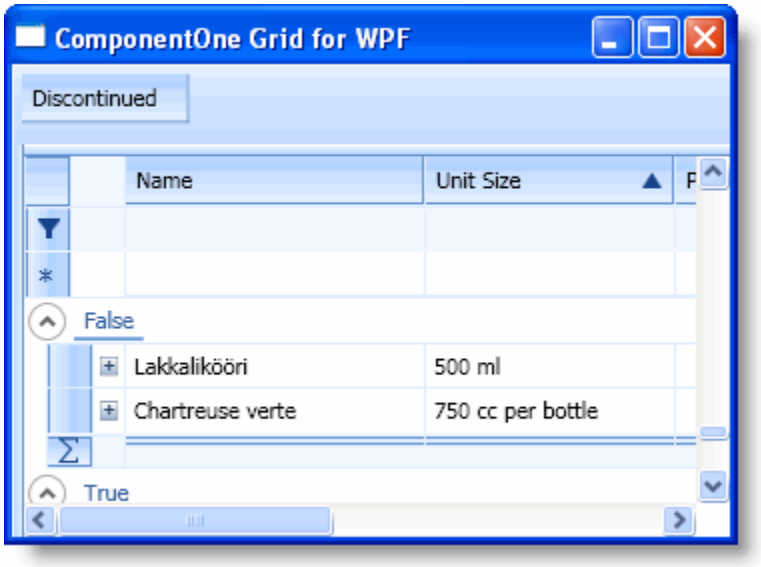
```
C1DataGrid1.GroupTotalBarVisibility = Windows.Visibility.Collapsed
```

- C#

```
c1DataGrid1.GroupTotalBarVisibility = Windows.Visibility.Collapsed;
```

### GroupTotalBarVisibility is set to Visible

When the GroupTotalBarVisibility property is set to **Visible**, the group total bar is visible on the grid. If you're including group totals calculation in your grid, set GroupTotalBarVisibility to **Visible**.




**In XAML**

You can set the GroupTotalBarVisibility property to **Visible** in the Properties window or by adding `GroupTotalBarVisibility="Visible"` to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" GroupTotalBarVisibility="Visible">
```

**In Code**

To explicitly set the GroupTotalBarVisibility property to **Visible**, add the following code to your project:

- Visual Basic

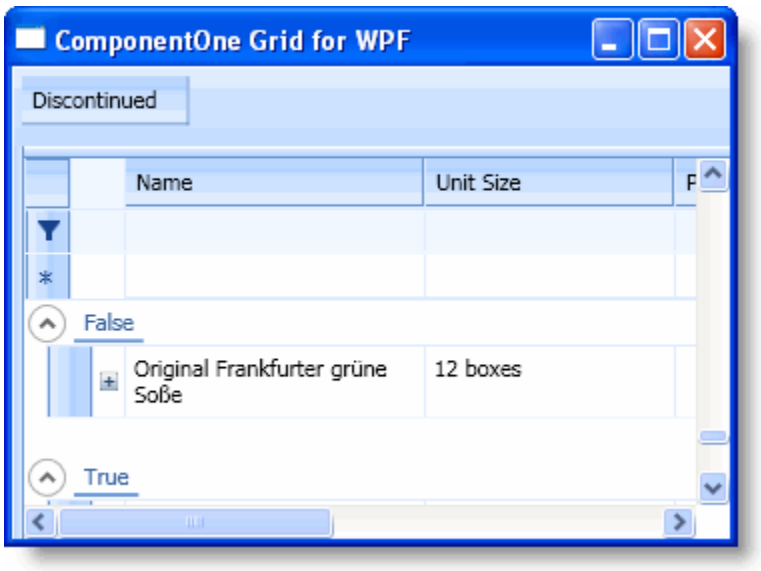
```
C1DataGrid1.GroupTotalBarVisibility = Windows.Visibility.Visible
```

- C#

```
c1DataGrid1.GroupTotalBarVisibility = Windows.Visibility.Visible;
```

**GroupTotalBarVisibility is set to Hidden**

When the GroupTotalBarVisibility property is set to **Hidden**, the space where the group total bar should appear appears blank.




**In XAML**

You can set the GroupTotalBarVisibility property to **Hidden** in the Properties window or by adding `GroupTotalBarVisibility="Hidden"` to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" GroupTotalBarVisibility="Hidden">
```

**In Code**

To set the GroupTotalBarVisibility property to **Hidden**, add the following code to your project:

- Visual Basic

```
C1DataGrid1.GroupTotalBarVisibility = Windows.Visibility.Hidden
```

- C#

```
c1DataGrid1.GroupTotalBarVisibility = Windows.Visibility.Hidden;
```

# Setting Text Alignment

In the 2009 v1 release, **Grid for WPF** added several options for grid text alignment. Now it's easy to specify how grid content will be aligned or justified.

The following alignment properties are included in the grid:

| Class | Member | Description |
|---|---|---|
| C1DataGrid | HeaderCellTextAlignment property | Defines a default text alignment of grid column header captions. |
| C1DataGrid | ItemCellTextAlignment property | Defines a default text alignment of grid item cell values. |
| C1DataGrid | TotalResultCellTextAlignment property | Defines a default text alignment of grid column totals. |
| Column | ActualHeaderCellTextAlignment property | Gets an effective text alignment of column header caption. |
| Column | ActualItemCellTextAlignment property | Gets an effective text alignment of column item cell values. |

| | | |
|---|---|---|
| Column | ActualTotalResultCellTextAlignment property | Gets an effective text alignment of column total result. |
| Column | ItemCellTextAlignment property | Defines a text alignment of column item cell values. |
| Column | HeaderCellTextAlignment property | Defines a text alignment of column header caption. |
| Column | TotalResultCellTextAlignment property | Defines a text alignment of column total result. |
| ContentCellPresenterBase | TextAlignment property | Gets cell's text alignment |

Using the GridTextAlignment enumeration you can justify various elements of the grid by setting their alignment to **General** (default), **Left**, **Right**, **Center**, or **Justify**:

| Option | Description |
|---|---|
| **General** (default) | Text will be left-aligned for string end enumeration types, right aligned for date and numeric types, and centered for Boolean values. |
| **Left** | All text will be left-aligned. |
| **Right** | All text will be right-aligned. |
| **Center** | All text will be centered. |
| **Justify** | All text will be justified and equally spaced. |

# Setting Text Wrapping

In the 2009 v1 release, **Grid for WPF** introduced several options for grid text alignment. Now it's easy to specify how grid content will be wrapped.

The following text wrapping properties are included in the grid:

| Class | Member | Description |
|---|---|---|
| C1DataGrid | HeaderCellTextWrapping property | Defines a default text wrapping of grid column header captions. |
| C1DataGrid | ItemCellTextWrapping property | Defines a default text wrapping for grid item cell values. |
| C1DataGrid | TotalResultCellTextWrapping property | Defines a default text wrapping of grid column totals. |
| Column | ActualHeaderCellTextWrapping property | Gets an effective text wrapping of the column header caption. |
| Column | ActualItemCellTextWrapping property | Gets an effective text wrapping of column item cell values. |
| Column | ActualTotalResultCellTextWrapping property | Gets an effective text wrapping of a column total result. |

| | | |
|---|---|---|
| Column | ItemCellTextWrapping property | Defines a text wrapping of column item cell values. |
| Column | HeaderCellTextWrapping property | Defines a text wrapping of the column header caption. |
| Column | TotalResultCellTextWrapping property | Defines a text wrapping of a column total result. |
| ContentCellPresenterBase | TextWrapping property | Gets cell's text wrapping. |

Using the **TextWrapping** enumeration you can customize how grid context is wrapped by setting the above properties to **WrapWithOverflow**, **Wrap**, or **NoWrap**:

| Option | Description |
|---|---|
| **WrapWithOverflow** | Line-breaking occurs if the line overflows beyond the available block width. However, a line may overflow beyond the block width if the line breaking algorithm cannot determine a line break opportunity, as in the case of a very long word constrained in a fixed-width container with no scrolling allowed. |
| **Wrap** (default) | No line wrapping is performed. |
| **NoWrap** | Line-breaking occurs if the line overflows beyond the available block width, even if the standard line breaking algorithm cannot determine any line break opportunity, as in the case of a very long word constrained in a fixed-width container with no scrolling allowed. |

# Themes and Templates

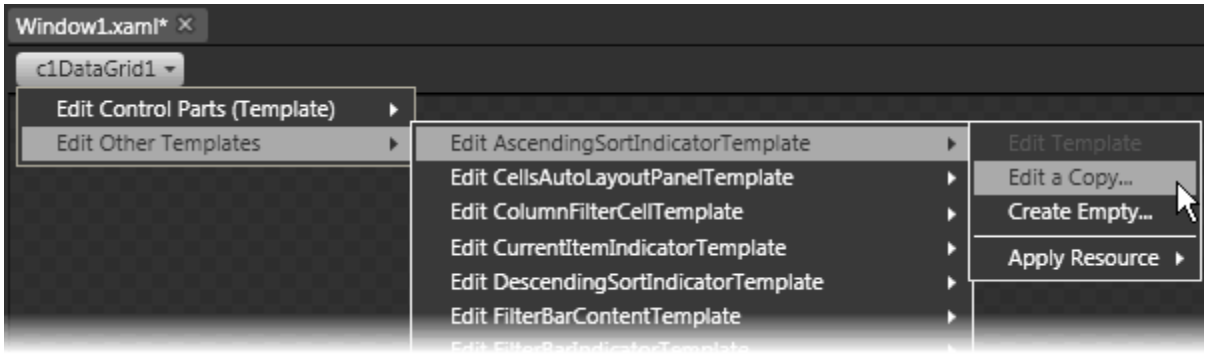
The following topics detail how to customize the C1DataGrid control using themes and templates. You can create a custom theme based on a built-in theme. For more information about built-in themes, see Using Themes (page 71). Themes allow you to customize the appearance of the grid and take advantage of WPF's XAML-based styling. You can also use templates to format and layout the grid and to customize grid actions.

## Editing Templates

One of the main advantages to using a WPF control is that controls are "lookless" with a fully customizable user interface. Just as you design your own user interface (UI), or look and feel, for WPF applications, you can provide your own UI for data managed by **ComponentOne Grid for WPF**. Extensible Application Markup Language (XAML; pronounced "Zammel"), an XML-based declarative language, offers a simple approach to designing your UI without having to write code. **Grid for WPF** includes several templates so that you don't have to begin creating your own UI from scratch .

### Accessing Templates

You can access templates in Microsoft Expression Blend by selecting the C1DataGrid control and, in the DataGrid's menu, selecting **Edit Other Templates**. To create a copy of a template that you can edit, open the C1DataGrid menu, select **Edit Other Templates**, choose the template you wish to edit, and select either **Edit a Copy**, to create an editable copy of the current template, or **Create Empty**, to create a new blank template.




**Note:** If you create a new template through the menu, the template will automatically be linked to that template's property. If you manually create a template in XAML you will have to link the appropriate template property to the template you've created.

### Included Templates

The following templates are included in **Grid for WPF**:

| Template | Description |
|---|---|
| AscendingSortIndicatorTemplate | Gets or sets a template that defines the appearance of a column header's indicator that denotes a column with ascending sort order. This is a dependency property. For an example of using the AscendingSortIndicatorTemplate template, see Changing the Appearance of the Sort Indicator (page 157). |
| CellsAutoLayoutPanelTemplate | Gets or sets a panel that is used to lay out grid item cells representing ordinary (non-list) columns, Filter Bar cells and column headers. This is a dependency property. |
| ColumnFilterCellTemplate | Gets or sets a template that defines the appearence of column cells in the C1DataGrid control's Filter Bar. |
| ColumnFilterCellContentTemplate | Gets or sets a template that defines the UI for Column cell content in the C1DataGrid control's Filter Bar. |
| CurrentItemIndicatorTemplate | Gets or sets a template that defines the appearance of an indicator that denotes a currently selected item. This is a dependency property. |
| DescendingSortIndicatorTemplate | Gets or sets a template that defines the appearance of a column header's indicator that denotes a column with descending sort order. This is a dependency property. For an example of using the DescendingSortIndicatorTemplate template, see Changing the Appearance of the Sort Indicator (page 157). |
| FilterBarContentTemplate | Gets or sets the default template that defines the UI for the content in the Filter Bar. This is a dependency property. |
| FilterBarIndicatorTemplate | Gets or sets a template that defines the appearance of the Filter Bar's indicator that appears next to the filter bar row. This is a dependency property. |
| FilterBarTemplate | Gets or sets the default template that defines the UI for the Filter Bar. This is a dependency property. |
| FixedNewItemCellTemplate | Gets or sets a template that defines the UI for the cells in the grid's add new item row. This is a dependency property. |

| | |
|---|---|
| FixedNewItemTemplate | Gets or sets a template that defines the UI for the grid's add new item row. This is a dependency property. |
| GroupByCellContentTemplate | Gets or sets the default template that defines the UI for the grid column header's content in the GroupBy area. This is a dependency property. |
| GroupByCellsAutoLayoutPanelTemplate | Gets or sets a panel that is used to lay out column headers in the GroupBy area. This is a dependency property. |
| GroupByCellTemplate | Gets or sets the default template that defines the UI for the grid column header in the GroupBy area. This is a dependency property. |
| GroupByContentTemplate | Gets or sets the default template that defines the UI for the grid column header's content in the GroupBy area. This is a dependency property. |
| GroupByTemplate | Gets or sets a template that defines the UI for the grid GroupBy area. This is a dependency property. |
| HeaderCellContentTemplate | Gets or sets the default template that defines the UI for the grid column header's content. This is a dependency property. |
| HeaderCellTemplate | Gets or sets the default template that defines the UI for the grid column header. This is a dependency property. |
| HeaderContentTemplate | Gets or sets a template that defines the UI for a grid header content. This is a dependency property. |
| HeaderTemplate | Gets or sets a template that defines the UI for the grid header. This is a dependency property. |
| ItemCellEditContentTemplate | Gets or sets the default template that defines the UI for editing of a grid item cell value. This is a dependency property. |
| ItemCellShowContentTemplate | Gets or sets the default template that defines the UI for showing of a grid item cell value. This is a dependency property. For an example of using the ItemCellShowContentTemplate template, see Formatting Cells Meeting Specific Criteria (page 162). |
| ItemCellTemplate | Gets or sets the default template that defines the UI for the grid item cell. This is a dependency property. |
| ItemChildListsToggleTemplate | Gets or sets a template defining the appearance of a child list expand/collapse indicator. This is a dependency property. |
| ItemContentTemplate | Gets or sets a template that defines the UI for the grid data item content. This is a dependency property. |
| ItemsPanelTemplate | Gets or sets a panel used to lay out grid data items. This is a dependency property. |
| ItemTemplate | Gets or sets a template that defines the UI for the grid data item. This is a dependency property. |
| ListCellsAutoLayoutPanelTemplate | Gets or sets a panel that is used to lay out grid item cells representing child list columns. This is a dependency property. |
| MoveColumnAfterAdornerTemplate | Gets or sets a template that defines an indicator that denotes a new column location during a column reordering operation performed in the Header Area, in case if the new proposed column order is greater than the current one. This is a dependency property. |

| | |
|---|---|
| MoveColumnBeforeAdornerTemplate | Gets or sets a template that defines an indicator that denotes a new column location during a column reordering operation performed in the Header Area, in case if the new proposed column order is less than the current one. This is a dependency property. |
| MoveGroupByColumnAfterAdornerTemplate | Gets or sets a template that defines an indicator that denotes a new column location during a column reordering operation performed in the GroupBy area, in case if the new proposed column order is greater than the current one. This is a dependency property. |
| MoveGroupByColumnBeforeAdornerTemplate | Gets or sets a template that defines an indicator that denotes a new column location during a column reordering operation performed in the GroupBy area, in case if the new proposed column order is less than the current one. This is a dependency property. |
| NewItemIndicatorTemplate | Gets or sets a template that defines the appearance of an indicator that denotes a grid item intended to add a new data item in an underlying data source. This is a dependency property. |
| SplitCellTemplate | Gets or sets a template that defines the UI for a split cell. This is a dependency property. |
| UniversalItemContentTemplate | Gets or sets a template that is used to define a content layout of different grid parts such as data item, header and Filter Bar by means of a single template. This is a dependency property. For an example of using the UniversalItemContentTemplate template see, Using the UniversalItemContentTemplate (page 160). |

# XAML Elements

Several auxiliary XAML elements are installed with **ComponentOne Grid for WPF**. These elements include templates and themes and are located in the **Studio for WPF** installation directory, by default in the **C1WPFDataGrid\XAML** folder. You can incorporate these elements into your project, for example, to create your own theme based on the included Office 2007 themes. For more information about the built-in themes and data views some of these elements represent, see Using Themes (page 71) and Using Data Views (page 84).

### Included Auxiliary XAML Elements

The following auxiliary XAML elements are included with **Grid for WPF** with their location within the **XAML** default installation folder noted:

| Element | Folder | Description |
|---|---|---|
| generic.xaml | Themes | Specifies the templates for different styles and the initial style of the grid. |
| TypeBasedCellStyles.xaml | Themes\Shared | Specifies the templates for different cell content types such as String, Boolean, and so on. |
| Shared.xaml | Themes\Shared | Specifies templates common to all elements. |
| ItemPresenterStyles.xaml | Themes\Shared | Specifies templates for item presenter styles. |
| SharedIndicators.xaml | Themes\Shared | Specifies the templates for common indicators, such as the sort indicator. |
| CardView.xaml | Themes\Office2007 | Specifies the card view template and settings. |

| | | |
|---|---|---|
| CarouselView.xaml | Themes\Office2007 | Specifies the carousel view template and settings. |
| Common.xaml | Themes\Office2007 | Specifies templates common to all of the built-in themes. |
| Indicators.xaml | Themes\Office2007 | Specifies the templates for common indicators in the built-in themes, such as the sort indicator and filter bar indicator. |
| Office2007.xaml | Themes\Office2007 | Specifies the location of auxiliary data view XAML elements. |
| Splitters.xaml | Themes\Office2007 | Specifies the templates for horizontal and vertical splits. |
| TabularHorizontalView.xaml | Themes\Office2007 | Specifies the tabular horizontal view template and settings. |
| TabularVerticalView.xaml | Themes\Office2007 | Specifies the tabular vertical view template and settings. |
| DuskBlue.xaml | Themes\Office2007\Themes | Specifies the attributes for the DuskBlue theme. |
| DuskGreen.xaml | Themes\Office2007\Themes | Specifies the attributes for the DuskGreen theme. |
| MediaPlayer.xaml | Themes\Office2007\Themes | Specifies the attributes for the MediaPlayer theme. |
| Office2003Blue.xaml | Themes\Office2007\Themes | Specifies the attributes for the Office2003Blue theme. |
| Office2003Classic.xaml | Themes\Office2007\Themes | Specifies the attributes for the Office2003Classic theme. |
| Office2003Olive.xaml | Themes\Office2007\Themes | Specifies the attributes for the Office2003Olive theme. |
| Office2003Royale.xaml | Themes\Office2007\Themes | Specifies the attributes for the Office2003Royale theme. |
| Office2003Silver.xaml | Themes\Office2007\Themes | Specifies the attributes for the Office2003Silver theme. |
| Office2007Black.xaml | Themes\Office2007\Themes | Specifies the attributes for the Office2007Black theme. |
| Office2007Blue.xaml | Themes\Office2007\Themes | Specifies the attributes for the Office2007Blue theme. |
| Office2007Default.xaml | Themes\Office2007\Themes | Specifies the attributes for the Office2007Default theme. The Default theme is very similar to the Office2007Blue theme but does not use Visual Brushes. |
| Office2007Silver.xaml | Themes\Office2007\Themes | Specifies the attributes for the Office2007Silver theme. |
| Vista.xaml | Themes\Office2007\Themes | Specifies the attributes for the Vista theme. |

# Editing Themes

You may choose to customize the appearance of the C1DataGrid control by editing one of the existing built-in themes or elements. If you plan on making several changes to the theme, you may choose to copy the existing XAML file for the theme you wish to customize and add it to your project. For information about included

theme files, see the XAML Elements (page 111) topic. If you only plan to change one or two attributes, a simpler option would be to override those attributes in the project without editing a theme file.

## Adding a Theme File to the Project

Themes are located in the **Studio for WPF** installation directory, by default in the **C1WPFDataGrid\XAML** folder. You can choose to incorporate a customized version of one of these themes into your project with a few simple steps. Note that in this example the **Office2007Default.xaml** file is used.

1. Copy the existing theme file (such as **Office2007Default.xaml**) to the project folder and rename it (for example to **MyTheme.xaml**).
2. In Blend, select **Project | Add Existing Item** to add the XAML file.
3. In the **Add Existing Item** dialog box, select the XAML file you copied to your project folder and select **OK**.

   The file will be added to your project.
4. Add the XAML file to your resource dictionary by adding the following XAML just after the `<Window>` tag:

```
<Window.Resources>
    <ResourceDictionary>
         <!--Replace gridtheme below with a name you choose -->
        <ResourceDictionary x:Key="gridtheme">
            <ResourceDictionary.MergedDictionaries>
                <!--Replace MyTheme.xaml below with the file name you
chose -->
                <ResourceDictionary Source="MyTheme.xaml" />
            </ResourceDictionary.MergedDictionaries>
        </ResourceDictionary>
    </ResourceDictionary>
</Window.Resources>
```

5. Set the C1DataGrid's theme property to the custom theme you just added, for example with the following XAML:

```
<!-- Add XAML to set the Theme property in the C1DataGrid tag --->
<c1grid:C1DataGrid Name="C1DataGrid1" Theme="{DynamicResource
gridtheme}">
```

6. Double-click the **MyTheme.xaml** file, or the file name you choose, in the Project pane in Blend and edit existing values you wish to customize.

   For information about the values that can be customized, see the Resource Keys (page 114) topic.

## Overriding Theme Attributes in the Project

As an alternative to creating or editing a theme file, you can override necessary attributes within the project file itself. This is a better choice when you are only changing a few attributes and do not want to create a new theme file. When overriding existing attributes you would create a resource dictionary based on the theme you wish to edit and set any attributes in that dictionary. For information about the values that can be customized, see the Resource Keys (page 114) topic.

For example, add the following XAML just after the `<Window>` tag to create a theme ResourceDictionary based on **Office2007Silver** theme and override the **C1DataGrid_GridLineVertical_Thickness** and **C1DataGrid_GridLineHorizontal_Thickness** values:

```
<Window.Resources>
       <ResourceDictionary x:Key="overrideTheme">
          <ResourceDictionary.MergedDictionaries>
              <!—Base the resource on the Office2007Silver theme. -->
```

```
            <StaticResourceExtension ResourceKey="{c1grid:C1ThemeKey
TypeInTargetAssembly={x:Type c1grid:C1DataGrid},
ThemeName=Office2007Silver}"/>
        </ResourceDictionary.MergedDictionaries>
        <!-- Override the C1DataGrid_GridLineVertical_Thickness and
C1DataGrid_GridLineHorizontal_Thickness values. -->
        <sys:Double
x:Key="C1DataGrid_GridLineVertical_Thickness">0</sys:Double>
        <sys:Double
x:Key="C1DataGrid_GridLineHorizontal_Thickness">0</sys:Double>
      </ResourceDictionary>
</Window.Resource>
```

This resource dictionary merges a theme dictionary directly from within the grid assembly (by means of the **StaticResource** extension) and directly adds two items to override values from the original dictionary. These two values set the grid to that vertical and horizontal grid lines are not visible.

Assign this dictionary to the Theme property:

```
<!-- Add XAML to set the Theme property in the C1DataGrid tag --->
<c1grid:C1DataGrid Name="C1DataGrid1" Theme="{DynamicResource
overrideTheme}">
```

# Resource Keys

The themes and resources included with **ComponentOne Grid for WPF** have several incorporated resource keys. These keys include brush, border, and other elements and can be customized to represent a unique appearance. When customizing a theme, a resource key that is not explicitly specified will revert to the default value. The included resource keys and their descriptions are noted in the following topics.

## Grid and Cell Resource Keys

The following tables describe grid and cell resource keys:

### Grid Background Resource Key

The following grid background resource key is included:

| Resource Key | Description |
|---|---|
| C1DataGrid_Brush | Represents the C1DataGrid background brush. |

### Grid Line Resource Keys

The following grid line resource keys are included:

| Resource Key | Description |
|---|---|
| C1DataGrid_GridLineVertical_Brush | Represents the vertical grid line's brush. |
| C1DataGrid_GridLineVertical_Thickness | Represents the vertical grid line's thickness. |
| C1DataGrid_GridLineHorizontal_Brush | Represents the horizontal grid line's brush. |
| C1DataGrid_GridLineHorizontal_Thickness | Represents the horizontal grid line's thickness. |

### Cell Resource Keys

The following  cell resource keys are included:

| Resource Key | Description |
|---|---|
| C1DataGrid_Cell_Brush | Represents the cell's background brush. |
| C1DataGrid_Cell_HoverBrush | Represents the cell's background brush when the pointer is over the cell. |
| C1DataGrid_Cell_SelectedBrush | Represents the selected cell's background brush. |
| C1DataGrid_Cell_SelectedHoverBrush | Represents the selected cell's background brush when the pointer is over the cell. |
| C1DataGrid_CellBorder_Brush | Represents the cell's border brush. |
| C1DataGrid_CellBorder_HoverBrush | Represents the cell's border brush when the pointer is over the cell. |
| C1DataGrid_CellBorder_SelectedBrush | Represents the selected cell's border brush. |
| C1DataGrid_CellBorder_SelectedHoverBrush | Represents the selected cell's border brush when the pointer is over the cell. |
| C1DataGrid_CellBorder_Thickness | Represents the cell's border thickness (all 4 edges). |
| C1DataGrid_CellBorder_CornerRadius | Represents the cell's border corner radius (all 4 corners). |
| C1DataGrid_CellText_Brush | Represents the cell's text brush. |
| C1DataGrid_CellText_HoverBrush | Represents the cell's text brush when the pointer is over the cell. |
| C1DataGrid_CellText_SelectedBrush | Represents the selected cell's text brush. |
| C1DataGrid_CellText_SelectedHoverBrush | Represents the selected cell's text brush when the pointer is over the cell. |

## Hierarchy and Child Grid Resource Keys

The following tables describe hierarchy and child grid resource keys:

### Hierarchy Expander Resource Keys

The following hierarchy expander resource keys are included:

| Resource Key | Description |
|---|---|
| C1DataGrid_HierarchyExpander_Brush | Represents the hierarchy expander's background brush. |
| C1DataGrid_HierarchyExpander_Foreground | Represents the hierarchy expander's foreground brush. |

### Child Toggle Resource Keys

The following child toggle resource keys are included:

| Resource Key | Description |
|---|---|
| C1DataGrid_ChildToggleHeader_Brush | Represents the child list toggle column header's background brush. |
| C1DataGrid_ChildToggleHeader_HoverBrush | Represents the child list toggle column header's background brush when the pointer is over the child toggle. |

| | |
|---|---|
| C1DataGrid_ChildToggleHeaderBorder_Brush | Represents the child list toggle column header's border brush. |
| C1DataGrid_ChildToggleHeaderBorder_HoverBrush | Represents the child list toggle column header's border brush when the pointer is over the child toggle. |
| C1DataGrid_ChildToggleHeaderBorder_Thickness | Represents the child list toggle column header's border thickness (all 4 edges). |
| C1DataGrid_ChildToggleHeaderBorder_CornerRadius | Represents the child list toggle column header's border corner radius (all 4 corners). |
| C1DataGrid_ChildToggleHeaderText_Brush | Represents the child list toggle column header's text brush. |
| C1DataGrid_ChildToggleHeaderText_HoverBrush | Represents the child list toggle column header's text brush when the pointer is over the child toggle. |
| C1DataGrid_ChildToggleText_Brush | Represents the child list toggle cell text's brush. |
| C1DataGrid_ChildToggleBorder_Brush | Represents the child list toggle cell border's brush. |
| C1DataGrid_ChildToggleBorder_HoverBrush | Represents the child list toggle cell border's brush when the pointer is over the child toggle. |
| C1DataGrid_ChildToggleBorder_Thickness | Represents the child list toggle cell border's thickness (all 4 edges). |
| C1DataGrid_ChildToggleBorder_CornerRadius | Represents the child list toggle cell border's corner radius (all 4 corners). |

## Column Header Resource Keys

The following tables describe column header resource keys:

### Column Header Background Resource Keys

The following column header background resource keys are included:

| Resource Key | Description |
|---|---|
| C1DataGrid_ColumnHeader_Brush | Represents the column header's background brush. |
| C1DataGrid_ColumnHeader_HoverBrush | Represents the column header's background brush when the pointer is over the column header. |
| C1DataGrid_ColumnHeader_SelectedBrush | Represents the selected column header's background brush. |
| C1DataGrid_ColumnHeader_SelectedHoverBrush | Represents the selected column header's background brush when the pointer is over the column header. |
| C1DataGrid_ColumnHeader_ActiveBrush | Represents the column header's background brush when activated by an item or row hover or selection. |
| C1DataGrid_ColumnHeader_ActiveHoverBrush | Represents the column header's background brush when activated by an item or row hover or selection when the pointer is over the column header. |

### Column Header Border Resource Keys

The following column header border resource keys are included:

| Resource Key | Description |
|---|---|
| C1DataGrid_ColumnHeaderBorder_Brush | Represents the column header's border brush. |
| C1DataGrid_ColumnHeaderBorder_HoverBrush | Represents the column header's border brush when the pointer is over the column header. |
| C1DataGrid_ColumnHeaderBorder_SelectedBrush | Represents the selected column header's border brush. |
| C1DataGrid_ColumnHeaderBorder_SelectedHoverBrush | Represents the selected column header's border brush when the pointer is over the column header. |
| C1DataGrid_ColumnHeaderBorder_ActiveBrush | Represents the column header's border brush when activated by an Item or Row hover or selection. |
| C1DataGrid_ColumnHeaderBorder_ActiveHoverBrush | Represents the column header's border brush when activated by an Item or row hover or selection when the pointer is over the column header. |
| C1DataGrid_ColumnHeaderBorder_Thickness | Represents the column header's border thickness (all 4 edges). |
| C1DataGrid_ColumnHeaderBorder_CornerRadius | Represents the column header's border corner radius (All 4 corners). |

### Column Header Text Resource Keys

The following column header text resource keys are included::

| Resource Key | Description |
|---|---|
| C1DataGrid_ColumnHeaderText_Brush | Represents the column header's text brush. |
| C1DataGrid_ColumnHeaderText_HoverBrush | Represents the column header's text brush when the pointer is over the column header. |
| C1DataGrid_ColumnHeaderText_SelectedBrush | Represents the selected column header's text brush. |
| C1DataGrid_ColumnHeaderText_SelectedHoverBrush | Represents the selected column header's text brush when the pointer is over the column header. |
| C1DataGrid_ColumnHeaderText_ActiveBrush | Represents the column header's text brush when activated by an item or row hover or selection. |
| C1DataGrid_ColumnHeaderText_ActiveHoverBrush | Represents the column header's text brush when activated by an item or row hover or selection when the pointer is over the column header. |

## GroupBy Area and Grid Row Group Header Resource Keys

The following tables describe GroupBy area and grid row group header resource keys:

### GroupBy Area Column Header Resource Keys

The following column header resource keys are included:

| Resource Key | Description |
|---|---|
| C1DataGrid_ColumnHeaderGroupArea_Brush | Represents the GroupBy area's column header background brush. |

| | |
|---|---|
| C1DataGrid_ColumnHeaderGroupArea_HoverBrush | Represents the GroupBy area's column header background brush when the pointer is over the GroupBy area. |
| C1DataGrid_ColumnHeaderGroupArea_SelectedBrush | Represents the selected GroupBy area's column header background brush. |
| C1DataGrid_ColumnHeaderGroupArea_SelectedHoverBrush | Represents the selected GroupBy area's column header background brush when the pointer is over the GroupBy area. |
| C1DataGrid_ColumnHeaderGroupArea_ActiveBrush | Represents the GroupBy area's column header background brush when activated by an item or row hover or selection. |
| C1DataGrid_ColumnHeaderGroupArea_ActiveHoverBrush | Represents the GroupBy area's column header background brush when activated by an item or row hover or selection when the pointer is over the GroupBy area. |
| C1DataGrid_ColumnHeaderGroupAreaBorder_Brush | Represents the GroupBy area's column header border brush. |
| C1DataGrid_ColumnHeaderGroupAreaBorder_HoverBrush | Represents the GroupBy area's column header border brush when the pointer is over the GroupBy area. |
| C1DataGrid_ColumnHeaderGroupAreaBorder_SelectedBrush | Represents the selected GroupBy area's column header border brush. |
| C1DataGrid_ColumnHeaderGroupAreaBorder_SelectedHoverBrush | Represents the selected GroupBy area's column header border brush when the pointer is over the GroupBy area. |
| C1DataGrid_ColumnHeaderGroupAreaBorder_ActiveBrush | Represents the GroupBy area's column header border brush when activated by an item or row hover or selection. |
| C1DataGrid_ColumnHeaderGroupAreaBorder_ActiveHoverBrush | Represents the GroupBy area's column header border brush when activated by an item or row hover or selection when the pointer is over the GroupBy area. |
| C1DataGrid_ColumnHeaderGroupAreaBorder_Thickness | Represents the GroupBy area's column header border thickness (all 4 edges). |
| C1DataGrid_ColumnHeaderGroupAreaBorder_CornerRadius | Represents the GroupBy area's column header border corner radius (all 4 corners). |
| C1DataGrid_ColumnHeaderGroupAreaText_Brush | Represents the GroupBy area's column header text brush. |
| C1DataGrid_ColumnHeaderGroupAreaText_HoverBrush | Represents the GroupBy area's column header text brush when the pointer is over the GroupBy area. |
| C1DataGrid_ColumnHeaderGroupAreaText_SelectedBrush | Represents the selected GroupBy area's column header text brush. |
| C1DataGrid_ColumnHeaderGroupAreaText_SelectedHoverBrush | Represents the selected GroupBy area's column header text brush when the pointer is over the GroupBy area. |
| C1DataGrid_ColumnHeaderGroupAreaText_ActiveBrush | Represents the GroupBy area's column header text brush when activated by an item or row hover or selection. |
| C1DataGrid_ColumnHeaderGroupAreaText_ActiveHoverBrush | Represents the GroupBy area's column header text brush when activated by an item or row hover or selection when the pointer is over the GroupBy area. |

### GroupBy Area Resource Keys

The following GroupBy area header resource keys are included:

| Resource Key | Description |
|---|---|
| C1DataGrid_GroupArea_Brush | Represents the GroupBy area's background brush. |
| C1DataGrid_GroupAreaBorder_Brush | Represents the GroupBy area's border brush. |
| C1DataGrid_GroupAreaBorder_Thickness | Represents the GroupBy area's border thickness (all 4 edges). |
| C1DataGrid_GroupAreaBorder_CornerRadius | Represents the GroupBy area's border corner radius (all 4 corners). |
| C1DataGrid_GroupAreaText_Brush | Represents the GroupBy area's text brush. |
| C1DataGrid_GroupAreaConnector_Pen | Represents the GroupBy area's column connector pen. |

### Grid Area Group Header Resource Keys

The following grid area group header resource keys are included:

| Resource Key | Description |
|---|---|
| C1DataGrid_GroupHeader_Brush | Represents the grid row group header's background brush when in GroupBy mode. |
| C1DataGrid_GroupHeader_HoverBrush | Represents the grid row group header's background brush when in GroupBy mode when the pointer is over the GroupBy area. |
| C1DataGrid_GroupHeaderBorder_Brush | Represents the grid row group header's border brush when in GroupBy mode. |
| C1DataGrid_GroupHeaderBorder_HoverBrush | Represents the grid row group header's border brush when in GroupBy mode when the pointer is over the GroupBy area. |
| C1DataGrid_GroupHeaderBorder_Thickness | Represents the grid row group header's border thickness (all 4 edges) when in GroupBy mode. |
| C1DataGrid_GroupHeaderBorder_CornerRadius | Represents the grid row group header's border corner radius (all 4 corners) when in GroupBy mode. |
| C1DataGrid_GroupHeaderText_Brush | Represents the grid row group header's text brush when in GroupBy mode. |
| C1DataGrid_GroupHeaderText_HoverBrush | Represents the grid row group header's text brush when in GroupBy mode when the pointer is over the GroupBy area. |

## Header Item Resource Keys

The table below describes header item resource keys.

### Header Item Resource Keys

The following header item resource keys are included:

| Resource Key | Description |
|---|---|
| C1DataGrid_HeaderItemHeader_Brush | Represents the header item's header background brush. |
| C1DataGrid_HeaderItemHeader_HoverBrush | Represents the header item's header background brush when the pointer is over the header. |

| | |
|---|---|
| C1DataGrid_HeaderItemHeaderBorder_Brush | Represents the header item's header border brush. |
| C1DataGrid_HeaderItemHeaderBorder_HoverBrush | Represents the header item's header border brush when the pointer is over the header. |
| C1DataGrid_HeaderItemHeaderBorder_Thickness | Represents the header item's header border thickness (all 4 edges). |
| C1DataGrid_HeaderItemHeaderBorder_CornerRadius | Represents the header item's header border corner radius (all 4 corners). |
| C1DataGrid_HeaderItemHeaderText_Brush | Represents the header item's header text brush. |
| C1DataGrid_HeaderItemHeaderText_HoverBrush | Represents the header item's header text brush when the pointer is over the header. |

## Row and Row Header Resource Keys

The following tables describe grid row and row header resource keys:

### Row Resource Keys

The following row resource keys are included:

| Resource Key | Description |
|---|---|
| C1DataGrid_Row_Brush | Represents the row's background brush. |
| C1DataGrid_Row_HoverBrush | Represents the row's background brush when the pointer is over the row. |
| C1DataGrid_Row_SelectedBrush | Represents the selected row's background brush. |
| C1DataGrid_Row_SelectedHoverBrush | Represents the selected row's background brush when the pointer is over the selected row. |

### Row Header Resource Keys

The following row header resource keys are included:

| Resource Key | Description |
|---|---|
| C1DataGrid_RowHeader_Brush | Represents the row header's background brush. |
| C1DataGrid_RowHeader_HoverBrush | Represents the row header's background brush when the pointer is over the row header. |
| C1DataGrid_RowHeader_SelectedBrush | Represents the selected row header's background brush. |
| C1DataGrid_RowHeader_SelectedHoverBrush | Represents the selected row header's background brush when the pointer is over the row header. |
| C1DataGrid_RowHeader_ActiveBrush | Represents the row header's background brush when activated by an item or column hover or selection. |
| C1DataGrid_RowHeader_ActiveHoverBrush | Represents the row header's background brush when activated by an item or column hover or selection when the pointer is over the row header. |
| C1DataGrid_RowHeaderBorder_Brush | Represents the row header's border brush. |
| C1DataGrid_RowHeaderBorder_HoverBrush | Represents the row header's border brush when the pointer is |

| | |
|---|---|
| | over the row header. |
| C1DataGrid_RowHeaderBorder_SelectedBrush | Represents the selected row header's border brush. |
| C1DataGrid_RowHeaderBorder_SelectedHoverBrush | Represents the selected row header's border brush when the pointer is over the row header. |
| C1DataGrid_RowHeaderBorder_ActiveBrush | Represents the row header's border brush when activated by an item or column hover or selection. |
| C1DataGrid_RowHeaderBorder_ActiveHoverBrush | Represents the row header's border brush when activated by an item or column hover or selection when the pointer is over the row header. |
| C1DataGrid_RowHeaderBorder_Thickness | Represents the row header's border thickness (all 4 edges). |
| C1DataGrid_RowHeaderBorder_CornerRadius | Represents the row header's border corner radius (all 4 corners). |
| C1DataGrid_RowHeaderText_Brush | Represents the row header's text brush. |
| C1DataGrid_RowHeaderText_HoverBrush | Represents the row header's text brush when the pointer is over the row header. |
| C1DataGrid_RowHeaderText_SelectedBrush | Represents the selected row header's text brush. |
| C1DataGrid_RowHeaderText_SelectedHoverBrush | Represents the selected row header's text brush when the pointer is over the row header. |
| C1DataGrid_RowHeaderText_ActiveBrush | Represents the row header's text brush when activated by an item or column hover or selection. |
| C1DataGrid_RowHeaderText_ActiveHoverBrush | Represents the row header's text brush when activated by an item or column hover or selection when the pointer is over the row header. |

## Split Resource Keys

The following tables describe resource keys related to vertical and horizontal splits:

### Vertical Split Resource Keys

The following vertical split resource keys are included:

| Resource Key | Description |
|---|---|
| C1DataGrid_SplitVerticalTop_Brush | Represents the vertical splitter's left top brush. |
| C1DataGrid_SplitVerticalTop_HoverBrush | Represents the vertical splitter's top line brush when the pointer is over the splitter bar. |
| C1DataGrid_SplitVerticalTop_SelectedBrush | Represents the selected vertical splitter's top line brush. |
| C1DataGrid_SplitVerticalTop_Thickness | Represents the vertical splitter's top line thickness. |
| C1DataGrid_SplitVerticalCenter_Brush | Represents the vertical splitter's center line brush. |
| C1DataGrid_SplitVerticalCenter_HoverBrush | Represents the vertical splitter's center line brush when the pointer is over the splitter bar. |
| C1DataGrid_SplitVerticalCenter_SelectedBrush | Represents the selected vertical splitter's center line brush. |

| | |
|---|---|
| C1DataGrid_SplitVerticalCenter_Thickness | Represents the vertical splitter's center line thickness. |
| C1DataGrid_SplitVerticalBottom_Brush | Represents the vertical splitter's bottom line brush. |
| C1DataGrid_SplitVerticalBottom_HoverBrush | Represents the vertical splitter's bottom line brush when the pointer is over the splitter bar. |
| C1DataGrid_SplitVerticalBottom_SelectedBrush | Represents the selected vertical splitter's bottom line brush. |
| C1DataGrid_SplitVerticalBottom_Thickness | Represents the vertical splitter's bottom line thickness. |

### Horizontal Split Resource Keys

The following horizontal split resource keys are included:

| Resource Key | Description |
|---|---|
| C1DataGrid_SplitHorizontalLeft_Brush | Represents the horizontal splitter's left line brush. |
| C1DataGrid_SplitHorizontalLeft_HoverBrush | Represents the horizontal splitter's left line brush on when the pointer is over the splitter bar. |
| C1DataGrid_SplitHorizontalLeft_SelectedBrush | Represents the selected horizontal splitter's left line brush. |
| C1DataGrid_SplitHorizontalLeft_Thickness | Represents the horizontal splitter's left line thickness. |
| C1DataGrid_SplitHorizontalCenter_Brush | Represents the horizontal splitter's center line brush. |
| C1DataGrid_SplitHorizontalCenter_HoverBrush | Represents the horizontal splitter's center line brush when the pointer is over the splitter bar. |
| C1DataGrid_SplitHorizontalCenter_SelectedBrush | Represents the selected horizontal splitter's center line brush. |
| C1DataGrid_SplitHorizontalCenter_Thickness | Represents the horizontal splitter's center line thickness. |
| C1DataGrid_SplitHorizontalRight_Brush | Represents the horizontal splitter's right line brush. |
| C1DataGrid_SplitHorizontalRight_HoverBrush | Represents the horizontal splitter's right line brush when the pointer is over the splitter bar. |
| C1DataGrid_SplitHorizontalRight_SelectedBrush | Represents the selected horizontal's splitter right line brush. |
| C1DataGrid_SplitHorizontalRight_Thickness | Represents the horizontal splitter's right line thickness. |

## Filter Bar and Filter Cells Resource Keys

The following tables describe resource keys related to filter bar, filtered cells, and the filter bar row:

### FilterBar Background Resource Keys

The following filter bar resource keys are included:

| Resource Key | Description |
|---|---|
| C1DataGrid_FilterBar_Brush | Represents the FilterBar's background brush. |
| C1DataGrid_FilterBar_HoverBrush | Represents the FilterBar's background brush when the pointer is over the filter bar. |

| | |
|---|---|
| C1DataGrid_FilterBar_SelectedBrush | Represents the selected FilterBar's background brush. |
| C1DataGrid_FilterBar_SelectedHoverBrush | Represents the selected FilterBar's background brush when the pointer is over the filter bar. |

### FilterBar Cell Resource Keys

The following filter cell resource keys are included:

| Resource Key | Description |
|---|---|
| C1DataGrid_FilterCell_Brush | Represents the FilterBar cell's background brush. |
| C1DataGrid_FilterCell_HoverBrush | Represents the FilterBar cell's background brush when the pointer is over the filter cell. |
| C1DataGrid_FilterCell_SelectedBrush | Represents the selected FilterBar cell's background brush. |
| C1DataGrid_FilterCell_SelectedHoverBrush | Represents the selected FilterBar cell's background brush when the pointer is over the filter cell. |
| C1DataGrid_FilterCellBorder_Brush | Represents the FilterBar cell's border brush. |
| C1DataGrid_FilterCellBorder_HoverBrush | Represents the FilterBar cell's border brush when the pointer is over the filter cell. |
| C1DataGrid_FilterCellBorder_SelectedBrush | Represents the selected FilterBar cell's border brush. |
| C1DataGrid_FilterCellBorder_SelectedHoverBrush | Represents the selected FilterBar cell's border brush when the pointer is over the filter cell. |
| C1DataGrid_FilterCellBorder_Thickness | Represents the FilterBar cell's border thickness (all 4 edges). |
| C1DataGrid_FilterCellBorder_CornerRadius | Represents the FilterBar cell's Border corner radius (all 4 corners). |
| C1DataGrid_FilterCellText_Brush | Represents the FilterBar cell's text brush. |
| C1DataGrid_FilterCellText_HoverBrush | Represents the FilterBar cell's text brush when the pointer is over the filter cell. |
| C1DataGrid_FilterCellText_SelectedBrush | Represents the selected FilterBar cell's text brush. |
| C1DataGrid_FilterCellText_SelectedHoverBrush | Represents the selected FilterBar cell's text brush when the pointer is over the filter cell. |

### FilterBar Row Resource Keys

The following filter bar row resource keys are included:

| Resource Key | Description |
|---|---|
| C1DataGrid_FilterItemRowHeader_Brush | Represents the FilterBar row's header background brush. |
| C1DataGrid_FilterItemRowHeader_HoverBrush | Represents the FilterBar row's header background brush when the pointer is over the filter bar. |
| C1DataGrid_FilterItemRowHeader_SelectedBrush | Represents the selected FilterBar row's header background brush. |
| C1DataGrid_FilterItemRowHeader_SelectedHoverBrush | Represents the selected FilterBar row's header background brush on when the pointer is over the filter bar. |

| | |
|---|---|
| C1DataGrid_FilterItemRowHeader_ActiveBrush | Represents the FilterBar row's header background brush when activated by an item or column hover or selection. |
| C1DataGrid_FilterItemRowHeader_ActiveHoverBrush | Represents the FilterBar row's header background brush when activated by an item or column hover or selection when the pointer is over the filter bar. |
| C1DataGrid_FilterItemRowHeaderBorder_Brush | Represents the FilterBar row's header border brush. |
| C1DataGrid_FilterItemRowHeaderBorder_HoverBrush | Represents the FilterBar row's header border brush when the pointer is over the filter bar. |
| C1DataGrid_FilterItemRowHeaderBorder_SelectedBrush | Represents the selected FilterBar row's header border brush. |
| C1DataGrid_FilterItemRowHeaderBorder_SelectedHoverBrush | Represents the selected FilterBar row's header border brush when the pointer is over the filter bar. |
| C1DataGrid_FilterItemRowHeaderBorder_ActiveBrush | Represents the FilterBar row's header border brush when activated by an item or column hover or selection. |
| C1DataGrid_FilterItemRowHeaderBorder_ActiveHoverBrush | Represents the FilterBar row's header border brush when activated by an item or column hover or selection when the pointer is over the filter bar. |
| C1DataGrid_FilterItemRowHeaderBorder_Thickness | Represents the FilterBar row's header border thickness (all 4 edges). |
| C1DataGrid_FilterItemRowHeaderBorder_CornerRadius | Represents the FilterBar row's header border corner radius (all 4 corners). |
| C1DataGrid_FilterItemRowHeaderText_Brush | Represents the FilterBar row's header text brush. |
| C1DataGrid_FilterItemRowHeaderText_HoverBrush | Represents the FilterBar row's header text brush when the pointer is over the filter bar. |
| C1DataGrid_FilterItemRowHeaderText_SelectedBrush | Represents the selected FilterBar row's header text brush. |
| C1DataGrid_FilterItemRowHeaderText_SelectedHoverBrush | Represents the selected FilterBar row's header text brush when the pointer is over the filter bar. |
| C1DataGrid_FilterItemRowHeaderText_ActiveBrush | Represents the FilterBar row's header text brush when activated by an item or column hover or selection. |
| C1DataGrid_FilterItemRowHeaderText_ActiveHoverBrush | Represents the FilterBar row's header text brush when activated by an item or column hover or selection when the pointer is over the filter bar. |

## New Item Resource Keys

The following tables describe resource keys related to new row items and the add new row area:

### New Item Background Resource Keys

The following new item resource keys are included:

| Resource Key | Description |
|---|---|
| C1DataGrid_NewItem_Brush | Represents the new item's background brush. |
| C1DataGrid_NewItem_HoverBrush | Represents the new item's background brush when the pointer |

| | |
|---|---|
| | is over the new item header. |
| C1DataGrid_NewItem_SelectedBrush | Represents the selected new item's background brush. |
| C1DataGrid_NewItem_SelectedHoverBrush | Represents the selected new item's background brush when the pointer is over the new item header. |

### New Item Cell Resource Keys

The following new item cell resource keys are included:

| Resource Key | Description |
|---|---|
| C1DataGrid_NewItemCell_Brush | Represents the new item cell's background brush. |
| C1DataGrid_NewItemCell_HoverBrush | Represents the new item cell's background brush when the pointer is over the new item cell. |
| C1DataGrid_NewItemCell_SelectedBrush | Represents the selected new item cell's background brush. |
| C1DataGrid_NewItemCell_SelectedHoverBrush | Represents the selected new item cell's background brush when the pointer is over the new item cell. |
| C1DataGrid_NewItemCellBorder_Brush | Represents the new item cell's border brush. |
| C1DataGrid_NewItemCellBorder_HoverBrush | Represents the new item cell's border brush when the pointer is over the new item cell. |
| C1DataGrid_NewItemCellBorder_SelectedBrush | Represents the selected new item cell's border brush. |
| C1DataGrid_NewItemCellBorder_SelectedHoverBrush | Represents the selected new item cell's border brush when the pointer is over the new item cell. |
| C1DataGrid_NewItemCellBorder_Thickness | Represents the new item cell's border thickness (all 4 edges). |
| C1DataGrid_NewItemCellBorder_CornerRadius | Represents the new item cell's border corner radius (all 4 corners). |
| C1DataGrid_NewItemCellText_Brush | Represents the new item cell's text brush. |
| C1DataGrid_NewItemCellText_HoverBrush | Represents the new item cell's text brush when the pointer is over the new item cell. |
| C1DataGrid_NewItemCellText_SelectedBrush | Represents the selected new item cell's text brush. |
| C1DataGrid_NewItemCellText_SelectedHoverBrush | Represents the selected new item cell's text brush when the pointer is over the new item cell. |

### New Item Header Resource Keys

The following new item header resource keys are included:

| Resource Key | Description |
|---|---|
| C1DataGrid_NewItemRowHeader_Brush | Represents the new item row header's background brush. |
| C1DataGrid_NewItemRowHeader_HoverBrush | Represents the new item row header's background brush when the pointer is over the new item row header. |
| C1DataGrid_NewItemRowHeader_SelectedBrush | Represents the selected new item row header's background brush. |

| | |
|---|---|
| C1DataGrid_NewItemRowHeader_SelectedHoverBrush | Represents the selected new item row header's background brush when the pointer is over the new item header. |
| C1DataGrid_NewItemRowHeader_ActiveBrush | Represents the new item row header's background brush when activated by an item or column hover or selection. |
| C1DataGrid_NewItemRowHeader_ActiveHoverBrush | Represents the new item row header's background brush when activated by an item or column hover or selection when the pointer is over the new item row header. |
| C1DataGrid_NewItemRowHeaderBorder_Brush | Represents the new item row header's border brush. |
| C1DataGrid_NewItemRowHeaderBorder_HoverBrush | Represents the new item row header's border brush when the pointer is over the new item row header. |
| C1DataGrid_NewItemRowHeaderBorder_SelectedBrush | Represents the selected new item row header's border brush. |
| C1DataGrid_NewItemRowHeaderBorder_SelectedHoverBrush | Represents the selected new item row header's border brush when the pointer is over the new item row header. |
| C1DataGrid_NewItemRowHeaderBorder_ActiveBrush | Represents the new item row header's border brush when activated by an item or column hover or selection. |
| C1DataGrid_NewItemRowHeaderBorder_ActiveHoverBrush | Represents the new item row header's border brush when activated by an item or column hover or selection when the pointer is over the new item row header. |
| C1DataGrid_NewItemRowHeaderBorder_Thickness | Represents the new item row header's border thickness (all 4 edges). |
| C1DataGrid_NewItemRowHeaderBorder_CornerRadius | Represents the new item row header's border corner radius (all 4 corners). |
| C1DataGrid_NewItemRowHeaderText_Brush | Represents the new item row header's text brush. |
| C1DataGrid_NewItemRowHeaderText_HoverBrush | Represents the new item row header's text brush when the pointer is over the new item row header. |
| C1DataGrid_NewItemRowHeaderText_SelectedBrush | Represents the selected new item row header's text brush. |
| C1DataGrid_NewItemRowHeaderText_SelectedHoverBrush | Represents the selected new item row header's text brush when the pointer is over the new item row header. |
| C1DataGrid_NewItemRowHeaderText_ActiveBrush | Represents the new item row header's text brush when activated by an item or column hover or selection. |
| C1DataGrid_NewItemRowHeaderText_ActiveHoverBrush | Represents the new item row header's text brush when activated by an item or column hover or selection when the pointer is over the new item row header. |

## Indicator Resource Keys

The following tables describe resource keys related to various indicators in the grid:

### Indicator Resource Keys

The following indicator resource keys are included:

| Resource Key | Description |
|---|---|

| | |
|---|---|
| C1DataGrid_FilterItemIndicator_Brush | Represents the filter item indicator's brush. |
| C1DataGrid_NewItemIndicator_Brush | Represents the new item indicator's brush. |
| C1DataGrid_CurrentItemIndicator_Brush | Represents the current item indicator's brush. |
| C1DataGrid_SortAscendingIndicator_Brush | Represents the column header ascending sort indicator's brush. |
| C1DataGrid_SortDescendingIndicator_Brush | Represents the column header descending sort indicator's brush. |
| C1DataGrid_ColumnReorderIndicatorVertical_Brush | Represents the column vertical reorder indicators which point to the new column position. |
| C1DataGrid_ColumnReorderIndicatorHorizontal_Brush | Represents the column horizontal reorder indicators which point to the new column position. |

## Card and Carousel View Resource Keys

The following tables describe card and carousel view resource keys:

### Card View Resource Keys

The following card and carousel view resource keys are included:

| Resource Key | Description |
|---|---|
| C1DataGrid_Card_Brush | Represents the card view's background brush. |
| C1DataGrid_Card_HoverBrush | Represents the card view's background brush on when the pointer is over the card. |
| C1DataGrid_Card_SelectedBrush | Represents the selected card view's background brush. |
| C1DataGrid_Card_SelectedHoverBrush | Represents the selected card view's background brush when the pointer is over the selected card. |
| C1DataGrid_CardBorder_Brush | Represents the card view's border brush. |
| C1DataGrid_CardBorder_HoverBrush | Represents the card view's border brush when the pointer is over the card. |
| C1DataGrid_CardBorder_SelectedBrush | Represents the selected card view's border brush. |
| C1DataGrid_CardBorder_SelectedHoverBrush | Represents the selected card view's border brush when the pointer is over the selected card. |
| C1DataGrid_CardBorder_Thickness | Represents the card view's border thickness (all 4 edges). |
| C1DataGrid_CardBorder_CornerRadius | Represents the card view's border corner radius (all 4 corners). |

### Card View Cell Resource Keys

The following card and carousel view cell resource keys are included:

| Resource Key | Description |
|---|---|
| C1DataGrid_CardCell_Brush | Represents the card view cell's background brush. |
| C1DataGrid_CardCell_HoverBrush | Represents the card view cell's background brush when the pointer is over a card cell. |

| | |
|---|---|
| C1DataGrid_CardCell_SelectedBrush | Represents the selected card view cell's background brush. |
| C1DataGrid_CardCell_SelectedHoverBrush | Represents the selected card view cell's background brush when the pointer is over a selected card cell. |
| C1DataGrid_CardCellBorder_Brush | Represents the card view cell's border brush. |
| C1DataGrid_CardCellBorder_HoverBrush | Represents the card view cell's border brush when the pointer is over a card sell. |
| C1DataGrid_CardCellBorder_SelectedBrush | Represents the selected card view cell's border brush. |
| C1DataGrid_CardCellBorder_SelectedHoverBrush | Represents the selected card view cell's border brush when the pointer is over a selected card cell. |
| C1DataGrid_CardCellBorder_Thickness | Represents the card view cell's border thickness (all 4 edges). |
| C1DataGrid_CardCellBorder_CornerRadius | Represents the card view cell's border corner radius (all 4 corners). |
| C1DataGrid_CardCellText_Brush | Represents the card view cell's text brush. |
| C1DataGrid_CardCellText_HoverBrush | Represents the card view cell's text brush when the pointer is over the card cell. |
| C1DataGrid_CardCellText_SelectedBrush | Represents the selected card view cell's text brush. |
| C1DataGrid_CardCellText_SelectedHoverBrush | Represents the selected card view cell's text brush when the pointer is over the card cell. |

### Card View New Item Cell Resource Keys

The following card and carousel view new item cell resource keys are included:

| Resource Key | Description |
|---|---|
| C1DataGrid_CardNewItemCell_Brush | Represents the card view new item cell's background brush. |
| C1DataGrid_CardNewItemCell_HoverBrush | Represents the card view new item cell's background brush when the pointer is over the new item cell. |
| C1DataGrid_CardNewItemCell_SelectedBrush | Represents the selected new item card view cell's background brush. |
| C1DataGrid_CardNewItemCell_SelectedHoverBrush | Represents the selected card view new item cell's background brush when the pointer is over the new item cell. |
| C1DataGrid_CardNewItemCellBorder_Brush | Represents the card view new item cell's border brush. |
| C1DataGrid_CardNewItemCellBorder_HoverBrush | Represents the card view new item cell's border brush when the pointer is over the new item cell. |
| C1DataGrid_CardNewItemCellBorder_SelectedBrush | Represents the selected card view new item cell's border brush. |
| C1DataGrid_CardNewItemCellBorder_SelectedHoverBrush | Represents the selected card view new item cell's border brush when the pointer is over the new item cell. |
| C1DataGrid_CardNewItemCellBorder_Thickness | Represents the card view new item cell's border thickness (all 4 edges). |
| C1DataGrid_CardNewItemCellBorder_CornerRadius | Represents the card view new item cell's border corner radius (all 4 corners). |

| | |
|---|---|
| C1DataGrid_CardNewItemCellText_Brush | Represents the card view new item cell's text brush. |
| C1DataGrid_CardNewItemCellText_HoverBrush | Represents the card view new item cell's text brush when the pointer is over the new item cell. |
| C1DataGrid_CardNewItemCellText_SelectedBrush | Represents the selected card view new item cell's text brush. |
| C1DataGrid_CardNewItemCellText_SelectedHoverBrush | Represents the selected card view new item cell's text brush when the pointer is over the new item cell. |

### Card View Column Header Resource Keys

The following card and carousel view column header resource keys are included:

| Resource Key | Description |
|---|---|
| C1DataGrid_CardColumnHeader_Brush | Represents card view column header's background brush. |
| C1DataGrid_CardColumnHeader_HoverBrush | Represents the card view column header's background brush on when the pointer is over the column header. |
| C1DataGrid_CardColumnHeader_SelectedBrush | Represents the selected card view column header's background brush. |
| C1DataGrid_CardColumnHeader_SelectedHoverBrush | Represents the selected card view column header's background brush when the pointer is over the column header. |
| C1DataGrid_CardColumnHeaderBorder_Brush | Represents the card view column header's border brush. |
| C1DataGrid_CardColumnHeaderBorder_HoverBrush | Represents the card view column header's border brush when the pointer is over the column header. |
| C1DataGrid_CardColumnHeaderBorder_SelectedBrush | Represents the selected card view column header's border brush. |
| C1DataGrid_CardColumnHeaderBorder_SelectedHoverBrush | Represents the selected card view column header's border brush when the pointer is over the selected column header. |
| C1DataGrid_CardColumnHeaderBorder_Thickness | Represents the card view column header's border thickness (all 4 edges). |
| C1DataGrid_CardColumnHeaderBorder_CornerRadius | Represents the card view column header's border corner radius (all 4 corners). |
| C1DataGrid_CardColumnHeaderText_Brush | Represents the card view column header's text brush. |
| C1DataGrid_CardColumnHeaderText_HoverBrush | Represents the card view column header's text brush when the pointer is over the column header. |
| C1DataGrid_CardColumnHeaderText_SelectedBrush | Represents the selected card view column header's text brush. |
| C1DataGrid_CardColumnHeaderText_SelectedHoverBrush | Represents the selected card view column header's text brush when the pointer is over the column header. |

### Card View New Item Column Header Resource Keys

The following card and carousel view new item column header resource keys are included:

| Resource Key | Description |
|---|---|

| | |
|---|---|
| C1DataGrid_CardNewItemColumnHeader_Brush | Represents the card view new item column header's background brush. |
| C1DataGrid_CardNewItemColumnHeader_HoverBrush | Represents the new item column header's background brush when the pointer is over the new item column header. |
| C1DataGrid_CardNewItemColumnHeader_SelectedBrush | Represents the new item column header's background brush. |
| C1DataGrid_CardNewItemColumnHeader_SelectedHoverBrush | Represents the selected card view new item column header's background brush when the pointer is over the selected new item column header. |
| C1DataGrid_CardNewItemColumnHeaderBorder_Brush | Represents the card view new item column header's border brush. |
| C1DataGrid_CardNewItemColumnHeaderBorder_HoverBrush | Represents the card view new item column header's border brush when the pointer is over the new item column header. |
| C1DataGrid_CardNewItemColumnHeaderBorder_SelectedBrush | Represents the selected card view new item column header's border brush. |
| C1DataGrid_CardNewItemColumnHeaderBorder_SelectedHoverBrush | Represents the selected card view new item column header's border brush when the pointer is over the selected new item column header. |
| C1DataGrid_CardNewItemColumnHeaderBorder_Thickness | Represents the card view new item column header's border thickness (all 4 edges). |
| C1DataGrid_CardNewItemColumnHeaderBorder_CornerRadius | Represents the card view new item column header's border corner radius (all 4 corners). |
| C1DataGrid_CardNewItemColumnHeaderText_Brush | Represents the card view new item column header's text brush. |
| C1DataGrid_CardNewItemColumnHeaderText_HoverBrush | Represents the card view's new item column header's text brush when the mouse pointer is over the new item column header. |
| C1DataGrid_CardNewItemColumnHeaderText_SelectedBrush | Represents the selected card view new item column header's text brush. |
| C1DataGrid_CardNewItemColumnHeaderText_SelectedHoverBrush | Represents the selected card view new item column header's text brush when the pointer is over the new item header. |

### Card View Header Resource Keys

The following card and carousel view header resource keys are included:

| Resource Key | Description |
|---|---|
| C1DataGrid_CardHeader_Brush | Represents the card view header's background brush. |
| C1DataGrid_CardHeader_HoverBrush | Represents the card view header's background brush when the pointer is over the card header. |
| C1DataGrid_CardHeader_SelectedBrush | Represents the selected card view header's background brush. |
| C1DataGrid_CardHeader_SelectedHoverBrush | Represents the selected card view header's background brush when the pointer is over the card header. |
| C1DataGrid_CardHeaderBorder_Brush | Represents the card view header's border brush. |

| | |
|---|---|
| C1DataGrid_CardHeaderBorder_HoverBrush | Represents the card view header's border brush when the pointer is over the card header. |
| C1DataGrid_CardHeaderBorder_SelectedBrush | Represents the selected card view header's border brush. |
| C1DataGrid_CardHeaderBorder_SelectedHoverBrush | Represents the selected card view header's border brush when the pointer is over the card header. |
| C1DataGrid_CardHeaderBorder_Thickness | Represents the card view header's border thickness (all 4 edges). |
| C1DataGrid_CardHeaderBorder_CornerRadius | Represents the card view header's border corner radius (all 4 corners). |
| C1DataGrid_CardHeaderText_Brush | Represents the card view header's text brush. |
| C1DataGrid_CardHeaderText_HoverBrush | Represents the card view header's text brush when the pointer is over the card header. |
| C1DataGrid_CardHeaderText_SelectedBrush | Represents the selected card view header's text brush. |
| C1DataGrid_CardHeaderText_SelectedHoverBrush | Represents the selected card view header's text brush when the pointer is over the card header. |

# Data Validation

You can now validate data that users enter into the grid at run time. With cell value validation, you can validate user input to the **ItemCell.Value** property (either declaratively or in the event handler), reflect errors reported by a data source (via exception or **IDataErrorInfo** interface), and visualize errors with a customizable error indication. For additional details about cell validation, see the following topics.

## Inputting Data

When a user inputs data into the grid at run time, there are two distinct steps involved: inputting data in the grid cell's user interface and changing the source item's underlying property value.

In the first stage, the end user inputs data into a UI element representing a grid cell editor (for example a TextBox), which is bound to the Value property. That means in this stage the end user implicitly changes the value of the Value property. Note that the same action can also be performed explicitly with code assigning the Value property value.

In the second stage, the ItemCell.Value property value is propagated to the underlying source item's property. This happens when a grid cell leaves edit mode, either implicitly when the end user interacts with the grid (for example by selecting another cell or pressing the ENTER key), or explicitly by calling the **EndEdit** method of **ItemCell** or **ItemCellPresenter** object that represents the cell.

The following topics detail these two stages.

**Note:** You can set the grid so that changes are committed to the underlying data source before the user leaves the cell edit mode. For details, see Committing Data to a Data Source (page 58).

## Changing the Value Property

The first stage of a source value update – changing the ItemCell.Value property – includes four main steps: converting the value to a data type, raising the CellValueChanging event, returning errors, and executing the ItemCellValidators. These steps are detailed below:

**Step 1: Converting the value to a data type**

In the first step, the value provided to the ItemCell.Value property is converted to the data type defined in the DataType property of the corresponding column.

**Step 2: Raising the CellValueChanging event**

In the second step, the CellValueChanging event is raised and passes the **CellValueChangingEventArgs** object to an event handler. The event handler includes the following event argument values:

- **IsError**: **True** if the conversion performed in step 1 has failed; otherwise, **False**.
- **ConversionException**: An exception raised during a conversion; otherwise, a null value.
- **NonConvertedValue**: The original (pre-conversion) value of the **ItemCell.Value** property.
- **ProposedValue**: A converted value if conversion in step 1 has succeeded; otherwise, the previous value of the **ItemCell.Value** property.

In the event handler you can:

- Check a proposed value (**ProposedValue** or **NonConvertedValue** arguments) and report an error by setting the **IsError** argument to **True**. You can also set the ErrorContent argument to a description of the error.

  Alternatively, you can suppress a conversion error by setting the IsError argument to **False** (note that IsError is set to **True** on event entry in this case).
- Provide a corrected value (different from an input value) for the **ItemCell.Value** property by setting another value to the ProposedValue argument.

**Step 3: Returning Errors**

If the IsError argument is **True** on the CellValueChanging event exit, the process is stopped and the CellErrorInfo object being returned by the ErrorInfo property gets the following property values from event arguments:

| ErrorInfo property | Value or value source |
|---|---|
| HasError | CellValueChangingEventArgs.IsError |
| ErrorContent | CellValueChangingEventArgs.ErrorContent |
| ErrorException | CellValueChangingEventArgs.ConversionException |
| MustFixError | True |
| IsDataSourceError | False |

**Step 4: Executing the ItemCellValidator**

If the IsError argument is **True** on the CellValueChanging event exit, the CellValidator-derived objects from the ItemCellValidators collection are executed. CellValidator is an abstract class with the **Validate** abstract method that should be implemented in a custom CellValidator-derived class and return a CellValidationResult object that indicates whether a value passed a validation. If at least one of the validations fails, that is CellValidationResult returned by the **Validate** method has the **IsValid** property set to **False**, the process is stopped and the CellErrorInfo object returned by the ErrorInfo property gets the following property values from CellValidationResult:

| ErrorInfo property | Value or value source |
|---|---|
| HasError | not CellValidationResult.IsValid |
| ErrorContent | CellValidationResult.ErrorContent |
| ErrorException | Null |
| MustFixError | True |
| IsDataSourceError | False |

A validator that checks an input value for positivity may look similar to the following:

- Visual Basic

```
Public Class PositiveValueValidator
    Inherits CellValidator
    Public Overloads Overrides Function Validate(ByVal value As Object,
ByVal cell As ValueCellBase, ByVal cultureInfo As CultureInfo) As
CellValidationResult
        If TypeOf value Is Integer Then
            If CInt(value) <= 0 Then
                Return New CellValidationResult(False, "Value must be
greater than 0.")
            End If
        Else
            Return New CellValidationResult(False, "Invalid value type")
        End If
        Return CellValidationResult.ValidResult
    End Function
End Class
```

- C#

```
public class PositiveValueValidator : CellValidator
{
    public override CellValidationResult Validate(object value,
ValueCellBase cell, CultureInfo cultureInfo)
    {
        if (value is int)
        {
            if ((int)value <= 0)
                return new CellValidationResult(false, "Value must be
greater than 0.");
        }
        else
            return new CellValidationResult(false, "Invalid value type");
        return CellValidationResult.ValidResult;
    }
}
```

There is a special validator class named CellValidationRuleWrapper that allows using a validator derived from the **System.Windows.Controls.ValidationRule** used in WPF Binding.

## Changing the Source Value

The second stage of the source value update – setting the source value to the value of the ItemCell.Value property – can only be started if there are no reported errors in the first stage (that is, ErrorInfo.HasError is **False**). The process can be started explicitly by calling the EndEdit method (or the EndEdit method, which

calls EndEdit) or implicitly by the action an end user takes such as selecting another cell or pressing the ENTER key.

The stage includes two steps: assigning ItemCell.Value to a source property and leaving edit mode.

**Step 1: Assigning ItemCell.Value to a source property**

The value of the ItemCell.Value property is assigned to a source property. If the source property raised an exception, the process is stopped and an error is reported in the ErrorInfo property, whose sub-properties are set as following:

| ErrorInfo property | Value or value source |
|---|---|
| HasError | True |
| ErrorContent | Exception.Message |
| ErrorException | Exception raised by the source property. |
| MustFixError | True |
| IsDataSourceError | True |

**Step 2: Leaving edit mode**

If step 1 has succeeded, the cell leaves edit mode, that is, the IsEdit property value becomes **False**. If the source item supports the **IDataErrorInfo** interface, the grid attempts to retrieve errors for the source property via this interface. If an error has been returned for the property, the error is reflected in the ErrorInfo property, with sub-properties set as following:

| ErrorInfo property | Value or value source |
|---|---|
| HasError | True |
| ErrorContent | IDataErrorInfo.Item[property_name] |
| ErrorException | Null |
| MustFixError | False |
| IsDataSourceError | True |

Note that you can set the grid so that changes are committed to the underlying data source before the user leaves the cell edit mode. For details, see Committing Data to a Data Source (page 58).

# Validating Cell Input

Cell input validation allows you to customize how users interact with the grid and what values the grid will accept.

With the built-in cell value validation mechanism, the grid can validate user input to the ItemCell.Value property (either declaratively or in an event handler), reflect errors reported by the data source (via exception or **IDataErrorInfo** interface), and visualize indicated errors in a customizable way.

The cell value validation mechanism covers both stages of the source property value update process outlined in the Inputting Data topic. If an error occurs during the update process, the ErrorInfo property of the CellErrorInfo type will have **HasError** property set to **True** and other properties will contain detailed information describing the error. A cell with an invalid input will also have a visual indication of the error – by default represented as a red rectangle around a cell and a ToolTip with the error's description.

## Cell Error Visualization

In cases where cell value editing performed via grid cell (represented by ItemCellPresenter object) caused an error (that is the HasError property is **True**), the grid creates an adorner object above the ItemCellPresenter. A UI of the adorner is defined in the ItemCellErrorAdornerTemplate property (default for all cells) or in the ItemCellErrorAdornerTemplate property, the latter allows you to provide a different template for a specific column. By default it shows a red rectangle around a cell and a ToolTip with a content retrieved from the ItemCellPresenter.ErrorInfo.ErrorContent property. In order to turn off error visualization you should assign the ItemCellErrorAdornerTemplate property with an empty template or a null value.

You can customize the error visualization UI by providing a custom template for the ItemCellErrorAdornerTemplate property of C1DataGrid or Column. The template should use the C1AdornedElementPlaceholder element, which is positioned and sized so as to exactly coincide with an adorned ItemCellPresenter, which is used as a placeholder to relatively position another element in the template. C1AdornedElementPlaceholder is an analogue of the **System.Windows.Controls.AdornedElementPlaceholder** element used in the **System.Windows.Controls.Validation.ErrorTemplate** template for WPF Binding error visualization.

The default implementation of the ItemCellErrorAdornerTemplate template looks as follows:

```
<ControlTemplate x:Key="ItemCellErrorAdornerTemplate"
TargetType="Control">
    <local:C1AdornedElementPlaceholder Name="placeHolder" >
        <Border BorderThickness="1" BorderBrush="Red"
CornerRadius="{DynamicResource C1DataGrid_CellBorder_CornerRadius}">
            <local:PropertyBridge Source="{Binding ErrorInfo.ErrorContent,
Mode=OneWay}" Target="{Binding AdornedElement.ToolTip,
ElementName=placeHolder, Mode=OneWayToSource}"/>
        </Border>
    </local:C1AdornedElementPlaceholder>
</ControlTemplate>
```

Note that this template assigns the **ToolTip** property of the adorned **ItemCellPresenter** presenter (which is done by a **PropertyBridge** bound to the AdornedElement.**ToolTip** property), instead of using of the **ToolTip** property of some element of the template itself. The reason is that the error adorner is not a hit testable element used in order to keep the ability for a user to interact with UI of the **ItemCellPresenter** decorated by the error adorner. As a result, a ToolTip will not be shown for elements of the error adorner template.

Alternatively, you can customize cell error visualization by providing an updated template for the ItemCellTemplate property. The template may include a trigger for the **ItemCellPresenter.HasError** property that changes the appearance of the cell, for example the cell's background color, font color, and so on. The **ItemCellPresenter.HasError** dependency property is a shortcut for the **ItemCellPresenter.ItemCell.ErrorInfo.HasError** property path and is introduced solely for the purpose of allowing you to use a relatively lightweight Trigger statement instead of slower a **DataTrigger** in the **ItemCellTemplate** template.

An example of an updated default **ItemCellTemplate** template that uses a Trigger based on the **HasError** property to provide error visualization may look like the following:

```
<ControlTemplate x:Key="Office2007ItemCellTemplate"
TargetType="local:ItemCellPresenter">
      <DockPanel SnapsToDevicePixels="True">
            <Rectangle DockPanel.Dock="Right"
            Width="{DynamicResource
C1DataGrid_GridLineVertical_Thickness}"
            Fill="{DynamicResource C1DataGrid_GridLineVertical_Brush}"
            VerticalAlignment="Stretch" HorizontalAlignment="Right"
            Margin="0,-1,0,-1"/>
      <Border Name="brd"
            BorderThickness="{DynamicResource
C1DataGrid_CellBorder_Thickness}"
```

```
            BorderBrush="{DynamicResource C1DataGrid_CellBorder_Brush}"
            Background="{DynamicResource C1DataGrid_Cell_Brush}"
            CornerRadius="{DynamicResource
C1DataGrid_CellBorder_CornerRadius}"
            TextElement.Foreground="{DynamicResource
C1DataGrid_CellText_Brush}">
            <local:ItemCellContentPresenter/>
      </Border>
</DockPanel>
<ControlTemplate.Triggers>
<!-- ................................................ -->
      <!-- Definitions of original triggers that goes here are omitted for
clarity; look in the Common.xaml file for a template with the
"Office2007ItemCellTemplate" key to see a full definition -->

            <Trigger Property="HasError" Value="true">
                <Setter TargetName="brd" Property="Background"
Value="Red"/>
                <Setter TargetName="brd" Property="TextElement.Foreground"
                  Value="White"/>
                <Setter TargetName="brd" Property="ToolTip"
Value="{Binding ErrorInfo.ErrorContent}"/>
            </Trigger>
        </ControlTemplate.Triggers>
      </ControlTemplate>
```

## Cell In Error and Focus

For the cell in error, the MustFixError property value indicates whether the focus can be removed from the cell. When **True**, focus can't be shifted from the cell until the error is corrected or editing is cancelled. Currently this property only returns **False**, allowing shifting focus from the cell in error, when the error was reported via the **IDataErrorInfo** interface of a source data item.

## Improving IDataErrorInfo Performance Using the IQuickDataErrorInfo Interface

If data source items implement the **IDataErrorInfo** interface, C1DataGrid needs to check if errors are present for each grid column via the interface. This may bring performance penalties that can vary depending on implementation details. The **IQuickDataErrorInfo** interface can help improve performance.

The **IQuickDataErrorInfo** interface extends the **IDataErrorInfo** interface (which it's derived from) by adding a single Boolean property named **HasErrors**. **HasErrors**, as you might expect, indicates whether at least one error is present on the item. If **HasErrors** returns **False**, the grid doesn't need to retrieve errors for each column's underlying property, making processing far faster.

To improve the C1DataGrid control's grid performance, you might consider implementing the **IQuickDataErrorInfo** interface instead of (or in addition to) the **IDataErrorInfo** interface in your custom item class.

# Grid for WPF Samples

Please be advised that this ComponentOne software tool is accompanied by various sample projects and/or demos, which may make use of other ComponentOne development tools included with the ComponentOne Studios.

**Note:** ComponentOne samples are also available at http://helpcentral.componentone.com.

### C# Samples

The following C# samples are included:

| Sample | Description |
|---|---|
| C1WPFGridSamples | For Visual Studio 2008. Demonstrates the functionality supported by **ComponentOne Grid for WPF**, including data binding, changing data views (including the card and carousel views), customizing the appearance of the grid (including through themes), and manipulating the grid at run time. This sample uses the C1DataGrid control. |
| C1WPFGridSamplesVS2005 | For Visual Studio 2005. Demonstrates the functionality supported by **ComponentOne Grid for WPF**, including data binding, changing data views (including the card and carousel views), customizing the appearance of the grid (including through themes), and manipulating the grid at run time. This sample uses the C1DataGrid control. |

### Running the Sample

To run the sample, complete the following steps:

1. Open Microsoft Expression Blend or Visual Studio.
2. Select **File | Open | Project/Solution**.
3. Click the drop-down **Look in** list and find the **ComponentOne Samples** folder installed in your **MyDocuments** folder (**Documents** in Vista). This is the default samples location created by the installation program. The location may be different if you installed **Grid for WPF** elsewhere on your machine.
4. Open a project folder, select a solution, and click **Open** to open the project in Blend or Visual Studio.
5. Select **Project | Test Solution** or click **F5** to run the sample.

# Grid for WPF Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio .NET and know how to use the C1DataGrid control in general. If you are unfamiliar with the **ComponentOne Grid for WPF** product, please see the Grid for WPF Quick Start (page 25) first.

Each topic in this section provides a solution for specific tasks using the **ComponentOne Grid for WPF** product.

Each task-based help topic also assumes that you have created a new WPF project. For additional information on this topic, see Creating a .NET Project in Visual Studio (page 1) or Creating a Microsoft Blend Project (page 1).

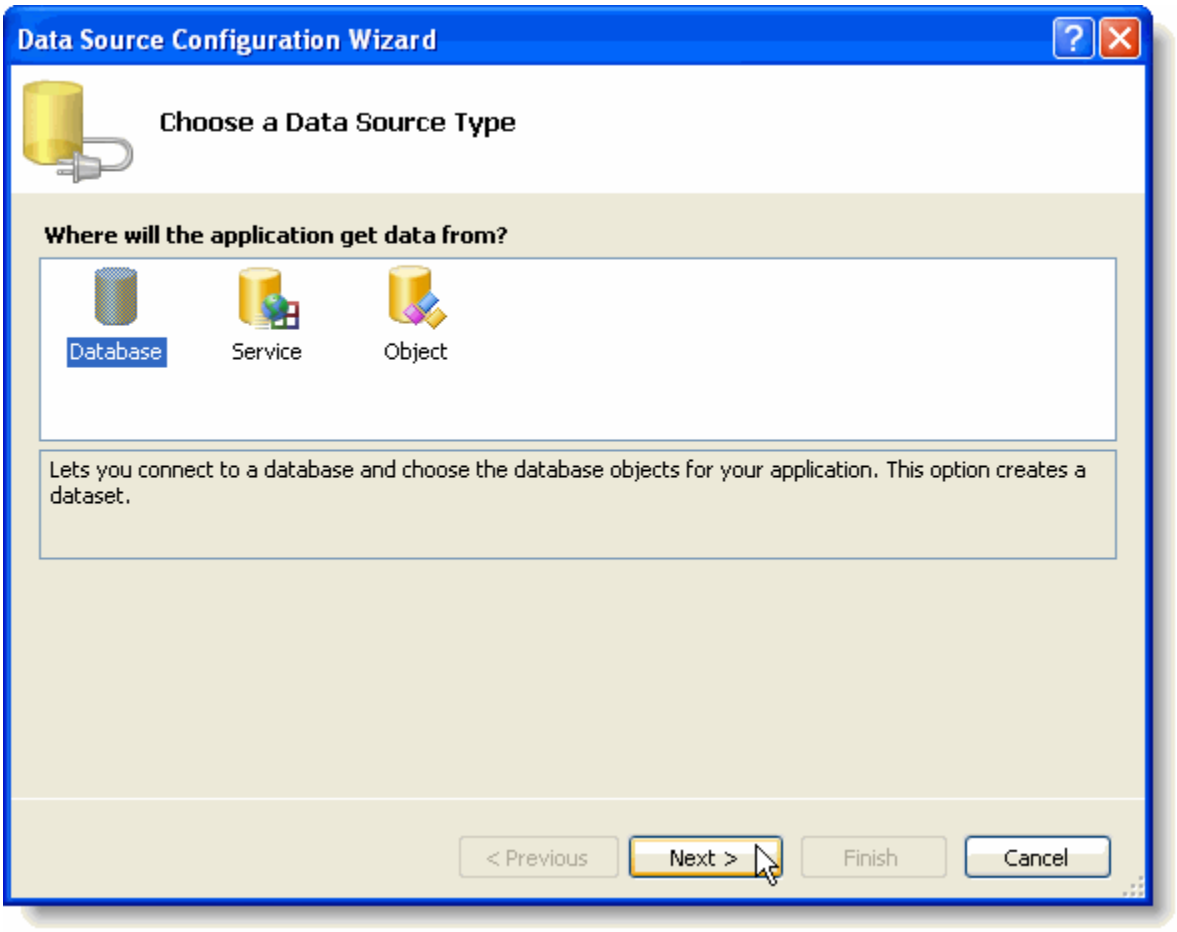
**Note:** Some of the examples reference the **C1NWind.mdb** database which is installed by default in the **ComponentOne Samples\Common** folder installed in your **MyDocuments** folder (**Documents** in Vista).

# Binding Grid for WPF to a Database

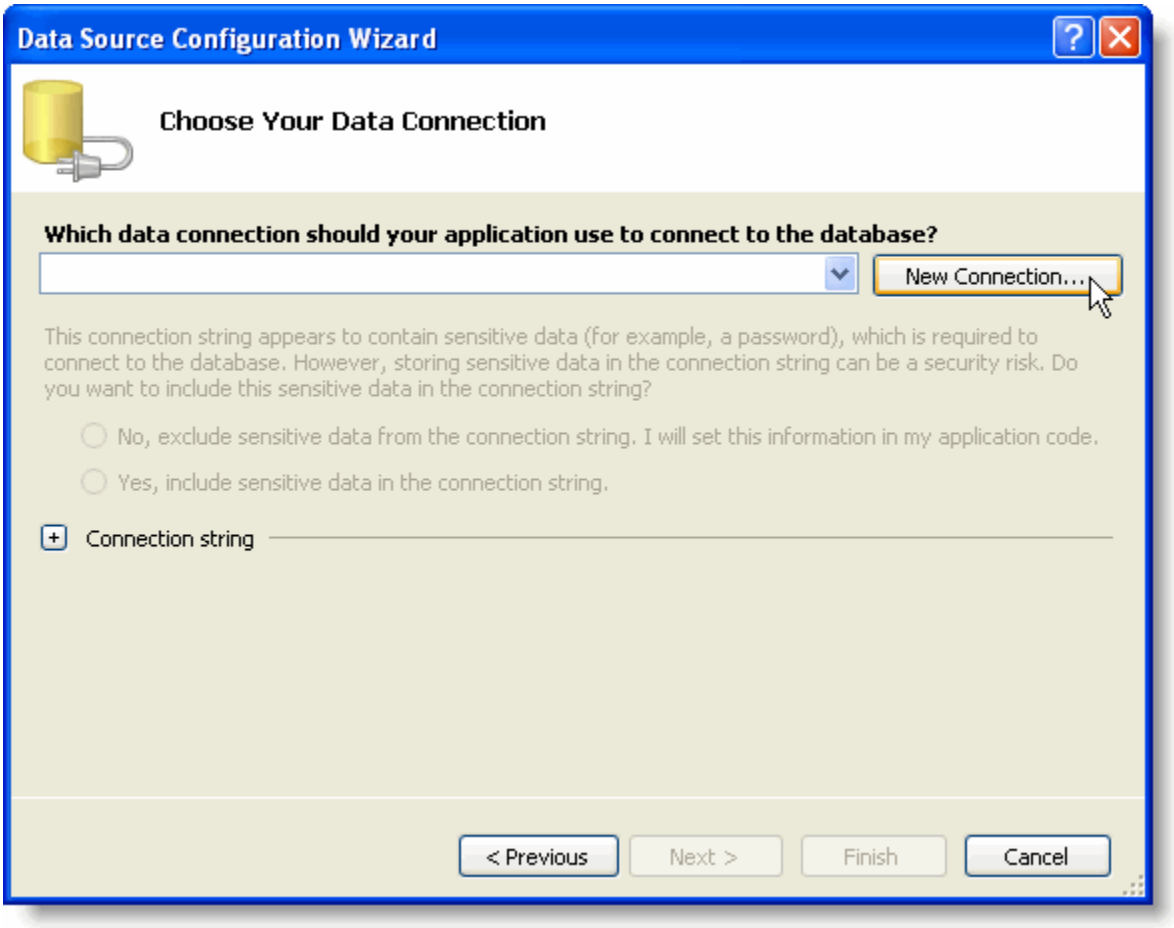
In this topic you'll add a data source to a project created in Visual Studio 2008. You'll then open the project in Expression Blend to complete binding the grid to the data source. Begin by creating a new WPF project in Visual Studio and adding the C1DataGrid component to your form.

**To add a data source and set up data binding in Visual Studio, complete the following steps:**

1. From the **Data** menu, select **Add New Data Source**. The **Data Source Configuration Wizard** appears.
2. Confirm that **Database** is selected and click **Next**.

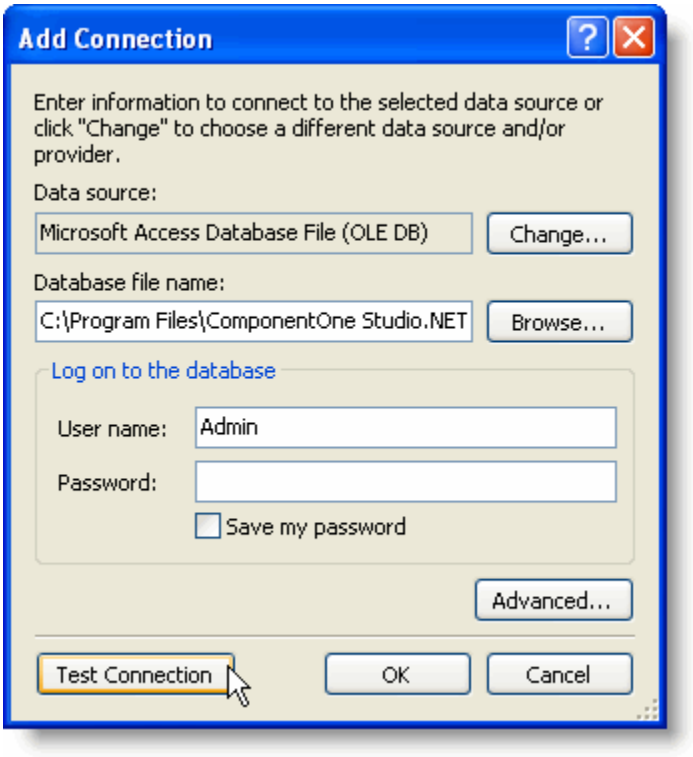



3. Click the **New Connection** button to locate and connect to a database.

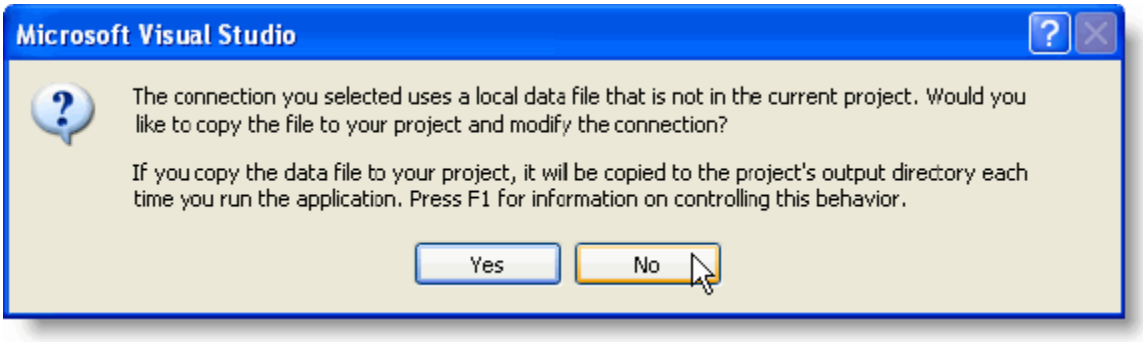



If the **Choose Data Source** dialog box appears, select **Microsoft Access Database File** and click **Continue**. The **Add Connection** dialog box will appear.

4. In the **Add Connection** dialog box, click the **Browse** button and locate **C1NWind.mdb** in the samples installation directory. Select it and click **Open**.
5. Click the **Test Connection** button to make sure that you have successfully connected to the database or server and click **OK**.

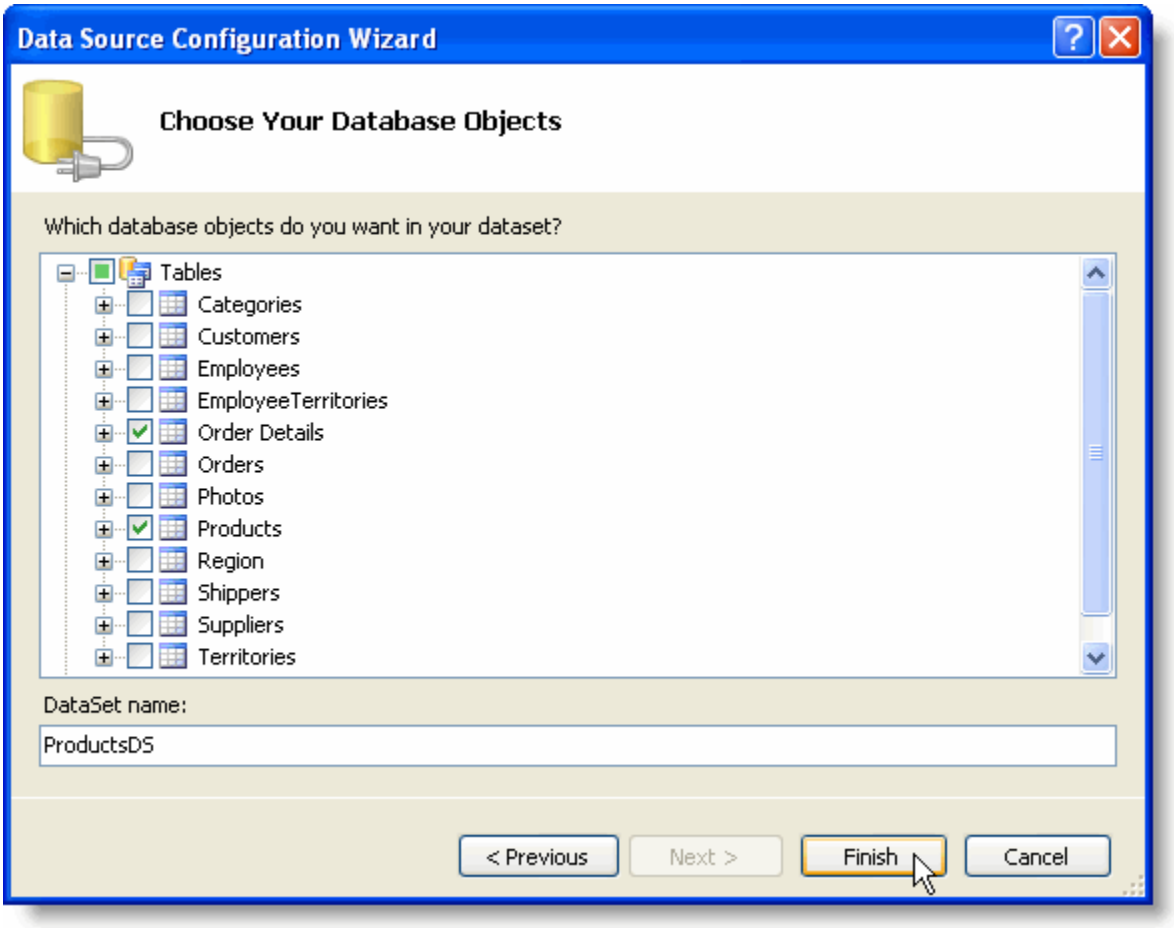



6. Click **OK** to close the **Add Connection** dialog box. The new string appears in the data connection drop-down list on the **Choose Your Data Connection** page.
7. Click the **Next** button to continue. A dialog box will appear asking if you would like to add the data file to your project and modify the connection string. Since it is not necessary to copy the database to your project, click **No**.

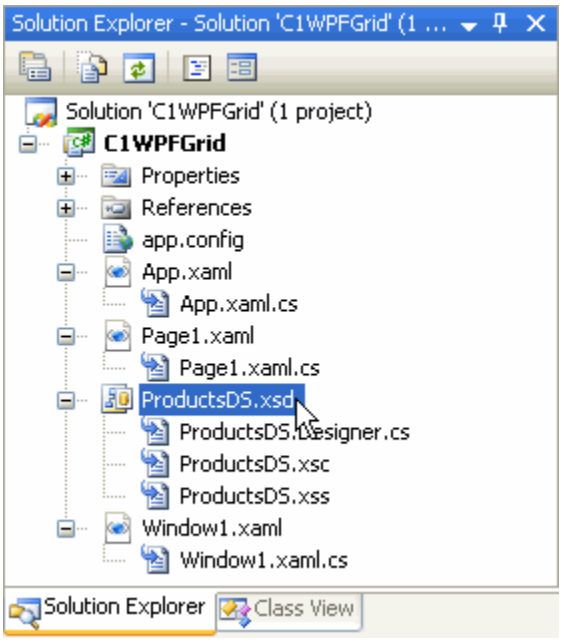



8. In the next window, the **Yes, save the connection as** checkbox is checked by default and a name has been automatically entered in the text box. Click **Next** to continue.


9. In the **Choose Your Database Objects** window, you can select the tables and fields that you would like in your dataset. Select the **Products** and **Order Details** tables and change the DataSet name to **ProductsDS**.




10. Click **Finish** to exit the wizard. The **ProductsDS.xsd** files now appear in the Solution Explorer.



11. In the Solution Explorer, double-click the Window1.xaml.cs (or Window1.xaml.vb) file to switch to code view.
12. Add the following references to the top of the Window1.xaml.cs (or Window1.xaml.vb) file, replacing *ProjectName* with the name of your project:

    - Visual Basic
    ```
    Imports C1.WPF.C1DataGrid
    Imports ProjectName.ProductsDSTableAdapters
    ```
    - C#
    ```
    using C1.WPF.C1DataGrid;
    using ProjectName.ProductsDSTableAdapters;
    ```
13. Add the following code to the Window1 class to retrieve the products and order details data from the database:

    - Visual Basic
    ```
    Public Partial Class Window1
        Inherits Window
        Private _productsDataSet As ProductsDS = Nothing
        Public ReadOnly Property ProductsDataSet() As ProductsDS
            Get
                If _productsDataSet Is Nothing Then
                    _productsDataSet = New ProductsDS()
                    Dim prodTA As New ProductsTableAdapter()
                    prodTA.Fill(_productsDataSet.Products)
                    Dim ordDetTA As New Order_DetailsTableAdapter()
                    ordDetTA.Fill(_productsDataSet.Order_Details)
                End If
                Return _productsDataSet
            End Get
        End Property

        Public Sub New()
            InitializeComponent()
        End Sub
    End Class
    ```
    - C#
    ```
    public partial class Window1 : Window
    {
        private ProductsDS _productsDataSet = null;
        public ProductsDS ProductsDataSet
        {
            get
            {
                if (_productsDataSet == null)
                {
                    _productsDataSet = new ProductsDS();
                    ProductsTableAdapter prodTA = new
    ProductsTableAdapter();
                    prodTA.Fill(_productsDataSet.Products);
                    Order_DetailsTableAdapter ordDetTA = new
    Order_DetailsTableAdapter();
                    ordDetTA.Fill(_productsDataSet.Order_Details);
                }
                return _productsDataSet;
            }
    ```

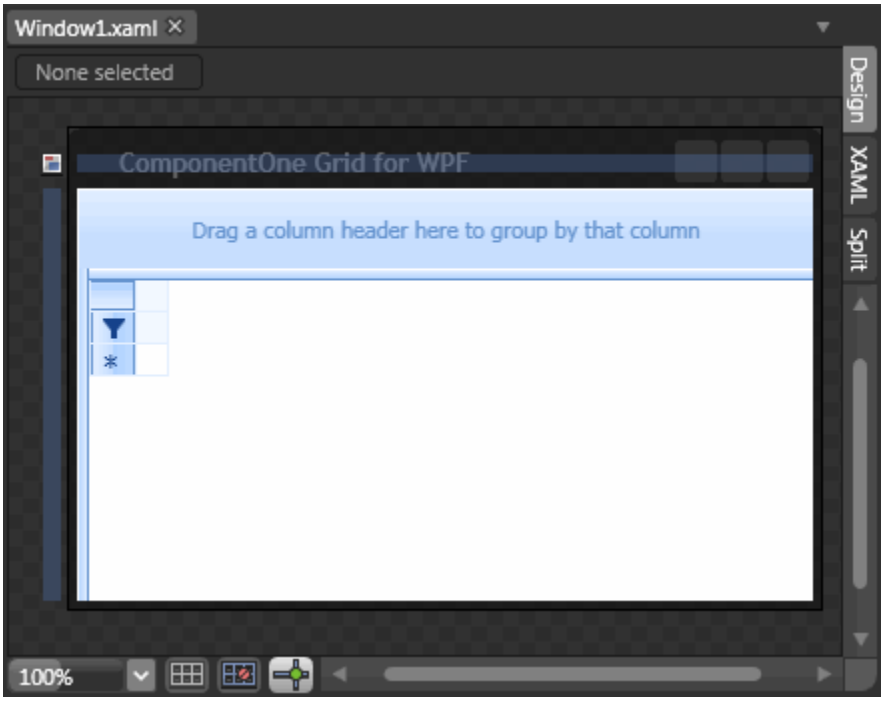
```
    }

    public Window1()
    {
        InitializeComponent();
    }
}
```

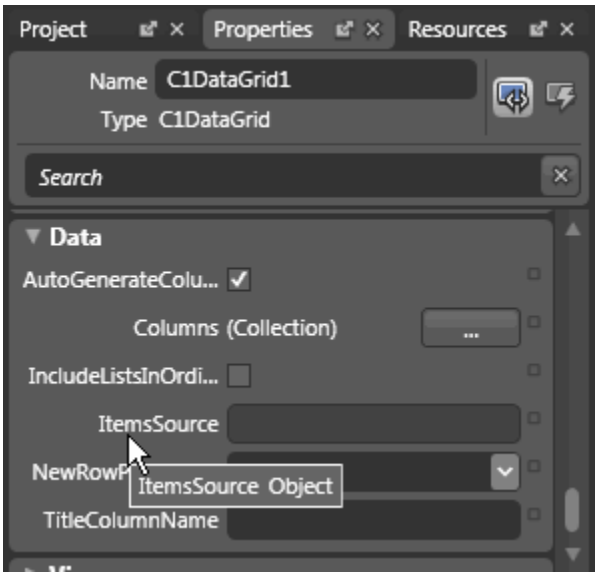
14. Run your project to ensure that everything is working correctly and then close your running application and save and close your project. You'll complete binding the grid to a data source in Expression Blend.

**To bind the grid to a data source in Expression Blend, complete the following steps:**

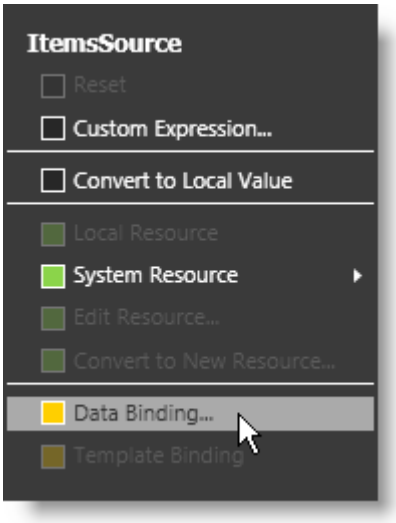
1. Open Expression Blend and select **Open Project/Solution** from the **File** menu.
2. In the **Open Project** dialog box, locate your project file and click **Open**. The project you created in Visual Studio will open in Blend.




3. Click once on the **C1DataGrid** control so that it is selected, navigate to the c1DataGrid1 Properties window, and locate the ItemsSource property.



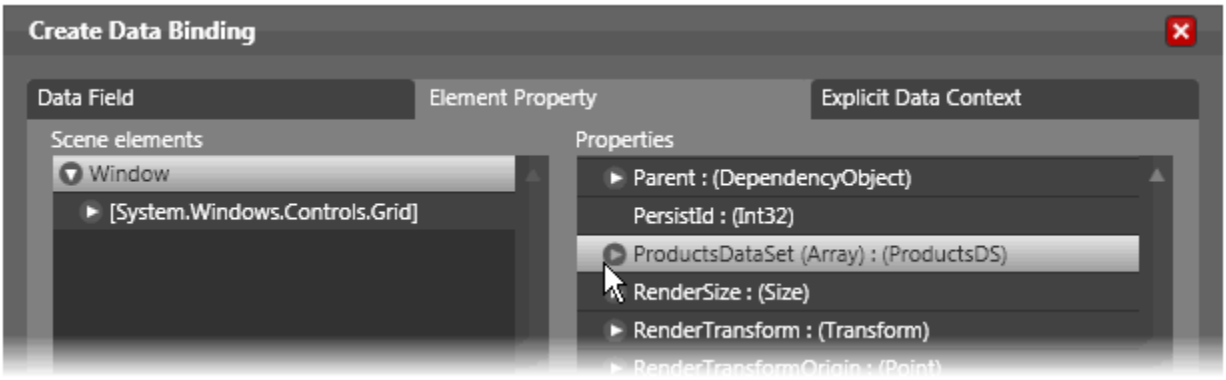

4. Click the square next to the ItemsSource property to access advanced properties and select **Data Binding** from the menu that appears.




5. In the **Create Data Binding** dialog box, select the **Element Property** tab.

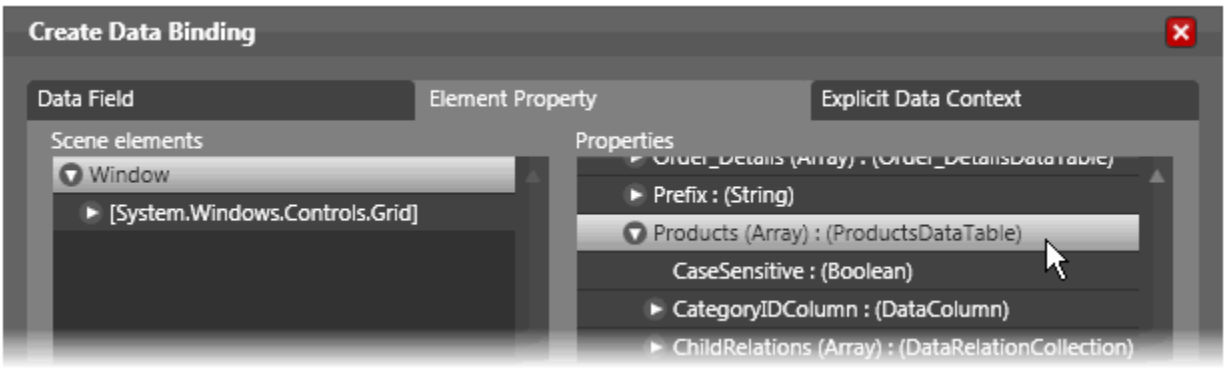

The **Element Property** tab appears with Scene elements in the left pane and Properties in the right pane.

6. In the right **Properties** pane of the **Element Property** tab, scroll down and click the arrow icon next to the **ProductsDataSet (Array) : (ProductsDS)** item to expand the available tables.




Note that this is the dataset that you added in Visual Studio.

7. Select **Products(Array) : (ProductsDataTable)** to bind the grid to the products table.



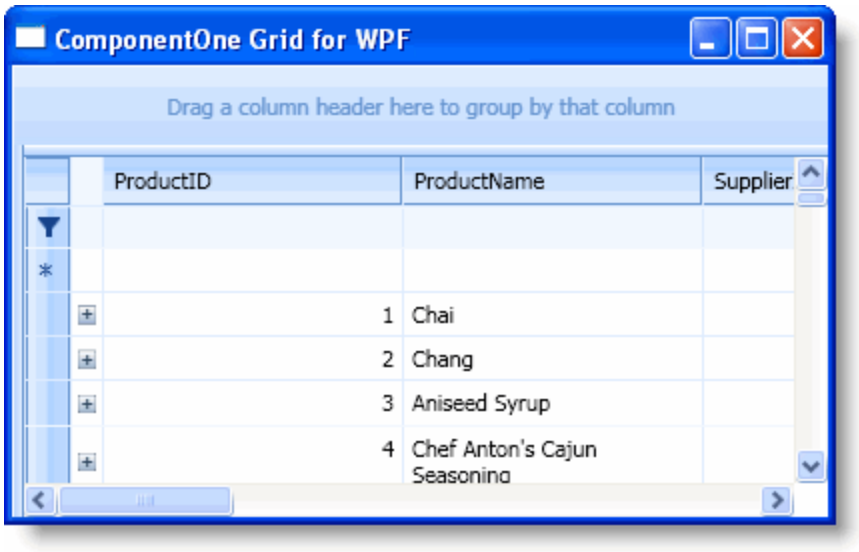

8. Click Finish to complete the data binding process and close the **Create Data Binding** dialog box.

   Notice in the XAML view, the `<c1grid:C1DataGrid>` now appears as the following:

```
<c1grid:C1DataGrid Name="c1DataGrid1" ItemsSource="{Binding
Path=ProductsDataSet.Products, ElementName=Window, Mode=Default}" />
```

**Run the program and observe:**

The grid is now populated with data from the *Products* table.




# Controlling Grid Interaction

The following task-based help topics detail how you can limit your users' interaction with **Grid for WPF**. For example, you can prevent users from filtering, sorting, reordering, deleting, splitting, and editing the grid through code and XAML.

## Preventing a Column from Being Filtered

You can prevent users from filtering particular columns at run time by using the AllowFilter property. By default the AllowFilter property is set to **True** and users can filter columns. By setting the AllowFilter property to **False**, you can prevent users from filtering particular columns in a grid.

### In XAML

Set the AllowFilter property to **False** and prevent users from filtering the *ProductName* column at run time by adding `AllowFilter="False"` to the `<c1grid:Column>` tag so that it looks similar to the following:

```
<c1grid:Column PropertyName="ProductName" Caption="Name"
AllowFilter="False"/>
```

### In Code

Set the AllowFilter property to **False** and prevent users from filtering the *ProductName* column at run time by adding the following code to your project:

- Visual Basic

```
C1DataGrid1.Columns("ProductName").AllowFilter = False
```

- C#

```
c1DataGrid1.Columns["ProductName"].AllowFilter = false;
```

## Preventing the Grid from Being Sorted

You can easily prevent the grid from being sorted by using the AllowSort property. By default the AllowSort property is set to **True** and sorting is allowed. By setting the AllowSort property **False**, you can prevent users from sorting the grid.

### In XAML

Prevent users from sorting the grid at run time by adding `AllowSort="False"` to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" AllowSort="False">
```

**In Code**

Prevent users from sorting the grid at run time by adding the following code to your project:

- Visual Basic
```
C1DataGrid1.AllowSort = False
```
- C#
```
c1DataGrid1.AllowSort = false;
```

**At Design Time**

Prevent users from sorting the grid at run time by setting the AllowSort property **False** in the C1DataGrid's Properties window.

## Preventing a Column from Being Sorted

You can easily prevent a column in the grid from being sorted by using the AllowSort property. By default the AllowSort property is set to **True** and sorting is allowed. By setting the AllowSort property **False**, you can prevent users from sorting a column in the grid.

**In XAML**

Prevent users from sorting the *ProductName* column at run time by adding `AllowSort="False"` to the `<c1grid:Column>` tag so that it looks similar to the following:

**<c1grid:Column PropertyName="ProductName" Caption="Name" `AllowSort="False"`/>In Code**

Prevent users from sorting the *ProductName* column at run time by adding the following code to your project:

- Visual Basic
```
C1DataGrid1.Columns("ProductName").AllowSort = False
```
- C#
```
c1DataGrid1.Columns["ProductName"].AllowSort = false;
```

## Preventing the Grid from Being Reordered

**ComponentOne Grid for WPF** allows users to easily reorder columns through a drag-and-drop operation. You can easily prevent all columns in a grid from being reordered by using the AllowColumnMove property. By default the AllowColumnMove property is set to **True** and reordering columns is allowed. By setting the AllowColumnMove property **False**, you can prevent users from reordering columns in the grid.

**In XAML**

Prevent users from reordering the grid at run time by adding `AllowColumnMove="False"` to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:
```
<c1grid:C1DataGrid Name="C1DataGrid1" AllowColumnMove="False"/>
```

**In Code**

Prevent users from reordering grid at run time by adding the following code to your project:

- Visual Basic
```
C1DataGrid1.AllowColumnMove = False
```
- C#
```
c1DataGrid1.AllowColumnMove = false;
```

**At Design Time**

Prevent users from reordering the grid at run time by setting the AllowColumnMove property **False** in the C1DataGrid's Properties window.

## Preventing a Column from Being Reordered

ComponentOne Grid for WPF allows users to easily reorder columns through a drag-and-drop operation. You can easily prevent specific columns in the grid from being reordered by using the AllowColumnMove property. By default the AllowColumnMove property is set to **True** and reordering columns is allowed. By setting the AllowColumnMove property **False**, you can prevent users from reordering a column in the grid.

### In XAML

Prevent users from reordering the *ProductName* column at run time by adding `AllowColumnMove="False"` to the `<c1grid:Column>` tag so that it looks similar to the following:

```
<c1grid:Column PropertyName="ProductName" Caption="Name"
AllowColumnMove="False"/>
```

### In Code

Prevent users from reordering the *ProductName* column at run time by adding the following code to your project:

- Visual Basic

```
C1DataGrid1.Columns("ProductName").AllowColumnMove = False
```

- C#

```
c1DataGrid1.Columns["ProductName"].AllowColumnMove = false;
```

## Preventing a Column from Being Grouped

You can prevent users from grouping particular columns at run time by using the AllowGroupBy property. By default the AllowGroupBy property is set to **True** and users can group columns. By setting the AllowGroupBy property to **False** you can prevent users from grouping particular columns in a grid.

### In XAML

Prevent users from grouping the *ProductName* column at run time by adding `AllowGroupBy="False"` to the `<c1grid:Column>` tag so that it looks similar to the following:

```
<c1grid:Column PropertyName="ProductName" Caption="Name"
AllowGroupBy="False"/>
```

### In Code

Prevent users from grouping the *ProductName* column at run time by adding the following code to your project:

- Visual Basic

```
C1DataGrid1.Columns("ProductName").AllowGroupBy = False
```

- C#

```
c1DataGrid1.Columns["ProductName"].AllowGroupBy = false;
```

## Preventing Rows from Being Deleted

You can prevent users from deleting rows at run time by using the AllowDelete property. By default the AllowDelete property is set to **True** and end-users can delete rows from the grid with the CTRL-D key combination. By setting the AllowDelete property to **False** you can prevent users from deleting rows in the grid.

### In XAML

Prevent users from deleting grid rows at run time by adding `AllowDelete="False"` to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" AllowDelete="False"/>
```

### In Code

Prevent users from deleting grid rows at run time by adding the following code to your project:

- Visual Basic
```
C1DataGrid1.AllowDelete = False
```
- C#
```
c1DataGrid1.AllowDelete = false;
```

## Preventing Users from Splitting the Grid

You can prevent users from creating and sizing splits by using the AllowHorizontalSplit, AllowHorizontalSplitSizing, AllowVerticalSplit, and AllowVerticalSplitSizing properties. The properties are all set to **True** by default to allow users to create and size splits at run time.

### In XAML

Set the AllowHorizontalSplit and the AllowVerticalSplit properties to **False** to prevent users from creating horizontal and vertical splits. For example, use the following XAML to prevent users from creating splits:

```
<c1grid:C1DataGrid Name="c1DataGrid1" AllowHorizontalSplit="False"
AllowVerticalSplit="False">
```

Set the AllowHorizontalSplitSizing and AllowVerticalSplit properties to **False** to prevent users from sizing horizontal and vertical splits. For example, use the following XAML to prevent users from sizing splits:

```
<c1grid:C1DataGrid Name="c1DataGrid1" AllowVerticalSplitSizing="False"
AllowHorizontalSplitSizing="False">
```

### In Code

Set the AllowHorizontalSplit and the AllowVerticalSplit properties to **False** to prevent users from creating horizontal and vertical splits. For example, use the following code to prevent users from creating splits:

- Visual Basic
```
C1DataGrid1.AllowHorizontalSplit = False
C1DataGrid1.AllowVerticalSplit = False
```
- C#
```
c1DataGrid1.AllowHorizontalSplit = false;
c1DataGrid1.AllowVerticalSplit = false;
```

Set the AllowHorizontalSplitSizing and AllowVerticalSplitSizing properties to **False** to prevent users from sizing horizontal and vertical splits. For example, use the following code to prevent users from sizing splits:

- Visual Basic
```
C1DataGrid1.AllowHorizontalSplitSizing = False
C1DataGrid1.AllowVerticalSplitSizing = False
```
- C#
```
c1DataGrid1.AllowHorizontalSplitSizing = false;
c1DataGrid1.AllowVerticalSplitSizing = false;
```

## Locking Columns from Being Edited

You can prevent users from altering particular columns by setting a column's **ReadOnly** property to **True**.

### In XAML

In the XAML below the *ProductName* column is added and is set to **ReadOnly** and so cannot be edited by users at run time:

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="Window1"
    Title="Window1" Height="300" Width="300" xmlns:c1grid="clr-
namespace:C1.WPF.C1DataGrid;assembly=C1.WPF.C1DataGrid">
    <Grid>
```

```
      <c1grid:C1DataGrid Name="C1DataGrid1" HorizontalAlignment="Left"
VerticalAlignment="Top">
            <c1grid:C1DataGrid.Columns>
                  <c1grid:Column Caption="Product Name"
ColumnName="ProductName" ReadOnly="True"/>
            </c1grid:C1DataGrid.Columns>
      </c1grid:C1DataGrid>

    </Grid>
</Window>
```

**In Code**

In the code below, the *ProductName* column is set to **ReadOnly** and so cannot be edited by users at run time:

- Visual Basic
```
C1DataGrid1.Columns("ProductName").ReadOnly = True
```
- C#
```
c1DataGrid1.Columns["ProductName"].ReadOnly = true;
```

### Locking the Grid from Being Edited

You can prevent users from altering the grid by setting the AllowEdit property to **False**.

**In the Designer**

Select the **C1DataGrid** control, locate the AllowEdit property in the Properties window, and set it to **False** to prevent users from editing the grid.

**In XAML**

Set the AllowEdit property to **False** to prevent users from editing the grid. For example, use the following XAML to prevent users from editing the grid:

```
<c1grid:C1DataGrid Name="C1DataGrid1" AllowEdit="False"/>
```

**In Code**

In the code below, the AllowEdit property is set to **False** to prevent users from editing the grid:

- Visual Basic
```
C1DataGrid1.AllowEdit = False
```
- C#
```
c1DataGrid1.AllowEdit = false;
```

## Customizing the Grid's Appearance

The following topics detail how to customize the grid's appearance, including how to hide scrollbars in the grid, hide child tables, change the text style of the grid, change the default text that appears in the GroupBy area, and add ToolTips to the grid.

### Hiding Grid Scrollbars

You can disable users from scrolling the grid at run time by hiding the horizontal and or vertical scrollbars. By default the HorizontalScrollbarPlacement property is set to **Bottom** and the VerticalScrollbarPlacement property is set to **Right** and the grid can be scrolled. By setting the HorizontalScrollbarPlacement and VerticalScrollbarPlacement properties to **None**, scrollbars will not be visible and you will prevent users from scrolling the grid.

### In XAML

Prevent users from scrolling the grid at run time by adding `HorizontalScrollbarPlacement="None" VerticalScrollbarPlacement="None"` to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" HorizontalScrollbarPlacement="None"
VerticalScrollbarPlacement="None"/>
```

### In Code

Prevent users from scrolling the grid at run time by adding the following code to your project:

- Visual Basic

```
C1DataGrid1.HorizontalScrollbarPlacement =
HorizontalScrollbarPlacement.None
C1DataGrid1.VerticalScrollbarPlacement = VerticalScrollbarPlacement.None
```

- C#

```
c1DataGrid1.HorizontalScrollbarPlacement =
HorizontalScrollbarPlacement.None;
c1DataGrid1.VerticalScrollbarPlacement = VerticalScrollbarPlacement.None;
```

## Hiding Child Tables

If you choose, you can prevent the grid from displaying hierarchical child data tables. For information about hierarchical data views, see Setting Up Hierarchical Data Views (page 55). By default the AllowHierarchicalData property is set to **True** and child tables are visible in the grid when the grid is automatically generated (AutoGenerateColumns is **True**). For details, see Viewing Child Tables (page 69). By setting the AllowHierarchicalData property to **False**, you can prevent child tables from being displayed.

### In XAML

Prevent child grids from being displayed by adding `AllowHierarchicalData="False"` to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" AllowHierarchicalData="False"/>
```

### In Code

Prevent users from scrolling the grid at run time by adding the following code to your project:

- Visual Basic

```
C1DataGrid1.AllowHierarchicalData = False
```

- C#

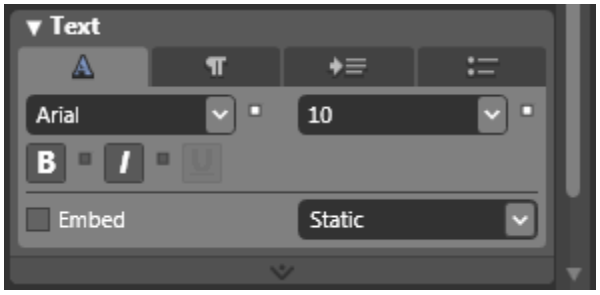
```
c1DataGrid1.AllowHierarchicalData = false;
```

## Changing Text Style

You can change the appearance of the text in the grid by using the **Text** properties in the C1DataGrid Properties window, through XAML, or through code.

### At Design Time

To change the font of the grid to Arial 10pt in the Properties window at design time, navigate to the Properties window and change the C1DataGrid.**FontFamily** property to **Arial** and the C1DataGrid.**FontSize** property to **10**:




**In XAML**

For example, to change the font of the grid to Arial 10pt in XAML add `FontFamily="Arial" FontSize="10"` to the `<c1grid:C1DataGrid>` tag so that it appears similar to the following:

```
<c1grid:C1DataGrid Name="c1DataGrid1" ItemsSource="{Binding
Path=ProductsDataSet.Products, ElementName=Window, Mode=Default}"
FontFamily="Arial" FontSize="10">
```

**In Code**

For example, to change the font of the grid to Arial 10pt add the following code to your project:

- Visual Basic

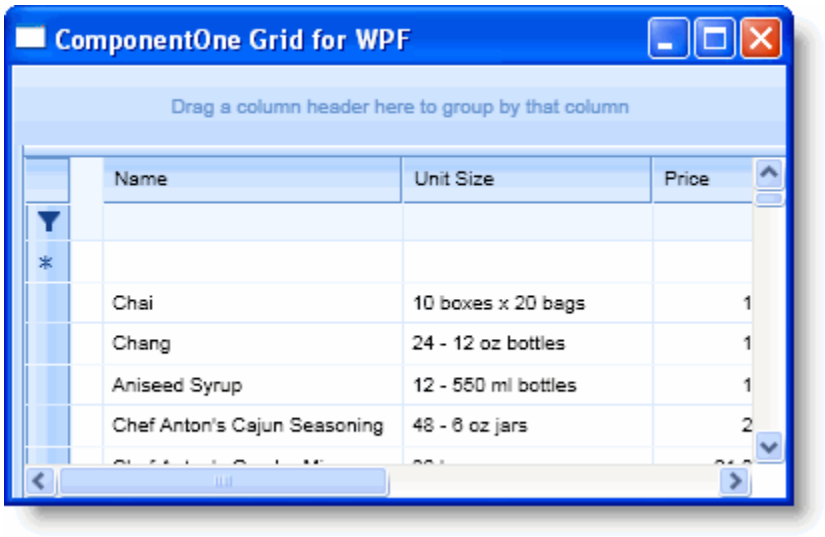
```
C1DataGrid1.FontSize = "10"
C1DataGrid1.FontFamily = New System.Windows.Media.FontFamily("Arial")
```

- C#

```
c1DataGrid1.FontSize = 10;
c1DataGrid1.FontFamily = new System.Windows.Media.FontFamily("Arial");
```

**Run your project and observe:**

The grid content will appear in Arial 10pt font:




## Changing Text in the GroupBy Area

The GroupBy area appears at the top of the grid and allows users to group columns. By default the GroupBy area of the grid is visible and includes the text "Drag a column header here to group by that column". This text is replaced by column headers when a column is dragged into the GroupBy area. You can change this text to

provide instructions or suggestions for the user with the GroupByCaption property in the C1DataGrid Properties window, through XAML, or through code.

**At Design Time**

For example, to change the text of the GroupBy area to be blank, navigate to the Properties window and delete the default text next to the GroupByCaption property.

**In XAML**

For example, to change the text of the GroupBy area to be blank in XAML add `GroupByCaption=""` to the `<c1grid:C1DataGrid>` tag so that it appears similar to the following:

```
<c1grid:C1DataGrid Name="c1DataGrid1" ItemsSource="{Binding
Path=ProductsDataSet.Products, ElementName=Window, Mode=Default}"
GroupByCaption="">
```

**In Code**

To change the text of the GroupBy area to be blank, add the following code to your project:

- Visual Basic

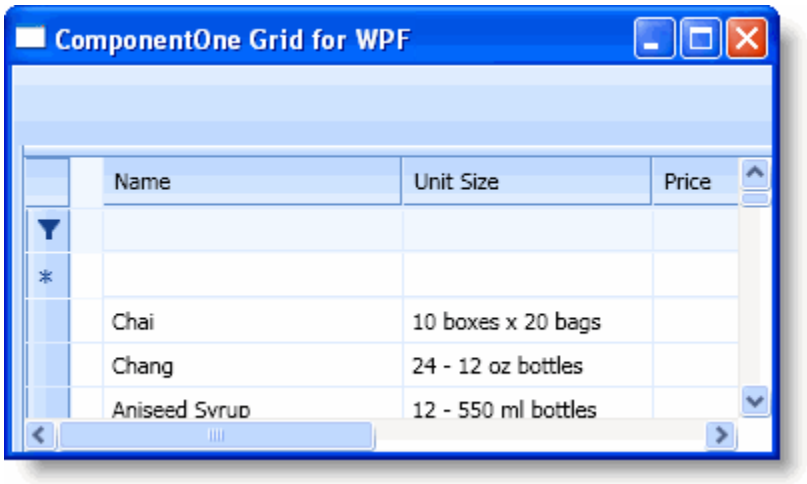
```
C1DataGrid1.GroupByCaption = ""
```

- C#

```
c1DataGrid1.GroupByCaption = "";
```

**Run your project and observe:**

The GroupBy area will appear blank:




## Adding ToolTips to the Grid

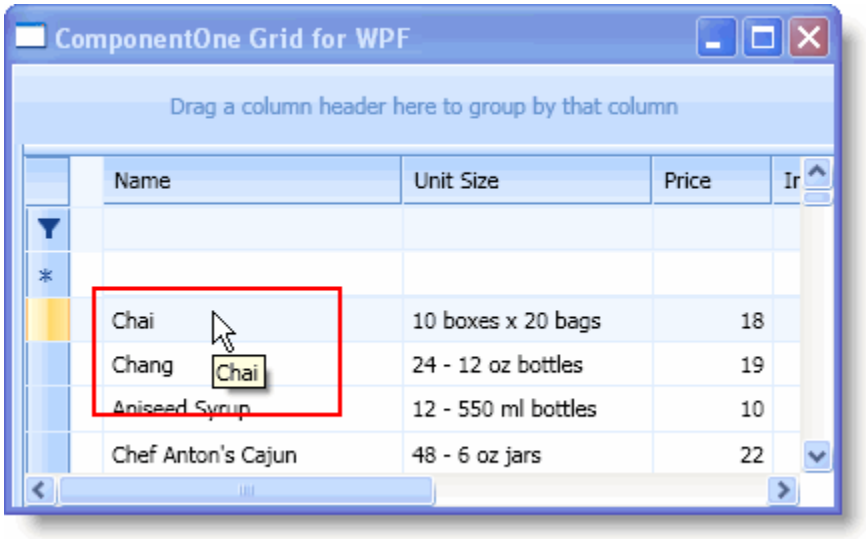
You can add ToolTips to **Grid for WPF** to present information about the content of grid cells. For example, you can add ToolTips to each cell noting the content of that cell using the ItemCellPresenter class and binding the value of the ItemCellPresenter.**ToolTip** property to the ItemCell"s Value property.

To do so, add the following XAML just after the `<Grid>` tag to add a style resource to the project:

```
<Grid.Resources>
    <Style TargetType="{x:Type c1grid:ItemCellPresenter}">
        <Setter Property="ToolTip" Value="{Binding
RelativeSource={RelativeSource Self}, Path=ItemCell.Value}"/>
    </Style>
</Grid.Resources>
```

**Run your application and observe:**

Grid cells will now include a ToolTip containing the content of that grid cell:




## Formatting a Column as Currency

You can easily customize the way a column is formatted using the Format property. If you'd like to format the column as currency using templates instead, you can see the Formatting a Column as Currency Using Templates (page 163) topic for details. The following steps assume that you've bound the grid to the *Products* table of the C1NWind.mdb database (see Binding Grid for WPF to a Database (page 138) for more information). In this example, you'll format the *UnitPrice* column as currency at design time, in XAML, or in code.

**Note:** In the example below, the longer `Format="{}{0:C}"` format string was used for context. You can also use the briefer form, `Format="C"`, with the same results.

### At Design Time

To format the *UnitPrice* column as currency at design time, complete the following steps:

1. Click once on the C1DataGrid control to select it.
2. Navigate to the Properties window and click the ellipsis button next to the Columns collection. This will open the **Column Collection Editor** which will enable you to create column definitions. Note that these steps assume that no columns have been previously defined.
3. In the **Column Collection Editor**, click the **Add another item** button to create a new column definition. A new column will appear in the left-side **Items** pane.
4. In the right-side Properties pane, locate the PropertyName property in the **Behavior** tab and set it to "UnitPrice".
5. Scroll down to the Format property in the **Miscellaneous** tab, and set it to "{0:C}".

### In XAML

To format the *UnitPrice* column as currency at design time add `Format="{}{0:C}"` to the `<c1grid:C1DataGrid.Columns>` tag so that it appears similar to the following:

```
<c1grid:C1DataGrid Name="c1DataGrid1" ItemsSource="{Binding
Path=ProductsDataSet.Products, ElementName=Window, Mode=Default}" />
```

```
    <c1grid:C1DataGrid.Columns>
        <c1grid:Column PropertyName="UnitPrice" Caption="Price"
HeaderCellWidth="60" Format="{}{0:C}"/>
    </c1grid:C1DataGrid.Columns>
</c1grid:C1DataGrid>
```

**In Code**

To format the *UnitPrice* column as currency, add the following code to your project:

- Visual Basic

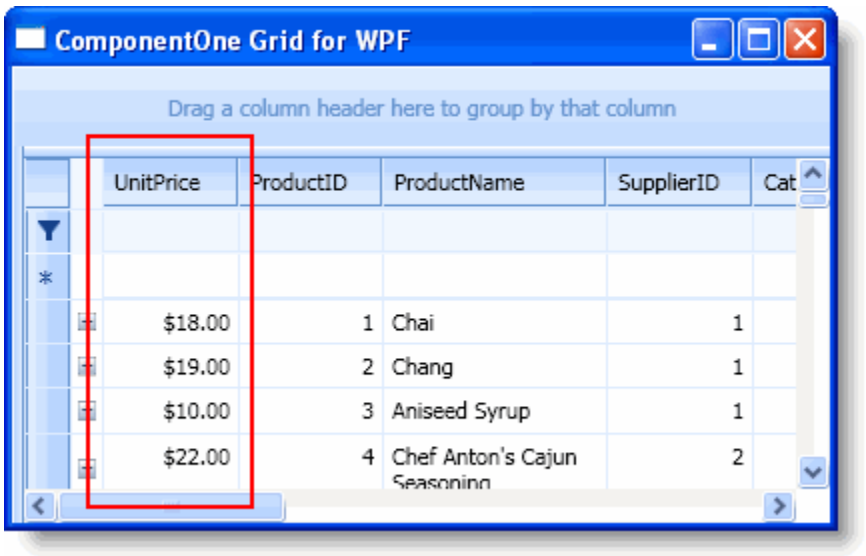
```
' Add a new column
Dim unitPrice As New Column
' Add the column to the grid and set properties
C1DataGrid1.Columns.Add(unitPrice)
unitPrice.PropertyName = "UnitPrice"
unitPrice.Caption = "Price"
' Format the column as currency
unitPrice.Format = "{0:C}"
```

- C#

```
// Add a new column
Column unitPrice = new Column();
// Add the column to the grid and set properties
c1DataGrid1.Columns.Add(unitPrice);
unitPrice.PropertyName = "UnitPrice";
unitPrice.Caption = "Price";
// Format the column as currency
unitPrice.Format = "{0:C}";
```

**Run your project and observe:**

The *UnitPrice* column will appear as currency:




# Using Templates

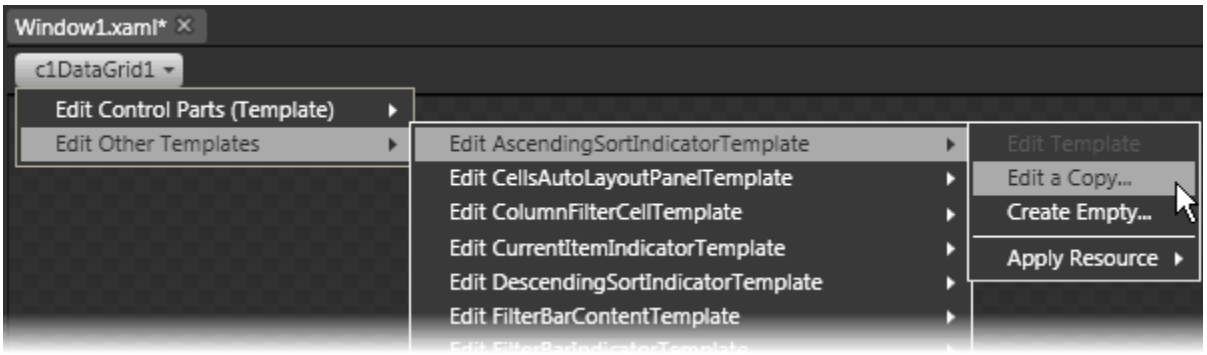
The following topics detail how you can use customized templates to change the grid's appearance and behavior. In the following topics you'll change the appearance of the sort indicator, change the appearance of

columns in the grid using a UniversalItemContentTemplate, format cell content meeting specific criteria, and format a column as currency.

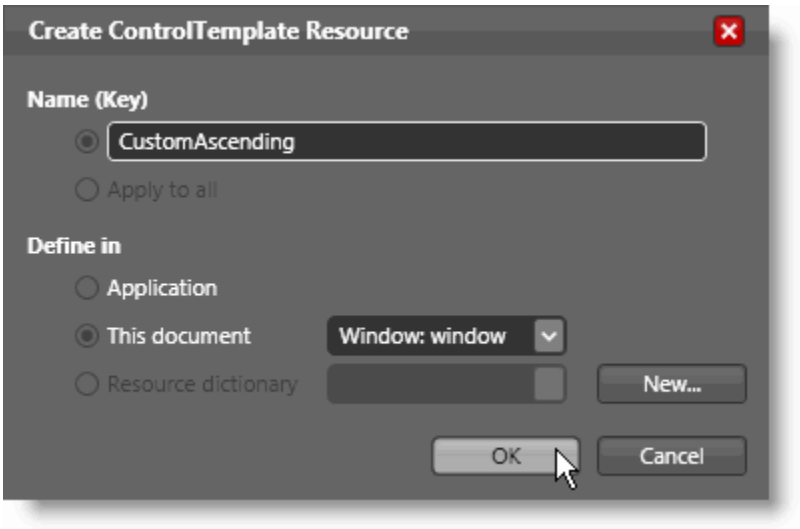
## Changing the Appearance of the Sort Indicator

You can easily change the appearance of the **Grid for WPF** sort indicator in Expression Blend by using the AscendingSortIndicatorTemplate and DescendingSortIndicatorTemplate properties. In the following steps you'll visually edit the sort indicator templates in Blend to change the color of the sort indicator arrows in design view and using XAML.

1. Complete steps 1 and 2 of the Grid for WPF Quick Start (page 25) to create a new WPF project, add a C1DataGrid control to the project, and bind the project to a data source.
2. In Blend, click once on the C1DataGrid control to select **C1DataGrid1**.
3. Click the **C1DataGrid1** menu, and select **Edit Other Templates | Edit AscendingSortIndicatorTemplate | Edit a Copy**.

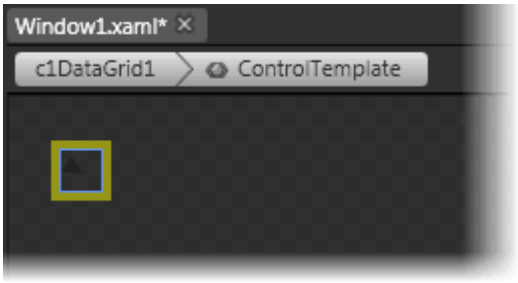



4. In the **Create ControlTemplate Resource** dialog box change the template's name to "CustomAscending" and click **OK**.



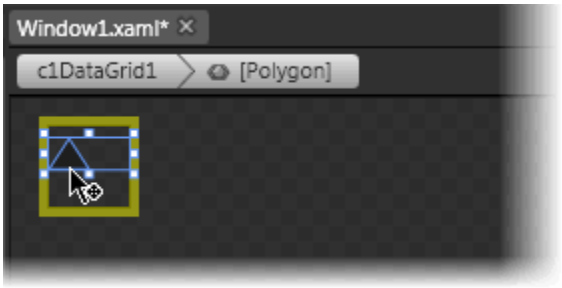

The new control template will appear in Blend and in the XAML the `<ControlTemplate x:Key="CustomAscending">` tag is added to the `<Window.Resources>` tag.

Notice that the template consists of a black polygon shape. You'll customize the appearance of this polygon to change the appearance of the ascending sort indicator.

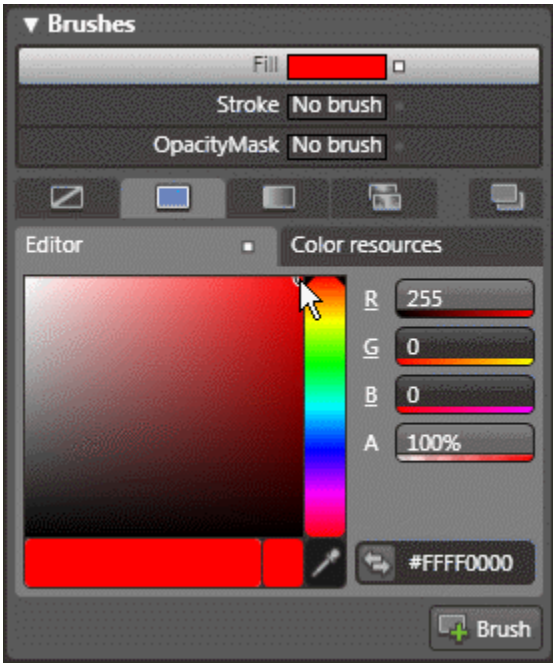



You can zoom in to better view the template.

5. Click the **Polygon** shape within the Control Template to select it.




6. Navigate to the Properties window and make sure that **Fill** brush and the **Solid color brush** tab are selected in the **Brushes** tab.
7. Select a new Fill color for the sort indicator; in the image below the color red (or #FFFF0000) was selected:




8. Select the **Stroke** brush and the **Solid color brush** tab and choose a color for the sort indicator's outline. Here the color black (#FF000000) was selected:


Note that the XAML of the control template now appears similar to the following:

```
<ControlTemplate x:Key="CustomAscending">

      <Polygon Margin="0,3,0,0" VerticalAlignment="Top" Fill="#FFFF0000"
Points="0,7.6 5,0 10,7.6" Stroke="#FF000000"/>
```

```
</ControlTemplate>
```

In the next step you'll add the descending sort item indicator using XAML.

9. To add a descending sort item indicator template, add the following XAML within the `<Window.Resources>` tag and under the `<ControlTemplate x:Key="CustomAscending">` tag:

```
<ControlTemplate x:Key="CustomDescending">
        <Polygon Margin="0,3,0,0" VerticalAlignment="Top"
Fill="#FFFF0000" Points="0,0 5,7.6 10,0" Stroke="#FF000000"/>
</ControlTemplate>
```

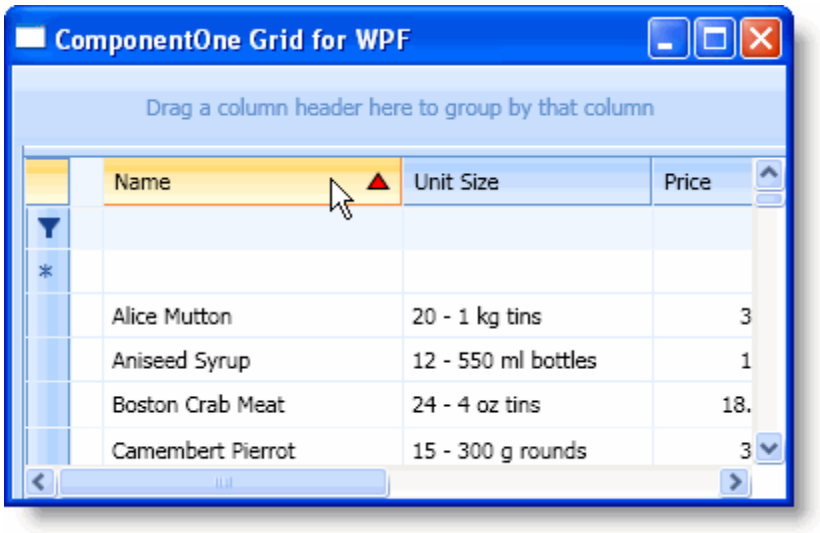
10. To set the DescendingSortIndicatorTemplate, add the `DescendingSortIndicatorTemplate="{DynamicResource CustomDescending}"` XAML to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:

```
<c1grid:C1DataGrid Name="c1DataGrid1" ItemsSource="{Binding
Path=ProductsDataSet.Products, ElementName=Window1, Mode=Default}"
AscendingSortIndicatorTemplate="{DynamicResource CustomAscending}"
DescendingSortIndicatorTemplate="{DynamicResource CustomDescending}">
```

You've finished changing the appearance of the **Grid for WPF** sort indicator in Expression Blend in the design view and in XAML.

**Run your application and observe:**

Click a column header and notice that the appearance of the ascending sort indicator has changed to an outlined red triangle:




Click the column header again and note that the appearance of the descending sort indicator has also changed.

## Using the UniversalItemContentTemplate

You can use the UniversalItemContentTemplate property to define a single template that determines the arrangement of column presenters in several grid parts such as the grid item, header and filter bar. To create a new template and set the apperance of columns, complete the following steps:

1. Complete steps 1 and 2 of the Grid for WPF Quick Start (page 25) to create a new WPF project, add a C1DataGrid control to the project, and bind the project to a data source. (Note that you can complete the following with an unbound C1DataGrid control, but you may need to adapt the following steps.)

2. Create a new universal template by adding the following XAML to your project after the opening `<Window>` tag to add a template to your window resources:

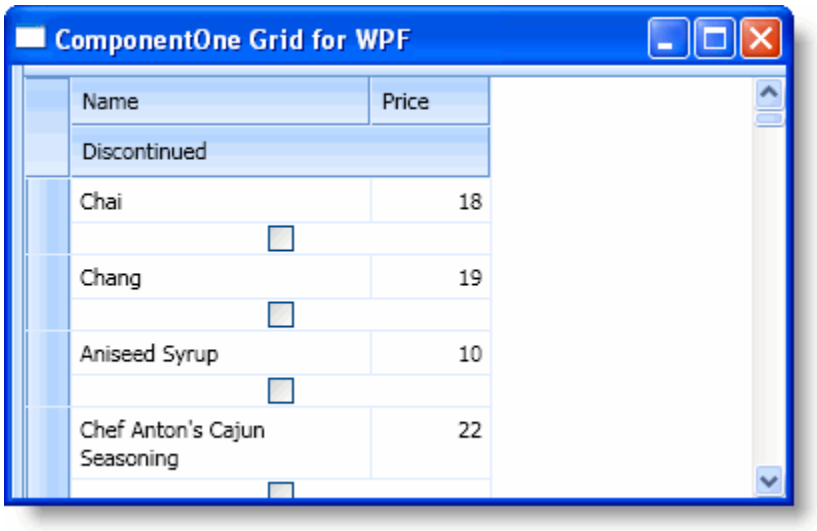
```
<Window.Resources>
   <ControlTemplate x:Key="universalTemplate">
         <Grid>
               <Grid.RowDefinitions>
                     <RowDefinition/>
                        <RowDefinition/>
               </Grid.RowDefinitions>
               <Grid.ColumnDefinitions>
                        <ColumnDefinition/>
                        <ColumnDefinition />
               </Grid.ColumnDefinitions>
               <c1grid:ColumnPresenter ColumnName="ProductName"
Grid.Row="0" Grid.Column="0"/>
               <c1grid:ColumnPresenter ColumnName="UnitPrice"
Grid.Row="0" Grid.Column="1"/>
               <c1grid:ColumnPresenter ColumnName="Discontinued"
Grid.Row="1" Grid.Column="0" Grid.ColumnSpan="2"/>
         </Grid>
   </ControlTemplate>
</Window.Resources>
```

3. Set the UniversalItemContentTemplate property to the template you just created by adding `UniversalItemContentTemplate="{DynamicResource universalTemplate}"` to the `<c1grid:C1DataGrid>` tag so that it looks similar to the following:

```
<c1grid:C1DataGrid Name="C1DataGrid1" ItemsSource="{Binding
Path=GridDataSet.Products, ElementName=Window1, Mode=Default}"
AutoGenerateColumns="False"
UniversalItemContentTemplate="{DynamicResource universalTemplate}">
```

**Run the program and observe:**

Your grid will appear similar to the following:



## Formatting Cells Meeting Specific Criteria

In this topic you'll learn how to use templates to format cells meeting specific criteria. By following the steps outlined below, you'll create a template that formats cell content above the number 50. You'll then apply that template to the *UnitPrice* column in a grid so that products with a price above 50 will be formatted in a red font. This topic assumes that you have completed steps 1 and 2 of the Grid for WPF Quick Start (page 25) and have bound the C1DataGrid to the **C1NWind.mdb** database.

Complete the following steps to create and apply a template:

1. Add the mscorlib namespace to your project by adding `xmlns:sys="clr-namespace:System;assembly=mscorlib"` to the `<Window>` tag so that it looks similar to the following:

```
<Window
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:sys="clr-namespace:System;assembly=mscorlib"
    xmlns:my="clr-
namespace:C1.WPF.C1DataGrid;assembly=C1.WPF.C1DataGrid"
x:Class="Window1"
    Height="240" Width="411" x:Name="Window">
```

2. Create a new ControlTemplate resource in your project by adding the following XAML just below the `<Window>` tag:

```
<Window.Resources>
 <ControlTemplate x:Key="unitPriceShowTemplate">
  <Grid>
   <TextBlock Name="tb" Text="{Binding Value}" TextWrapping="Wrap"
Margin="3"
    TextAlignment="Right"/>
   <c1grid:MethodCaller Name="compareValue" ObjectType="sys:Decimal"
    MethodName="Compare" MethodParameters0="{Binding Value}"
    PermanentCall="True">
    <c1grid:MethodCaller.MethodParameters1>
     <sys:Decimal>50</sys:Decimal>
    </c1grid:MethodCaller.MethodParameters1>
    </c1grid:MethodCaller>
  </Grid>
  <ControlTemplate.Triggers>
    <Trigger SourceName="compareValue" Property="Result">
     <Trigger.Value>
      <sys:Int32>1</sys:Int32>
     </Trigger.Value>
      <Setter TargetName="tb" Property="Foreground" Value="Red"/>
    </Trigger>
   </ControlTemplate.Triggers>
 </ControlTemplate>
</Window.Resources>
```

   Note that the template uses the MethodCaller element that calls the Decimal.Compare static method. The result of this method is processed in Trigger, which sets the Foreground color to red if Decimal.Compare returned 1.

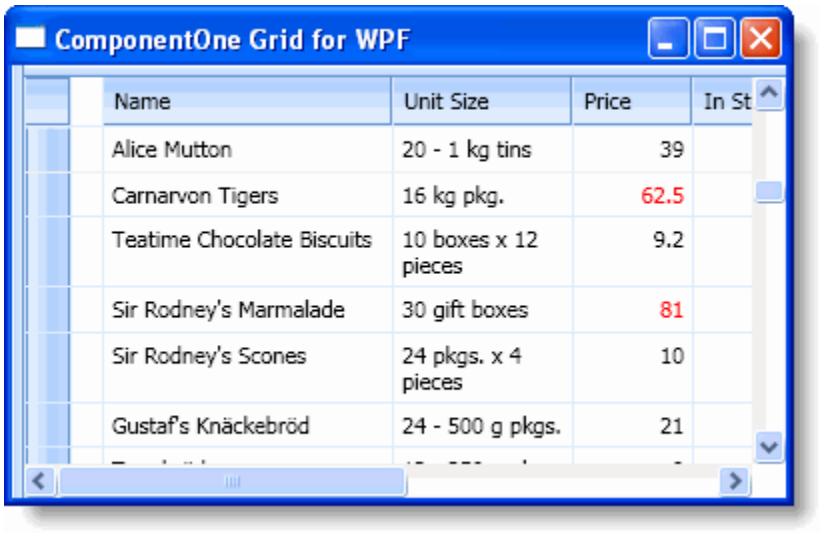
3. Now add the following XAML within the `<c1grid:C1DataGrid>` tag to define the columns and call the ItemCellShowContentTemplate in the *UnitPrice* column:

```
<c1grid:C1DataGrid Name="c1DataGrid1" ItemsSource="{Binding
Path=ProductsDataSet.Products, ElementName=Window, Mode=Default}"
AutoGenerateColumns="False">
   <c1grid:C1DataGrid.Columns>
```

```
        <c1grid:Column PropertyName="ProductName" Caption="Name"
HeaderCellWidth="150"/>
        <c1grid:Column PropertyName="UnitPrice" Caption="Price"
HeaderCellWidth="60" ItemCellShowContentTemplate="{StaticResource
unitPriceShowTemplate}"/>
        <c1grid:Column PropertyName="QuantityPerUnit" Caption="Unit
Size" HeaderCellWidth="125"/>
        <c1grid:Column PropertyName="UnitsInStock" Caption="In Stock"
HeaderCellWidth="75"/>
        <c1grid:Column PropertyName="UnitsOnOrder" Caption="On Order"
HeaderCellWidth="80"/>
        <c1grid:Column PropertyName="ReorderLevel" Caption="Reorder
Level" HeaderCellWidth="100"/>
        <c1grid:Column PropertyName="Discontinued"
Caption="Discontinued" HeaderCellWidth="95"/>
   </c1grid:C1DataGrid.Columns>
</c1grid:C1DataGrid>
```

**Run the project and observe:**

The grid displays items in the *UnitPrice* column above 50 in a red font:




## Formatting a Column as Currency Using Templates

In this topic you'll learn how to use templates to format a column using templates. Note that you can also more easily format the column's appearance using the Format property. For more information, see the Formatting a Column as Currency (page 155) topic.

In the following steps you'll create a custom content template that uses an **IValueConverter** to format the appearance of a column as currency and assign the template to a column's ItemCellShowContentTemplate property. By following the steps outlined below, you'll set the *UnitPrice* column in a grid so that values in the column are formatted as currency. This topic assumes that you have completed steps 1 and 2 of the Grid for WPF Quick Start (page 25) and have bound the C1DataGrid to the **C1NWind.mdb** database.

Complete the following steps:

1. Open your project in Visual Studio 2008, right-click the window, and select **View Code** to open the Code Editor.
2. Add the following import statements to the top of the Window1.xaml.cs (or Window1.xaml.vb) file:

- Visual Basic

```
Imports System.Globalization
```

- C#

```
using System.Globalization;
```

3. Add the following code to the project to create the **IValueConverter** that transform the current value of a column to currency:

- Visual Basic

```
Public Class CurrencyConverter
    Implements IValueConverter
    Public Function Convert(ByVal value As Object, ByVal targetType As
System.Type, ByVal parameter As Object, ByVal culture As
System.Globalization.CultureInfo) As Object Implements
System.Windows.Data.IValueConverter.Convert
        If (Not value Is Nothing) AndAlso ((Not
Object.Equals(String.Empty, value))) Then
            Try
                ' This will convert the string value provided to a
double value before formatting.
                Dim tempD As Double = System.Convert.ToDouble(value,
CultureInfo.CurrentCulture)
                Return String.Format(CultureInfo.CurrentCulture,
"{0:C}", tempD)
                ' Catch exceptions.
            Catch e1 As FormatException
            Catch e2 As OverflowException
            End Try
        End If
        ' Return the string value.
        Return String.Format(CultureInfo.CurrentCulture, "{0}", value)
    End Function
    Public Function ConvertBack(ByVal value As Object, ByVal targetType
As System.Type, ByVal parameter As Object, ByVal culture As
System.Globalization.CultureInfo) As Object Implements
System.Windows.Data.IValueConverter.ConvertBack
        Return Double.Parse(TryCast(value, String),
NumberStyles.Currency, CultureInfo.CurrentCulture)
    End Function
End Class
```

- C#

```
[ValueConversion(typeof(double), typeof(string))]
public class CurrencyConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object
parameter, CultureInfo culture)
    {
        if ((value != null) && (!object.Equals(string.Empty, value)))
        {
            try
            {
                // This will convert the string value provided to a
double value before formatting.
                double tempD = System.Convert.ToDouble(value,
CultureInfo.CurrentCulture);
                // Return the string value.
```

```
                return string.Format(CultureInfo.CurrentCulture,
"{0:C}", tempD);
            }
            // Catch exceptions.
            catch (FormatException)
            {
            }
            catch (OverflowException)
            {
            }
        }
        return string.Format(CultureInfo.CurrentCulture, "{0}", value);
    }
    public object ConvertBack(object value, Type targetType, object
parameter, CultureInfo culture)
    {
        return double.Parse(value as string, NumberStyles.Currency,
CultureInfo.CurrentCulture);
    }
}
```

4. Add the local namespace to your project by adding `xmlns:local="clr-namespace:ProjectName"` to the `<Window>` tag, where *ProjectName* is the name of your project, so that it looks similar to the following:

```
<Window
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:ProjectName"
    xmlns:my="clr-
namespace:C1.WPF.C1DataGrid;assembly=C1.WPF.C1DataGrid"
x:Class="Window1"
    Height="240" Width="411" x:Name="Window">
```

5. Create a new ControlTemplate resource in your project by adding the following XAML just below the `<Window>` tag:

```
  <Window.Resources>
    <local:CurrencyConverter x:Key="convertCurrency"/>
    < ControlTemplate x:Key="ItemCellShowContentTemplate_currency">
      <TextBlock Text="{Binding Value, Converter={StaticResource
convertCurrency}}"/>
    </ControlTemplate >
  </Window.Resources>
```

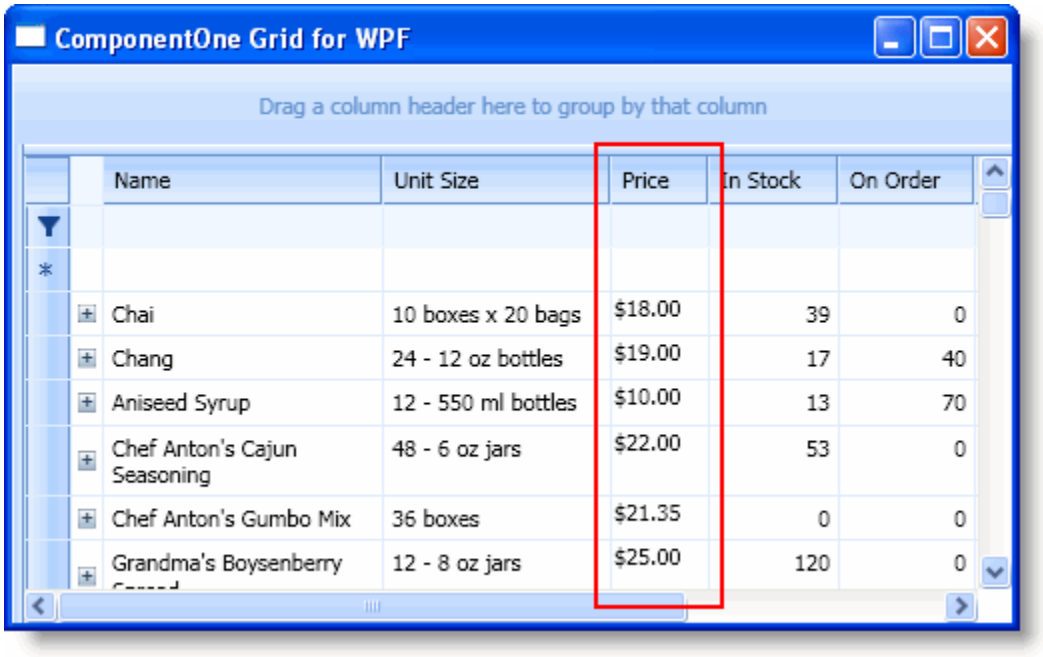
6. Now add the following XAML within the `<c1grid:C1DataGrid>` tag to define the columns and call the ItemCellShowContentTemplate in the *UnitPrice* column:

```
<c1grid:C1DataGrid Name="c1DataGrid1" ItemsSource="{Binding
Path=ProductsDataSet.Products, ElementName=Window, Mode=Default}"
AutoGenerateColumns="False">
   <c1grid:C1DataGrid.Columns>
         <c1grid:Column PropertyName="ProductName" Caption="Name"
HeaderCellWidth="150"/>
         <c1grid:Column PropertyName="UnitPrice" Caption="Price"
HeaderCellWidth="60" ItemCellShowContentTemplate="{StaticResource
ItemCellShowContentTemplate_currency}"/>
```

```
        <c1grid:Column PropertyName="QuantityPerUnit" Caption="Unit
Size" HeaderCellWidth="125"/>
        <c1grid:Column PropertyName="UnitsInStock" Caption="In Stock"
HeaderCellWidth="75"/>
        <c1grid:Column PropertyName="UnitsOnOrder" Caption="On Order"
HeaderCellWidth="80"/>
        <c1grid:Column PropertyName="ReorderLevel" Caption="Reorder
Level" HeaderCellWidth="100"/>
        <c1grid:Column PropertyName="Discontinued"
Caption="Discontinued" HeaderCellWidth="95"/>
   </c1grid:C1DataGrid.Columns>
</c1grid:C1DataGrid>
```

**Run the project and observe:**

The grid displays items in the *UnitPrice* column as currency:




# Reordering Columns Programmatically

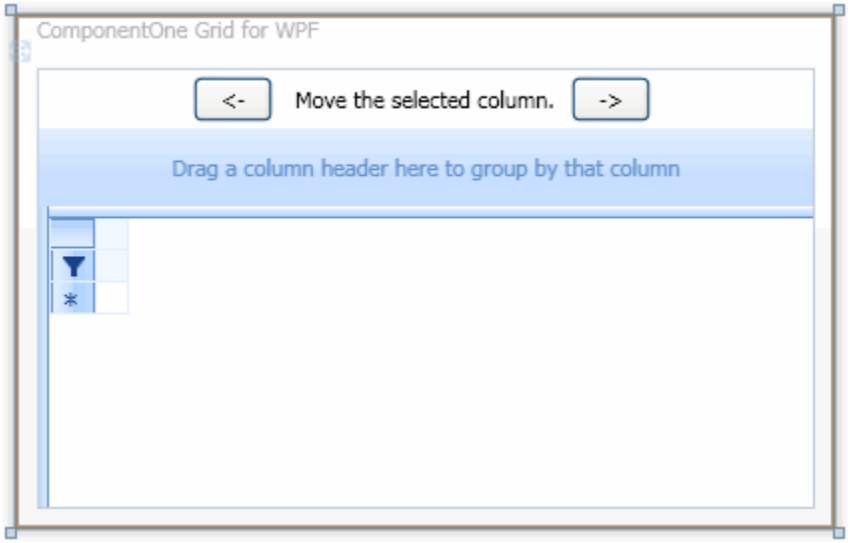
You can programmatically reorder columns in **C1DataGrid**'s actual column collections to change the visual order of columns. In this topic you'll use the Move method to move columns. You'll create an application with two buttons, enabling the user to move the selected column to the right or left. To implement the Move method, you'll create a separate method that takes a shift direction (left or right) as a parameter and call this method from the buttons' event handlers.

Complete the following steps:

1. Complete steps 1 and 2 of the Grid for WPF Quick Start (page 25) to create a new WPF project, add a C1DataGrid control to the project, and bind the project to a data source. (Note that you can complete the following with an unbound C1DataGrid control, but you may need to adapt the following steps.)
2. From the Toolbox add two **Button** controls and a **Label** control to the window.
3. Resize all the controls and, in the Properties window, set the following properties:
   - Set **Button1**'s **Content** property to "<-".

- Set **Label1**'s **Content** property to "Move the selected column."
- Set **Button2**'s **Content** property to "->".

The window will look similar to the following:




4. Double-click **Button1** to create the **Button1_Click** event handler and switch to the code editor.
5. Add code to the **Button1_Click** event handler to call the **MoveColumn** method, which you'll add in the next step:

- Visual Basic

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.Windows.RoutedEventArgs) Handles Button1.Click
    ' Move the selected column to the left.
    Me.MoveColumn(True)
End Sub
```

- C#

```
private void Button1_Click(object sender, System.EventArgs e)
{
    // Move the selected column to the left.
    MoveColumn(true);
}
```

6. Add the following code above the **Button1_Click** event to create the **MoveColumn** method:

- Visual Basic

```
' Method that takes a shift direction (left or right) as a parameter.
Private Sub MoveColumn(ByVal toTheLeft As Boolean)
    If C1DataGrid1.SelectedItemCellPresenter Is Nothing Then
        Exit Sub
    End If
    Dim currentIdx As Integer =
C1DataGrid1.ActualColumns.IndexOf(C1DataGrid1.SelectedItemCellPresenter
.ItemCell.Column)
    Dim newIdx As Integer = If(toTheLeft, currentIdx - 1, currentIdx +
1)
    If newIdx >= 0 AndAlso newIdx < C1DataGrid1.ActualColumns.Count
Then
        C1DataGrid1.ActualColumns.Move(currentIdx, newIdx)
```

```
    End If
End Sub
```

- C#

```
// Method that takes a shift direction (left or right) as a parameter.
private void MoveColumn(bool toTheLeft)
{
    if (c1DataGrid1.SelectedItemCellPresenter == null)
        return;
    int currentIdx =

c1DataGrid1.ActualColumns.IndexOf(c1DataGrid1.SelectedItemCellPresenter
.ItemCell.Column);
    int newIdx = toTheLeft ? currentIdx - 1 : currentIdx + 1;
    if (newIdx >= 0 && newIdx < c1DataGrid1.ActualColumns.Count)
        c1DataGrid1.ActualColumns.Move(currentIdx, newIdx);
}
```

7. Return to Design view and double-click **Button2** to create the **Button2_Click** event handler.
8. Add code to the **Button2_Click** event to call the **MoveColumn** method:

- Visual Basic

```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.Windows.RoutedEventArgs) Handles Button1.Click
    ' Move the selected column to the right.
    Me.MoveColumn(False)
End Sub
```

- C#

```
private void Button2_Click(object sender, System.EventArgs e)
{
    // Move the selected column to the right.
    MoveColumn(false);
}
```

**Run your program and observe:**

Select a column and click the left column button – the column will move to the left. Click the right column button; the column will move to the right.

# Exporting Data to an Excel File

Starting with the 2009 v3 release, you can export the content of your grid application to a Microsoft Excel file.

In this topic you'll add a button to the project that, when clicked, will export the grid's content. You'll use the ExportToExcel method to export grid content, and you'll change the export's settings using ExcelExportSettings.

Complete the following steps:

1. Complete steps 1 and 2 of the Grid for WPF Quick Start (page 25) to create a new WPF project, add a C1DataGrid control to the project, and bind the project to a data source. (Note that you can complete the following with an unbound C1DataGrid control, but you may need to adapt the following steps.)
2. From the Toolbox add a **Button** control to the window, resize it, and, in the Properties window, set **Button1**'s **Content** property to "Export".
3. Double-click **Button1** to create the **Button1_Click** event handler and switch to Code view.
4. Add code to the **Button1_Click** event handler to call the ExportToExcel method:

- Visual Basic

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.Windows.RoutedEventArgs) Handles Button1.Click
    ' Customize export settings.
    Dim ExportSet as New ExcelExportSettings
    Me.ExportSet.FileFormat = ExcelFormat.Excel97
    Me.ExportSet.IncludeColumns.Add("ProductName")
    ' Export data to the selected file.
    Me.C1DataGrid1.ExportToExcel("c:\products.xls", ExportSet)
End Sub
```

- C#

```
private void Button1_Click(object sender, System.EventArgs e)
{
    // Customize export settings.
    ExcelExportSettings ExportSet = new ExcelExportSettings();
    this.ExportSet.FileFormat = ExcelFormat.Excel97;
    this.ExportSet.IncludeColumns.Add("ProductName");
    // Export data to the selected file.
    this.c1DataGrid1.ExportToExcel(@"c:\products.xls", ExportSet);
}
```

**Run your program and observe:**

Click the **Export** button. The *ProductName* column will be saved as a Microsoft Excel 97/2003 file named products.xls in the C:\ directory.