ComponentOne

DataGrid 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

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 DataGrid for WPF Overview

Add advanced data visualization to your WPF applications with **ComponentOne DataGrid™ for WPF**. The robust data-bound **C1DataGrid** control makes it easy to display, edit, and analyze tabular data in WPF applications.

Getting Started

Get started with the following topics:

- Key Features (page 14)

- Quick Start (page 17)

- Task-Basked Help (page 65)

What's New in DataGrid for WPF

This documentation was last revised for the 2010 v3 release. The following changes and enhancements were made in this release:

- **Printing Support**

  You can now print your grids. In the 2010 v3 release, printing support was added to **DataGrid for WPF**. See the Print methods for more information about printing grids.

- **Excel Export Support**

  You can export to Microsoft Excel files. In the 2010 v3 release an export to Excel file format feature was added to **DataGrid for WPF**. See the Save methods for more information.

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.

Installing DataGrid for WPF

The following sections provide helpful information on installing **ComponentOne DataGrid 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 DataGrid for WPF**, the following DLLs are installed: • C1.WPF.dll • C1.WPF.Expression.Design.dll • C1.WPF.VisualStudio.Design.dll • C1.WPF.Expression.Design.4.dll

- C1.WPF.VisualStudio.Design.4.dll
- C1.WPF.DataGrid

In addition, the following files from the Microsoft WPF Toolkit are also installed:

- WPFToolkit.dll
- WPFToolkit.Design.dll
- WPFToolkit.VisualStudio.Design.dll

For more information about the Microsoft WPF Toolkit, see CodePlex. The C1.WPF.dll and WPFToolkit.dll assemblies are required for deployment.

H2Help	Contains Microsoft Help 2.0 integrated documentation for all Studio components.
HelpViewer	Contains Microsoft Help Viewer Visual Studio 2010 integrated documentation for all Studio components.
C1WPFDataGrid\XAML	Contains the full XAML definitions of C1DataGrid styles and templates which can be used for creating your own custom styles and templates.

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.
Studio for WPF	Contains samples for **DataGrid for WPF**.

Samples can be accessed from the **ComponentOne Studio for WPF ControlExplorer**. To view samples, on your desktop, click the **Start** button and then click **All Programs | ComponentOne | Studio for WPF | Samples | WPF ControlExplorer**.

System Requirements

System requirements include the following:

Operating Systems:	Microsoft Windows® XP with Service Pack 2 (SP2) Windows Vista™ Windows 7 Windows 2008 Server
Environments:	.NET Framework 3.5 or later

	Visual Studio® 2005 extensions for .NET Framework 2.0 November 2006 CTP Visual Studio® 2008 or later
Microsoft® Expression® Blend Compatibility:	**DataGrid for WPF** includes design-time support for Expression Blend.

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

Installing Demonstration Versions

If you wish to try **ComponentOne DataGrid 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 DataGrid for WPF

To uninstall **ComponentOne Studio 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.

To uninstall **ComponentOne Studio for WPF** integrated help:

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

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**
  Microsoft integrated ComponentOne documentation can be installed with each of our products, and documentation 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 DataGrid 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.dll
- C1.WPF.DataGrid.dll

In addition, the following file from the Microsoft WPF Toolkit is also installed and is redistributable:

- WPFToolkit.dll

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

About this Documentation

You can create your 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

Microsoft, Windows, Windows Vista, Visual Studio, and Microsoft Expression are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

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 or later. With XAML you can create a graphically rich customized user interface, perform data binding, and much more. For more information on XAML, 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:c1="http://schemas.componentone.com/winfx/2006/xaml"
```

The namespace value is **datagrid** and the XML namespace is **http://schemas.componentone.com/winfx/2006/xaml**. This is a unified namespace; once this is in the project, all ComponentOne WPF controls found in your references will be accessible through XAML (and IntelliSense). Note that you still need to add references to the assemblies for each control you need to use.

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

```
xmlns:MyGrid="http://schemas.componentone.com/winfx/2006/xaml"
```

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">
```

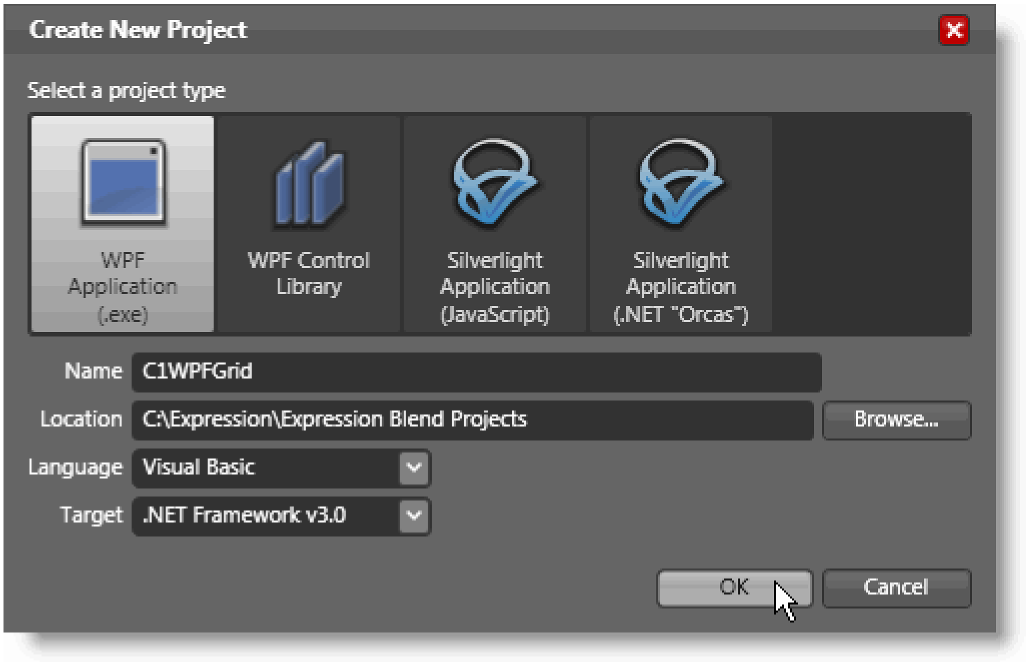
Creating a Microsoft Blend Project

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**.



Name: C1WPFGrid
Location: C:\Expression\Expression Blend Projects
Language: Visual Basic
Target: .NET Framework v3.0

A new Blend project with a XAML window is created.

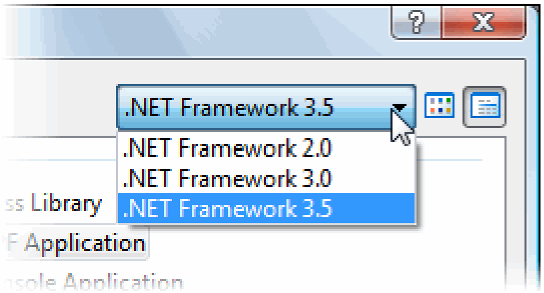
Creating a .NET Project in Visual Studio

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.

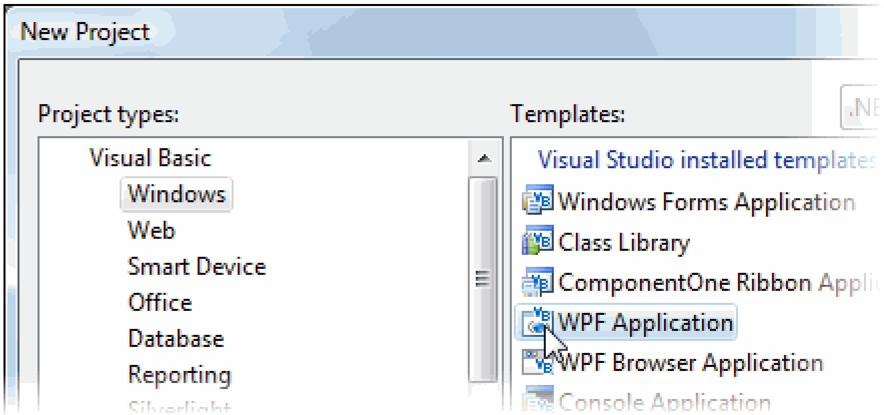
2. Choose the appropriate .NET Framework from the Framework drop-down box in the top-right of the dialog box.



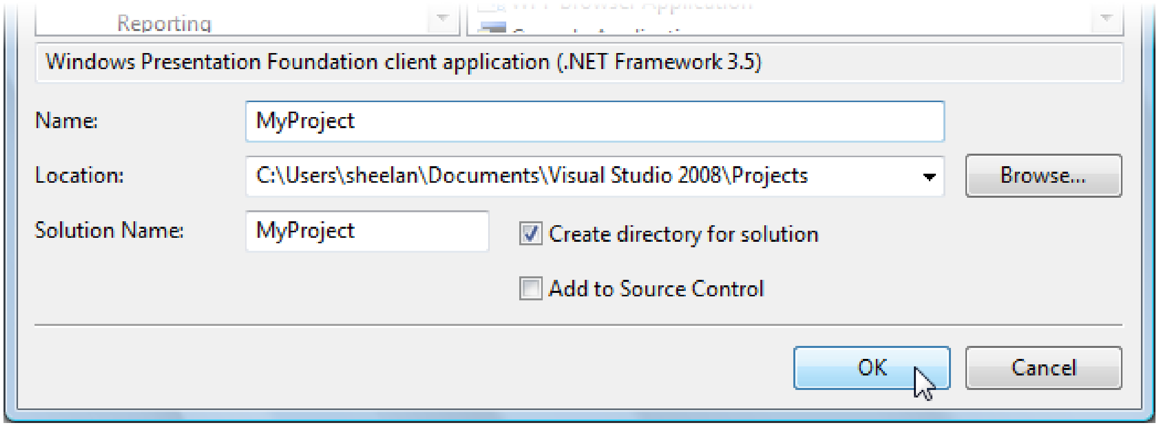
3. Under **Project types**, select either **Visual Basic** or **Visual C#**.

> **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**.



Windows Presentation Foundation client application (.NET Framework 3.5)

Name: MyProject

Location: C:\Users\sheelan\Documents\Visual Studio 2008\Projects

Solution Name: MyProject

☑ Create directory for solution

☐ Add to Source Control

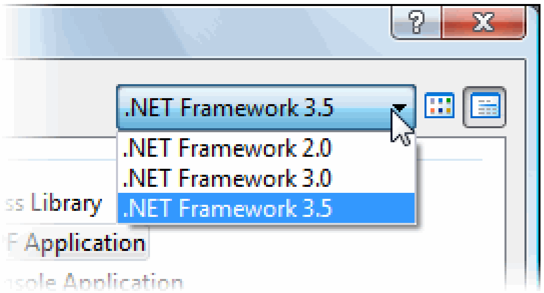
A new Microsoft Visual Studio .NET WPF project is created with a XAML file that will be used to define your user interface and commands in the application.

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 DataGrid for WPF Components to a Blend Project

In order to use C1DataGrid or another **ComponentOne DataGrid for WPF** component in the Design workspace of Blend, you must first add references to the **C1.WPF.dll**, **C1.WPF.DataGrid**, and **WPFToolkit.dll** assemblies and then add the component from Blend's **Asset Library**.

To add a reference to the assembly:

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

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

2. Select **C1.WPF.DataGrid.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.DataGrid assembly, click the Asset Library button » in the Blend Toolbox. The Asset Library appears.
2. Click the **Controls** drop-down arrow and select **All**.
3. Select **C1DataGrid**. The component will appear in the Toolbox below the Asset Library button.
4. Double-click the **C1DataGrid** component in the Toolbox to add it to Window1.xaml.

Adding the DataGrid for WPF Components to a Visual Studio Project

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

ComponentOne DataGrid for WPF provides the following control:

- C1DataGrid

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

Manually Adding DataGrid for WPF to the Toolbox

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

- C1DataGrid

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 **DataGrid 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.DataGrid** namespace. Note that there may be more than one component for each namespace.

Adding DataGrid for WPF to the Window

To add **ComponentOne DataGrid 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 **DataGrid for WPF** assembly, complete the following steps:

1. Select the **Add Reference** option from the **Project** menu of your project.
2. Select the **ComponentOne DataGrid for WPF** assembly from the list on the **.NET** tab or on the **Browse** tab, browse to find the **C1.WPF.DataGrid.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.DataGrid
   ```

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

Key Features

ComponentOne DataGrid for WPF includes several key features, such as:

- **Fully Interactive Grid**

  Enhance the end-user experience by creating a fully interactive grid. **DataGrid for WPF** has many built-in interactive features such as column resizing and reordering, editing, sorting, filtering, grouping, freezing, and selecting. See Run-time Interaction (page 51) for more information.

- **Data Grouping and Totals**

  DataGrid for WPF supports Outlook-style grouping. Simply drag a column header to the area above the grid to group the data. Expandable and collapsible nodes are automatically generated. You can also show calculated aggregate functions or totals in grouped header rows. See Grouping Columns (page 59) for details.

- **Excel-like Filtering**

  By default, **DataGrid for WPF** supports Excel-like filtering. This type of filtering features a drop-down menu on each column allowing users to create a filter condition. See Filtering Columns (page 56) for more information.

- **High Performance**

  DataGrid for Silverlight utilizes both row and column recycling (UI Virtualization) to achieve optimal performance when handling large data sets.

- **Several Built-in Column Types**

  DataGrid for WPF provides many built-in column editors that cover all of the common data types. The built-in editors include text, check box, DateTime picker, combo box and images. You can also choose from a selection of custom column editors including masked text, hyperlink, multi-line text and a color picker. See Column Types (page 24) for details.

- **RowDetails and Hierarchical Support**

  DataGrid also supports a RowDetails template for embedding UIElements inside a collapsible section of each row. For example, just embed another DataGrid and you can create a master-detail grid for displaying hierarchical data. For more information, see Adding Row Details (page 37).

- **Top and Bottom Row Templates**

  With **DataGrid for WPF**'s Top and Bottom row templates you can easily create and add custom rows to the grid. For example, you can design your own filter or total rows and embed any UIElements inside.

- **Multiple Selection Modes**

  Give end-users all of the following cell selection options: single cell, single row, single column, single range, multi-row, multi-column, and multi-range. With **DataGrid for WPF**'s clipboard support, end-users can then easily paste selected cells into any text editor, such as Microsoft Excel.

- **New Row**

  Allow users to add new rows to **DataGrid for WPF** by displaying an empty new row at either the top or bottom of the grid. See Adding Rows to the Grid (page 63) and Setting New Row Visibility (page 46) for details.

- **Custom Rows and Columns**

  Design your own data template for each DataGrid row and create composite columns which can combine data from multiple data fields.

- **Easily Change Colors with ClearStyle**

  DataGrid for WPF supports ComponentOne's new ClearStyle™ technology that allows you to easily change control colors without having to change control templates. With just setting a few color properties you can quickly style the entire grid. For details, see C1DataGrid ClearStyle (page 48).

DataGrid for WPF Quick Start

The following quick start guide is intended to get you up and running with **DataGrid for WPF**. In this quick start you'll start in Visual Studio and create a new project, add **DataGrid 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.

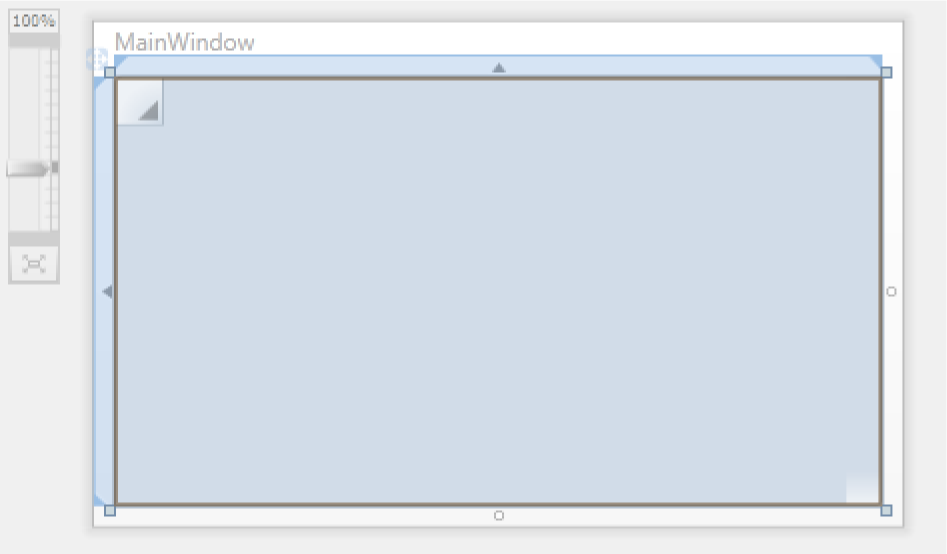
> **Note:** This quick start guide uses the **C1NWind.mdb** database, installed by default in the **ComponentOne Samples\Common** folder installed in your **MyDocuments** folder (**Documents** in 7/Vista). You could also use the standard Microsoft Northwind database instead, NWind.mdb, and adapt the appropriate steps.

Step 1 of 3: Adding Grid for WPF to your Project

In this step you'll begin in Visual Studio to create a grid application using **DataGrid 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.

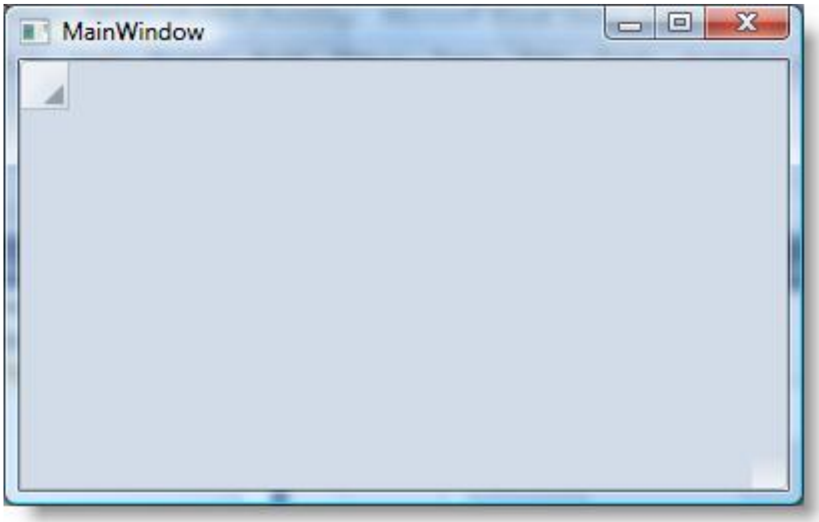
To set up your project and add a C1DataGrid control to your application, complete the following steps:

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 11).
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:



What You've Accomplished

Run the application and observe that the grid application will appear similar to the following image:



You've successfully created a very basic grid application, but the grid is blank. In the next step you'll add a data source to your project and bind the grid to the data source.

Step 2 of 3: 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 in the **Data Source Configuration Wizard** and click **Next**.
3. If the **Choose a Database Model** screen appears, select **Dataset** and click **Next**.
4. On the **Choose Your Data Connection** screen, 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.
5. In the **Add Connection** dialog box, click the **Browse** button and locate **C1NWind.mdb** in the samples installation directory. Select it and click **Open**.
6. Click the **Test Connection** button to make sure that you have successfully connected to the database or server and click **OK**.
7. 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.
8. Click the **Next** button to continue. If a dialog box appears asking if you would like to add the data file to your project and modify the connection string, click **No** since it is not necessary to copy the database to your project.
9. In the next window, confirm that the **Yes, save the connection as** check box is selected and a name has been automatically entered in the text box ("C1NWindConnectionString"). Click **Next** to continue.
10. In the **Choose Your Database Objects** window, you can select the tables and fields that you would like in your dataset. Select the **Products** table (you may need to expand the **Tables** node first) and change the DataSet name to **ProductsDS**.

11. Click **Finish** to exit the wizard. The **ProductsDS.xsd** files now appear in the Solution Explorer.
12. In the Solution Explorer, double-click the **Window1.xaml.cs** (or Window1.xaml.vb) file to switch to code view.
13. 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.DataGrid
    Imports ProjectName.ProductsDSTableAdapters
    ```

    - C#

    ```
    using C1.WPF.DataGrid;
    using ProjectName.ProductsDSTableAdapters;
    ```

14. Add the following code to the **MainWindow** class to retrieve the products and order details data from the database:

    - Visual Basic

    ```
    Class MainWindow
        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)
                End If
                Return _productsDataSet
            End Get
        End Property

        Public Sub New()
            InitializeComponent()
        End Sub
    End Class
    ```

    - C#

    ```
    public partial class MainWindow : Window
    {
        private ProductsDS _productsDataSet = null;
        public ProductsDS ProductsDataSet
        {
            get
            {
                if (_productsDataSet == null)
                {
                    _productsDataSet = new ProductsDS();
                    ProductsTableAdapter prodTA = new
    ProductsTableAdapter();
                    prodTA.Fill(_productsDataSet.Products);
                }
                return _productsDataSet;
            }
        }

        public MainWindow()
        {
    ```

```
        InitializeComponent();
    }
}
```

15. Press F5 to run your project to ensure that everything is working correctly. Notice that the grid still appears blank in the running application; you will need to complete binding before content appears.
16. Close the running application and return to the project.
17. Add code to the **MainWindow** constructor so that it looks like the following:

- Visual Basic

```
Public Sub New()
    InitializeComponent()
    Me.C1DataGrid1.ItemsSource = ProductsDataSet.Products
End Sub
```

- C#

```
public MainWindow()
{
    InitializeComponent();
    this.c1DataGrid1.ItemsSource = ProductsDataSet.Products;
}
```

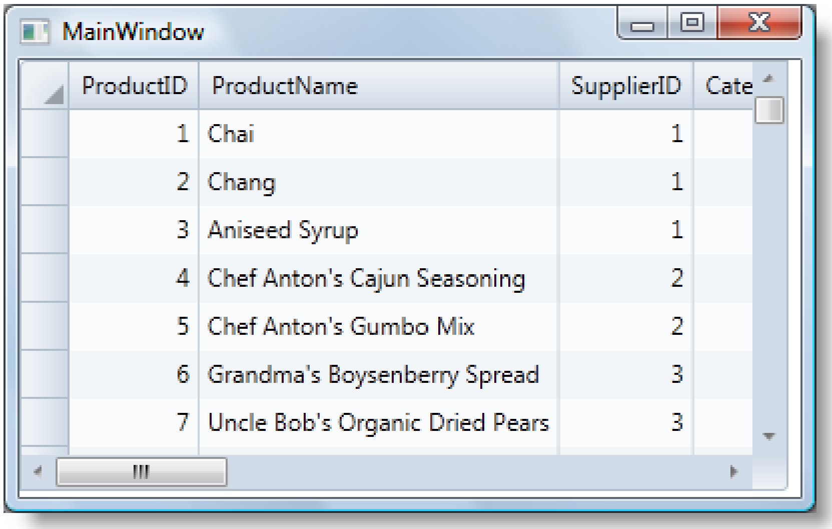
This code will bind the grid to the **Products** table in the C1NWind database.

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

```
<c1:C1DataGrid HorizontalAlignment="Left" Name="C1DataGrid1"
VerticalAlignment="Top" Height="215" Width="384"/>
```

Run the program and observe:

The grid is now populated with data from the **Products** table:



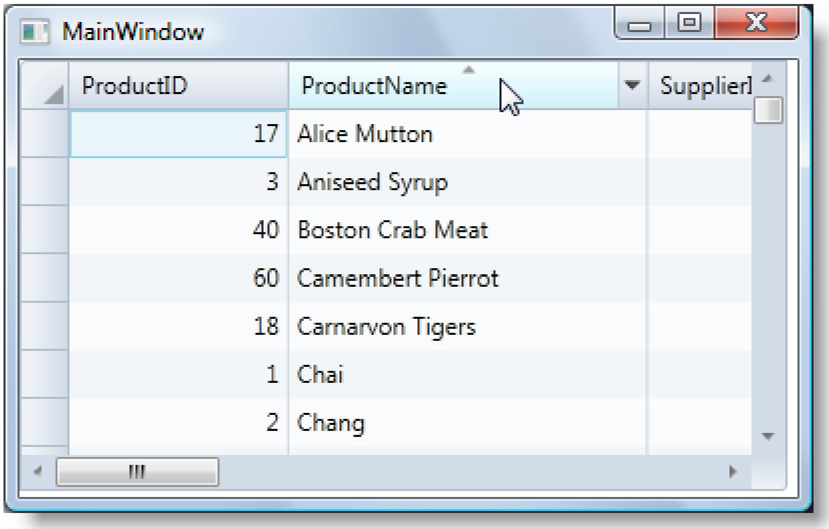
ProductID	ProductName	SupplierID	Cate
1	Chai	1	
2	Chang	1	
3	Aniseed Syrup	1	
4	Chef Anton's Cajun Seasoning	2	
5	Chef Anton's Gumbo Mix	2	
6	Grandma's Boysenberry Spread	3	
7	Uncle Bob's Organic Dried Pears	3	

You've successfully bound **DataGrid for WPF**'s C1DataGrid control to a data source. In the next step you'll explore some of the run-time interactions that are possible in your grid application.

Step 3 of 3: Running the Grid Application

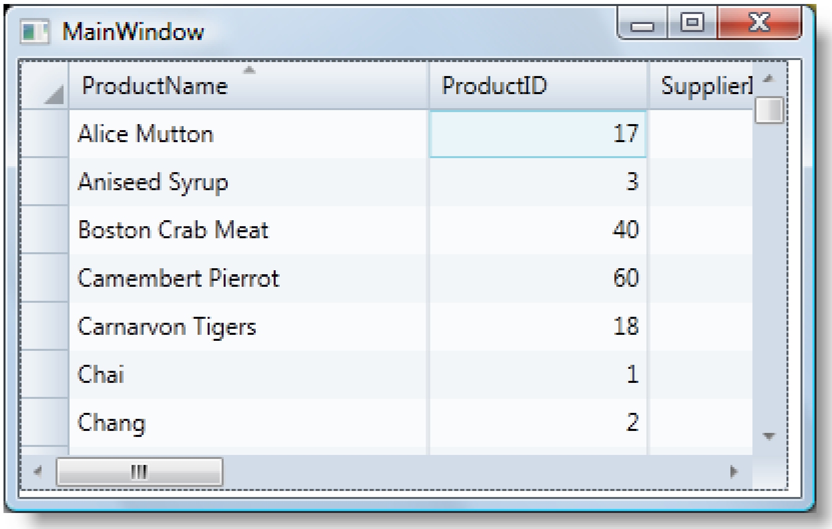
Now that you've created a grid application and bound the grid to a database, the only thing left to do is run your application. To run your grid application and observe **Grid for WPF**'s run-time behavior, complete the following steps:

1. From the **Debug** menu, select **Start Debugging** to view how your grid application will appear at run time.
2. Click the *ProductName* 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.



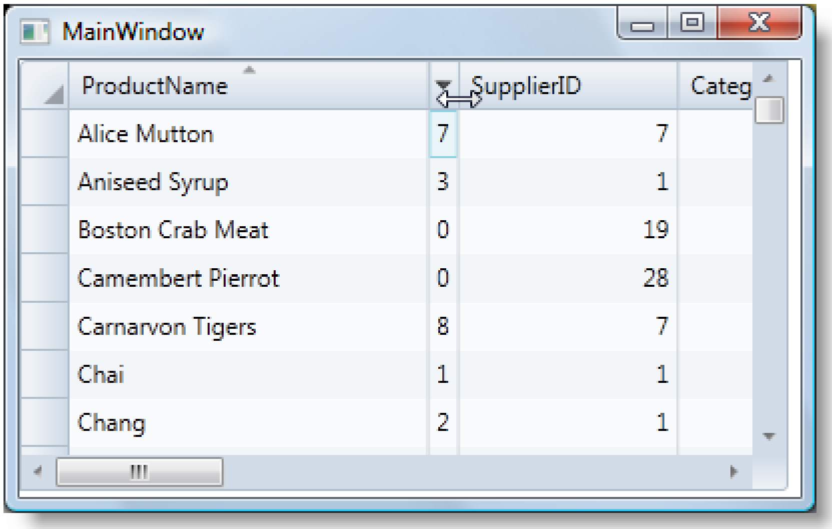
ProductID	ProductName	SupplierI
17	Alice Mutton	
3	Aniseed Syrup	
40	Boston Crab Meat	
60	Camembert Pierrot	
18	Carnarvon Tigers	
1	Chai	
2	Chang	

Re-order the columns by clicking the *ProductName* column header and dragging it in front of the *ProductID* column header. The *ProductName* column will now appear as the first column in the grid:



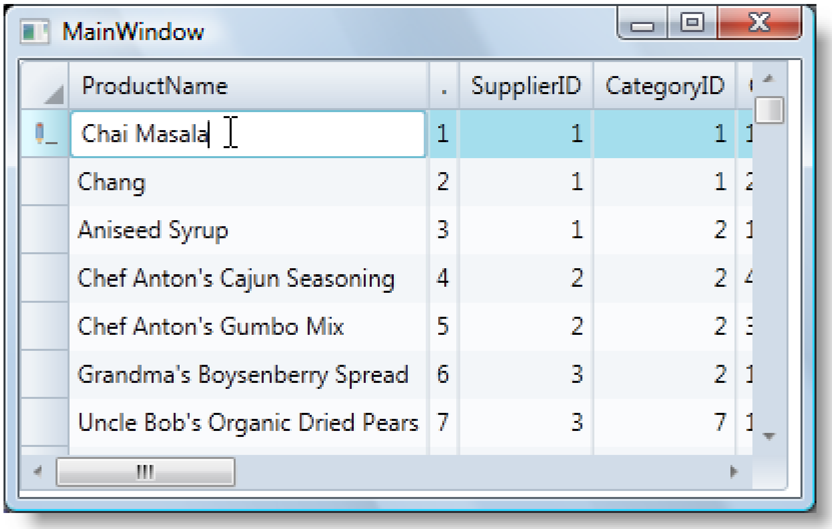
ProductName	ProductID	SupplierI
Alice Mutton	17	
Aniseed Syrup	3	
Boston Crab Meat	40	
Camembert Pierrot	60	
Carnarvon Tigers	18	
Chai	1	
Chang	2	

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



ProductName		SupplierID	Categ
Alice Mutton	7	7	
Aniseed Syrup	3	1	
Boston Crab Meat	0	19	
Camembert Pierrot	0	28	
Carnarvon Tigers	8	7	
Chai	1	1	
Chang	2	1	

4. Click once on a cell, edit the contents of that cell, and press the ENTER key.



ProductName	.	SupplierID	CategoryID
Chai Masala	1	1	1
Chang	2	1	1
Aniseed Syrup	3	1	2
Chef Anton's Cajun Seasoning	4	2	2
Chef Anton's Gumbo Mix	5	2	2
Grandma's Boysenberry Spread	6	3	2
Uncle Bob's Organic Dried Pears	7	3	7

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

Working with DataGrid for WPF

ComponentOne DataGrid for WPF allows you to select, edit, add, delete, filter, group, and sort the items displayed in the table generated by the **C1DataGrid** component.

The columns of a table created using the **C1DataGrid** component correspond to the fields in a data source. You can control which columns are displayed, the types of columns to display, and the appearance of the whole table.

Using the **AutoGenerateColumns** property, you can generate columns automatically, manually, or both. Setting this property to **True** (default) creates columns automatically when the **ItemsSource** property is set. Setting this property to **False** allows you to specify the columns to display, which are added to the **Columns** collection.

> **Note:** By default explicitly declared columns are rendered first, followed by automatically generated columns. You can change the order of rendered columns by setting the **DisplayIndex** property of the column. Automatically generated columns are now added to the **Columns** collection.

Class Hierarchy

The following list summarizes the class relationships between the more important classes included in the **DataGrid for WPF**:

- C1.WPF.DataGrid.C1DataGrid : System.Windows.Controls.Control
  Encapsulates most of the grid functionality. This component is shown in Visual Studio's Toolbox.
- C1.WPF.DataGrid.DataGridColumn : System.Object
  Represents a column in the grid.
- C1.WPF.DataGridColumnCollection : System.Object
  Represents the collection of columns of the data grid.
- C1.WPF.DataGrid.DataGridColumnHeaderPresenter : System.Windows.Controls.Control
  Content control that represent the header of a column; this control contains the sort, resize and filter elements.
- C1.WPF.DataGrid.DataGridRow : System.Object
  Represents a row in the grid.
- C1.WPF.DataGrid.DataGridRowCollection : System.Object
  Collection of rows.
- C1.WPF.DataGrid.DataGridCell : System.Object
  Represents an individual grid cell.

Data Binding

ComponentOne DataGrid 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). You can use the **C1DataGrid.ItemsSource** property to bind the **C1DataGrid**.

To bind the grid, simply set the **ItemsSource** property to an **IEnumerable** implementation. Each row in the data grid will be bound to an object in the data source, and each column in the data grid will be bound to a property of the data object.

Note that in order for the **C1DataGrid** user interface to update automatically when items are added to or removed from the source data, the control must be bound to a collection that implements **INotifyCollectionChanged**, such as an **ObservableCollection<(Of <(T>)>)**.

For steps on binding a **C1DataGrid** control to an XML data source, see the DataGrid for WPF Quick Start (page 17).

Defining Columns

You can use **ComponentOne DataGrid for WPF**'s **Columns** collection to programmatically add, insert, remove, and change any columns in the control at run time. You can also specify columns in XAML with or without automatically generating columns.

Creating your own columns enables you to use additional column types, such as the **DataGridTemplateColumn** type or custom column types. The **DataGridTemplateColumn** type provides an easy way to create a simple custom column. The **CellTemplate** and **CellEditingTemplate** properties enable you to specify content templates for both display and editing modes.

Generating Columns

By default, the **C1DataGrid** control generates columns automatically, based on the type of data, when you set the **ItemsSource** property. The generated columns are of type **DataGridCheckBoxColumn** for bound Boolean (and nullable Boolean) properties, and of type **DataGridTextColumn** for bound string data, **DataGridComboBoxColumn** for bound enum data, **DataGridDateTimeColumn** for bound date/time data, and **DataGridNumericColumn** for bound numeric data. Bound undefined data is displayed in a **DataGridBoundColumn** type column. If a property does not have a String or numeric value type, the generated text box columns are read-only and display the data object's **ToString** value.

You can prevent automatic column generation by setting the **AutoGenerateColumns** property to **False**. This is useful if you want to create and configure all columns explicitly. Alternatively, you can let the data grid generate columns, but handle the **AutoGeneratingColumn** event to customize columns after creation. To rearrange the display order of the columns, you can set the **DisplayIndex** property for individual columns.

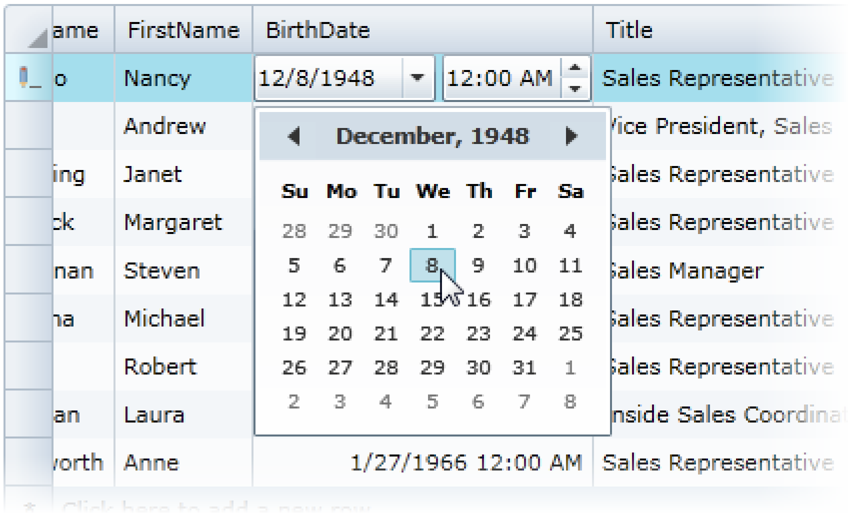
Column Types

ComponentOne DataGrid for WPF's **C1DataGrid** control provides a flexible way to display a collection of data in rows and columns by providing many built-in column editors that cover all of the common data types. Built-in column types include:

Column Type	Description
DataGridBoundColumn	A column that can bind to a property in the grid's data source. This is the default column type for bound undefined data.
DataGridTextColumn	A text column. This is the default column type for bound string data.
DataGridCheckBoxColumn	A check box column. This is the default column type for bound Boolean data.
DataGridComboBoxColumn	A combo box column. This is the default column type for bound enumeration type data.
DataGridDateTimeColumn	A date time column (see below for an image). This is the default column type for bound date/time data.
DataGridImageColumn	An image column.
DataGridNumericColumn	A numeric column. This is the default column type for bound numeric data (the format will be inferred from the type. For example, if the type is int the format will not contain decimal places).
DataGridTemplateColumn	A template column for hosting custom content.

CustomColumns	A custom column. See the C1DataGrid_Demo sample for examples ofcustom columns like a Composite Column, Color Column, Gif Column, Hyperlink Column, Masked Text Column, Multi line Text Column, and so on.

These column types can provide built-in input validation; for example the **DataGridDateTimeColumn** column includes a calendar for selecting a date:

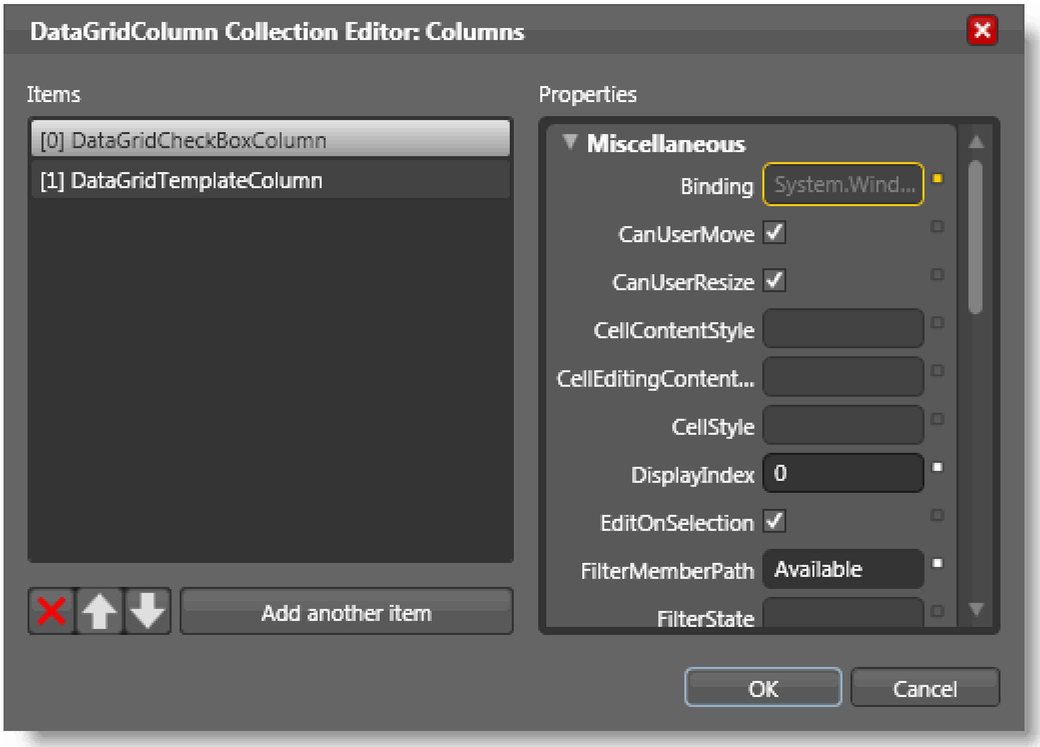


ame	FirstName	BirthDate	Title
o	Nancy	12/8/1948 12:00 AM	Sales Representative
	Andrew		ice President, Sales
ing	Janet		ales Representative
ck	Margaret		ales Representative
nan	Steven		ales Manager
na	Michael		ales Representative
	Robert		ales Representative
an	Laura		nside Sales Coordina
orth	Anne	1/27/1966 12:00 AM	Sales Representative

Explicitly Defining Columns

If you choose, you can explicitly define columns. If the **AutoGenerateColumns** property is **False** only the columns you have defined will appear in the grid.

In Microsoft Expression Blend, you can use the **DataGridColumn Collection Editor** to define columns in your grid. Select the **C1DataGrid** control, and in the Properties window select the ellipsis button next to the **Columns(Collection)** item in the **Miscellaneous** group. The **DataGridColumn Collection Editor** dialog box will appear:



DataGridColumn Collection Editor: Columns

Items

[0] DataGridCheckBoxColumn
[1] DataGridTemplateColumn

Properties

Miscellaneous

Binding System.Wind...
CanUserMove ☑
CanUserResize ☑
CellContentStyle
CellEditingContent...
CellStyle
DisplayIndex 0
EditOnSelection ☑
FilterMemberPath Available
FilterState

You can also define custom columns in the grid in XAML by using the **Columns** collection.

For example:

- XAML

```
<c1:C1DataGrid x:Name="grid" Grid.Row="1" Grid.ColumnSpan="2"
Margin="5" AutoGeneratingColumn="grid_AutoGeneratingColumn"
CanUserAddRows="False" ColumnHeaderHeight="30" >
    <c1:C1DataGrid.Columns>
        <!--
        Custom check box column.
        Adds a check box to the header and listens to its events.
        -->
        <c1:DataGridCheckBoxColumn Binding="{Binding Available,
Mode=TwoWay}" DisplayIndex="0" SortMemberPath="Available"
FilterMemberPath="Available" MinWidth="108" >
            <c1:DataGridColumn.Header>
                <StackPanel Orientation="Horizontal"
HorizontalAlignment="Left">
                    <TextBlock Margin="6,0,6,0"
VerticalAlignment="Center" Text="Available"/>
                    <CheckBox HorizontalAlignment="Left"
IsHitTestVisible="True" VerticalAlignment="Center" Grid.Column="1"
Checked="CheckBox_Checked" Unchecked="CheckBox_Checked"
Loaded="CheckBox_Loaded"/>
                </StackPanel>
            </c1:DataGridColumn.Header>
        </c1:DataGridCheckBoxColumn>
        <!--
        Custom "merged" column made with a DataGridTemplateColumn.
        You can also inherit from DataGridTemplateColumn and set
        this configuration in the constructor to make your XAML
        cleaner.
        -->
        <c1:DataGridTemplateColumn>
            <c1:DataGridTemplateColumn.Header>
                <local:MergedColumnEditor ControlMode="Header" />
            </c1:DataGridTemplateColumn.Header>
            <c1:DataGridTemplateColumn.CellTemplate>
                <DataTemplate>
                    <local:MergedColumnEditor ControlMode="Cell" />
                </DataTemplate>
            </c1:DataGridTemplateColumn.CellTemplate>
            <c1:DataGridTemplateColumn.CellEditingTemplate>
                <DataTemplate>
                    <local:MergedColumnEditor ControlMode="EditingCell"
/>
                </DataTemplate>
            </c1:DataGridTemplateColumn.CellEditingTemplate>
        </c1:DataGridTemplateColumn>
    </c1:C1DataGrid.Columns>
</c1:C1DataGrid>
```

Customizing Automatically Generated Columns

You can customize columns even if columns are automatically generated. If the **AutoGenerateColumns** property is set to **True** and columns are automatically generated, you can customize how generated columns are displayed in code by handling the **C1DataGrid.AutoGeneratingColumn** event.

Adding the AutoGeneratingColumn Event Handler

Complete the following steps to add the **AutoGeneratingColumn** event handler:

1. Switch to Code view and add an event handler for the **AutoGeneratingColumn** event, for example:
   - Visual Basic

   ```
   Private Sub C1DataGrid1_AutoGeneratingColumn(ByVal sender As
   System.Object, ByVal e As
   C1.WPF.DataGrid.DataGridAutoGeneratingColumnEventArgs) Handles
   C1DataGrid1.AutoGeneratingColumn
       ' Add code here.
   End Sub
   ```

   - C#

   ```
   private void C1DataGrid1_AutoGeneratingColumn(object sender,
   C1.WPF.DataGrid.DataGridAutoGeneratingColumnEventArgs e)
   {
       // Add code here.
   }
   ```

2. Switch to Source view and add the event handler to instances of the **C1DataGrid** control, for example:

   ```
   <c1:C1DataGrid x:Name="c1DataGrid1" AutoGenerateColumns="True"
   AutoGeneratingColumn=" c1DataGrid1_AutoGeneratingColumn"></c1:C1DataGrid>
   ```

You can now add code to the **AutoGeneratingColumn** event handler to customize the appearance and behavior of automatically generated columns. Below are examples of customizing column formatting and behavior.

Canceling Column Generation

You can cancel the generation of specific columns in the **AutoGeneratingColumn** event. For example, you can use the following code to cancel the generation of Boolean columns in the grid:

- Visual Basic

```
Private Sub C1DataGrid1_AutoGeneratingColumn(ByVal sender As
System.Object, ByVal e As
C1.WPF.DataGrid.DataGridAutoGeneratingColumnEventArgs) Handles
C1DataGrid1.AutoGeneratingColumn
    ' Cancel automatic generation of all Boolean columns.
    If e.Property.PropertyType Is GetType(Boolean) Then
        e.Cancel = True
    End If
End Sub
```

- C#

```
private void c1DataGrid1_AutoGeneratingColumn(object sender,
C1.WPF.DataGrid.DataGridAutoGeneratingColumnEventArgs e)
{
    // Cancel automatic generation of all Boolean columns.
    if (e.Property.PropertyType == typeof(bool))
        e.Cancel = true;
}
```

Changing a Column Header

In the **AutoGeneratingColumn** event you can change the text that appears in the header of automatically generated columns. For example, you can change the "ProductName" column so that it appears with the "Name" header using the following code:

- Visual Basic

```
Private Sub C1DataGrid1_AutoGeneratingColumn(ByVal sender As
System.Object, ByVal e As
C1.WPF.DataGrid.DataGridAutoGeneratingColumnEventArgs) Handles
C1DataGrid1.AutoGeneratingColumn
    ' Modify the header of the ProductName column.
    If e.Column.Header.ToString() = "ProductName" Then
         e.Header = "Name"
    End If
End Sub
```

- C#

```
private void c1DataGrid1_AutoGeneratingColumn(object sender,
C1.WPF.DataGrid.DataGridAutoGeneratingColumnEventArgs e)
{
    // Modify the header of the ProductName column.
    if (e.Column.Header.ToString() == "ProductName")
        e.Column.Header = "Name";
}
```

Preventing Column Interaction

Using the **AutoGeneratingColumn** event you can change how end users interact with specific generated columns. For example, you can prevent users from moving read-only columns with the following code:

- Visual Basic

```
Private Sub C1DataGrid1_AutoGeneratingColumn(ByVal sender As
System.Object, ByVal e As
C1.WPF.DataGrid.DataGridAutoGeneratingColumnEventArgs) Handles
C1DataGrid1.AutoGeneratingColumn
    ' Modify the header of the ProductName column.
    If e.Column.IsReadOnly = True Then
       e.Column.CanUserMove = False
    End If
End Sub
```

- C#

```
private void c1DataGrid1_AutoGeneratingColumn(object sender,
C1.WPF.DataGrid.DataGridAutoGeneratingColumnEventArgs e)
{
    // Modify the header of the ProductName column.
    if (e.Column.IsReadOnly == true)
        e.Column.CanUserMove = false;
}
```

Creating Custom Columns

ComponentOne DataGrid for WPF supports creating specific behavior columns. For example you can create a Hyperlink column, a GIF column, a Rich Text column, and so on.

By creating a custom column you'll be able to customize the cell content and editing content of all the cells belonging to a column, you can even customize the header presenter of the column.

First, you should add a new class file where the custom column will be written, for example complete the following steps:

1. Navigate to the Solution Explorer, right-click the project name and select **Add | New Item**.
2. In the **Add New Item** dialog box choose **Class** in the list of templates.
3. Name the class, for example "DataGridHyperlinkColumn", and click the **Add** button to add the class to the project.

Once the file is created it must inherit from **DataGridBoundColumn**. Update the class so it appears similar to the following:

- Visual Basic

```
Imports C1.WPF.DataGrid
Public Class DataGridHyperlinkColumn
    Inherits DataGridBoundColumn
End Class
```

- C#

```
using C1.WPF.DataGrid;
public class DataGridHyperlinkColumn : DataGridBoundColumn
{

}
```

Customizing Column Cell Content

In this section you'll find information about changing the UI element shown as the content of cells belonging to a column when the cell is not in editing mode.

It’s important to note that cell content UI elements are recycled by the data grid; that means that this column could potentially use UI elements created by other columns.

To implement custom cell content you'll need to override the following methods:

- **GetCellContentRecyclingKey**: Key used to store the cell content for future reuse in a shared pool. Columns returning the same **RecyclingKey** will be candidates to share the same cell content instances.
- **CreateCellContent**: Creates the visual element that will be used to display the information inside a cell.
- **BindCellContent**: Initializes the cell content presenter. This method must set **cellContent** properties, the **SetBinding** of the corresponding dependency property being "row.DataItem", the source which can be set directly in the binding or in the **DataContext** of the **cellContent**.

- **UnbindCellContent**: This method is called before the cell content is recycled.

In the implementation of a hyperlink column the methods might look similar to the example below. In the following method a different key for this column is returned (the default key is typeof(TextBlock)), That means this column will not share the cell content element with other columns (unless it would be another column which returned the same key, but that's not likely to happen).

- Visual Basic

```
Public Overloads Overrides Function GetCellContentRecyclingKey(ByVal
row As DataGridRow) As Object
    Return (GetType(HyperlinkButton))
End Function
```

- C#

```
public override object GetCellContentRecyclingKey(DataGridRow row)
{
    return typeof(HyperlinkButton);
}
```

The **CreateCellContent** method will be called by the data grid if there is no recycled hyperlink. In this case a new hyperlink will be created which will be used in the cell once the cell that contains the hyperlink is unloaded; the hyperlink will be saved to be used in a future cell:

- Visual Basic

```
Public Overloads Overrides Function CreateCellContent(ByVal row As
DataGridRow) As FrameworkElement
    Return New HyperlinkButton()
End Function
```

- C#

```
public override FrameworkElement CreateCellContent(DataGridRow row)
{
    return new HyperlinkButton();
}
```

After the hyperlink is created or a recycled one is taken, the **BindCellContent** method will be called by the data grid passing the hyperlink as a parameter. In this method you should set the properties of the hyperlink to bind it to the data of the cell:

- Visual Basic

```
Public Overloads Overrides Sub BindCellContent(ByVal cellContent As
FrameworkElement, ByVal row As DataGridRow)
    Dim hyperlink = DirectCast(cellContent, HyperlinkButton)
    If Binding IsNot Nothing Then
        Dim newBinding As Binding = CopyBinding(Binding)
        newBinding.Source = row.DataItem
        hyperlink.SetBinding(HyperlinkButton.NavigateUriProperty,
newBinding)
    End If
    hyperlink.HorizontalAlignment = HorizontalAlignment
    hyperlink.VerticalAlignment = VerticalAlignment
End Sub
```

- C#

```
public override void BindCellContent(FrameworkElement cellContent,
DataGridRow row)
{
    var hyperlink = (HyperlinkButton)cellContent;
    if (Binding != null)
    {
        Binding newBinding = CopyBinding(Binding);
        newBinding.Source = row.DataItem;
        hyperlink.SetBinding(HyperlinkButton.NavigateUriProperty,
newBinding);
    }
    hyperlink.HorizontalAlignment = HorizontalAlignment;
    hyperlink.VerticalAlignment = VerticalAlignment;
}
```

Note that you can also set the data item as the data context of the hyperlink instead of setting it in the **Source** property of the binding. For example:

- Visual Basic

  ```
  Hyperlink.DataContext = row.DataItem
  ```

- C#

  ```
  Hyperlink.DataContext = row.DataItem;
  ```

Although you will end up with the same result, this technique is does not perform as well as setting the binding source property directly.

Adding Properties to a Custom Column

You may want to add properties to a column in order to set a specific behavior. Continuing with the hyperlink column created in the previous topics, in this topic you'll add a property called **TargetName**. This property allows the user to specify the name of the target window or frame where the page will open.

Complete the following steps:

1. Add the following code to create the **TargetName** property:

   - Visual Basic

     ```
     Private _TargetName As String
     Public Property TargetName() As String
         Get
             Return _TargetName
         End Get
         Set(ByVal value As String)
             _TargetName = value
         End Set
     End Property
     ```

   - C#

     ```
     public string TargetName { get; set; }
     ```

2. Once the property is created you'll propagate this to the hyperlink in the **BindCellContent** method:

   - Visual Basic

     ```
     Public Overloads Overrides Sub BindCellContent(ByVal cellContent As
     FrameworkElement, ByVal row As DataGridRow)
         Dim hyperlink = DirectCast(cellContent, HyperlinkButton)
         If Binding IsNot Nothing Then
             Dim newBinding As Binding = CopyBinding(Binding)
             newBinding.Source = row.DataItem
             hyperlink.SetBinding(HyperlinkButton.NavigateUriProperty,
     newBinding)
         End If
         hyperlink.HorizontalAlignment = HorizontalAlignment
         hyperlink.VerticalAlignment = VerticalAlignment
         hyperlink.TargetName = TargetName
     End Sub
     ```

- C#
```
public override void BindCellContent(FrameworkElement cellContent,
DataGridRow row)
{
    var hyperlink = (HyperlinkButton)cellContent;
    if (Binding != null)
    {
        Binding newBinding = CopyBinding(Binding);
        newBinding.Source = row.DataItem;
        hyperlink.SetBinding(HyperlinkButton.NavigateUriProperty,
newBinding);
    }
    hyperlink.HorizontalAlignment = HorizontalAlignment;
    hyperlink.VerticalAlignment = VerticalAlignment;
    hyperlink.TargetName = TargetName;
}
```

Tips

You may find the following tips helpful when adding properties to a custom column:

- Provide a constructor that takes **PropertyInfo** as parameter calling base(property) in order to automatically set the **Binding**, **SortMemberPath**, **FilterMemberPath** and **Header** properties as well as properties set using custom attributes. Currently supported attributes include: **DisplayAttribute** (**AutoGenerateFilter**, **Name**, **GroupName**, **Order**), **DisplayFormatAttribute**, and **EditableAttribute**.

```
public DataGridHyperlinkColumn(PropertyInfo property) : base(property)
```

- You can set a converter in the binding to help you to manage scenarios where you need to use a column bound to property source that is not the same type. Suppose you want to bind a numeric column against a string property, this scenario will work correctly if you set a converter type which converts the string to a double.

Creating Custom Rows

You may be able to solve several scenarios by creating custom rows like a new row, group row, filter row, summary row, totals row, template row, and so on. Some of these rows are implemented internally and others are provided as samples.

When creating a custom row you'll be able to change the following parts:

- Cells content
- Row presenter
- Row header presenter

See Customizing Row Cell Content (page 32) for more details.

Customizing Row Cell Content

This topic explains how to customize cell content. For example, suppose you wanted to build a filter row. You could create a grid where the first row has a **TextBox** in each cell and when you type on it the grid is filtered by the typed text as in the following image:

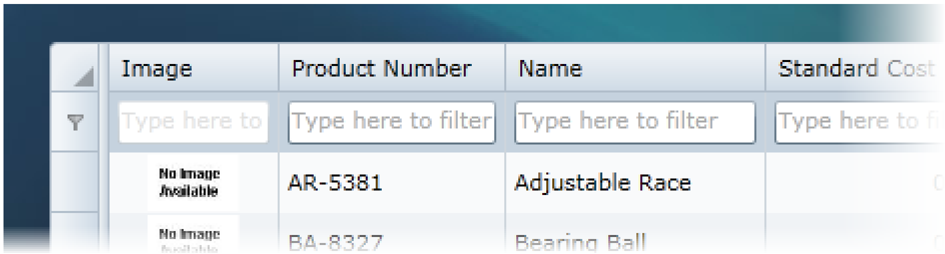


Image	Product Number	Name	Standard Cost
Type here to	Type here to filter	Type here to filter	Type here to fi
No Image Available	AR-5381	Adjustable Race	
No Image	BA-8327	Bearing Ball	

Adding a Class File

You would need to add a new class file where the custom row will be written. For example, complete the following steps to add a new class file:

1. Navigate to the Solution Explorer, right-click the project name and select **Add | New Item**.
2. In the **Add New Item** dialog box choose **Class** in the list of available templates.
3. Name the class, for example "DataGridFilterRow", and click the **Add** button to add the class to the project.
4. Update the class so it appears similar to the following:
   - Visual Basic
     ```
     Imports C1.WPF.DataGrid
     Public Class DataGridFilterRow
        Inherits DataGridRow
     End Class
     ```
   - C#
     ```
     using C1.WPF.DataGrid;
     public class DataGridFilterRow : DataGridRow
     {

     }
     ```

   This will update the class to inherit from **DataGridRow**. Once the file is created it must inherit from **DataGridRow**.

Once you've added the class, you can use it to implement filtering in the grid.

Overriding Methods

The methods you would need to override to specify the cell content of custom row are very similar to those exposed in custom columns. To implement custom cell content you'd need to override the following methods:

- **HasCellPresenter**: Determines whether a cell should exist for this row and the specified column.
- **GetCellContentRecyclingKey**: Key used to store the cell content for future reuse in a shared pool. Rows returning the same **RecyclingKey** can share the same cell content instances.
- **CreateCellContent**: Creates a visual element that will be used to display information inside a cell in this column.
- **BindCellContent**: Initializes the cell content presenter.
- **UnbindCellContent**: This method is called before the cell content is recycled.

In the filter row the **HasCellPresenter** method will return always true, because all columns will have a corresponding cell. In other scenarios like a summary row, only the columns where there is an aggregate function will have a cell.

The **GetCellContentRecyclingKey** method will return typeof(TextBox), which allows recycling the text boxes, and the **CreateCellContent** will create a new instance of it. Add the following code to

- Visual Basic

```
Protected Overrides Function GetCellContentRecyclingKey(column As
DataGridColumn) As Object
      Return GetType(TextBox)
End Function

Protected Overrides Function CreateCellContent(column As DataGridColumn)
As FrameworkElement
      Return New TextBox()
End Function
```

- C#

```
protected override object GetCellContentRecyclingKey(DataGridColumn
column)
{
    return typeof(TextBox);
}

protected override FrameworkElement CreateCellContent(DataGridColumn
column)
{
    return new TextBox();
}
```

Implementing Filtering

In the previous steps you added a **TextBox** in each cell, but these controls currently do not do anything; to implement filtering complete the following steps:

1. Add the following code to the **BindCellContent** method:

   - Visual Basic

```
Protected Overrides Sub BindCellContent(cellContent As
FrameworkElement, column As DataGridColumn)
   Dim filterTextBox = DirectCast(cellContent, TextBox)
   'If the column doesn't have a FilterMemberPath specified
   'it won't allow entering text in the TextBox;
   If String.IsNullOrEmpty(column.FilterMemberPath) Then
         filterTextBox.IsEnabled = False
         filterTextBox.Text = "Not available"
   Else
         filterTextBox.Text = ""
         filterTextBox.IsEnabled = True
   End If
   ' Handle TextChanged to apply the filter to the column.
   filterTextBox.TextChanged += New EventHandler(Of
TextChangedEventArgs)(filterTextBox_TextChanged)
End Sub
```

   - C#

```
protected override void BindCellContent(FrameworkElement cellContent,
DataGridColumn column)
{
    var filterTextBox = (TextBox)cellContent;
    //If the column doesn't have a FilterMemberPath specified
    //it won't allow entering text in the TextBox;
    if (string.IsNullOrEmpty(column.FilterMemberPath))
    {
        filterTextBox.IsEnabled = false;
        filterTextBox.Text = "Not available";
    }
    else
    {
        filterTextBox.Text = "";
        filterTextBox.IsEnabled = true;
    }
    // Handle TextChanged to apply the filter to the column.
    filterTextBox.TextChanged += new
EventHandler<TextChangedEventArgs>(filterTextBox_TextChanged);
}
```

2. In **UnbindCellContent** you must remove the text changed handler to avoid leaking memory:

   - Visual Basic

```
Protected Overrides Sub UnbindCellContent(cellContent As
FrameworkElement, column As DataGridColumn)
   Dim filterTextBox = DirectCast(cellContent, C1SearchBox)
   filterTextBox.TextChanged -= New EventHandler(Of
TextChangedEventArgs)(filterTextBox_TextChanged)
End Sub
```

   - C#

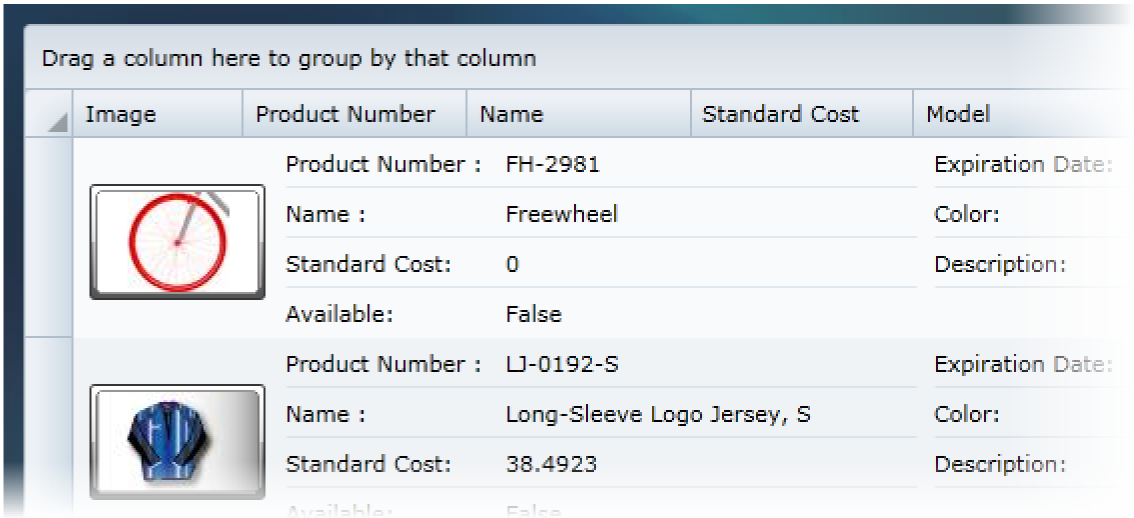
```
protected override void UnbindCellContent(FrameworkElement cellContent,
DataGridColumn column)
{
    var filterTextBox = (C1SearchBox)cellContent;
    filterTextBox.TextChanged -= new
EventHandler<TextChangedEventArgs>(filterTextBox_TextChanged);
}
```

Adding a Custom Row to the Data Grid

You can replace rows the data grid uses to show the data of each data item or group with custom rows, or you can add custom rows on top or bottom of data item rows.

Replacing Data Item Row

In order to replace the rows generated by the data grid you must add a handler to the **CreatingRow** event. For example, in the following image the rows were replaced with template rows:



Drag a column here to group by that column

Image	Product Number	Name	Standard Cost	Model

Product Number : FH-2981

Name : Freewheel

Standard Cost: 0

Available: False

Expiration Date:

Color:

Description:

Product Number : LJ-0192-S

Name : Long-Sleeve Logo Jersey, S

Standard Cost: 38.4923

Available: False

Expiration Date:

Color:

Description:

The following code replaces the default row with a template row:

- Visual Basic

```
Private Sub C1DataGrid_CreatingRow(sender As Object, e As
DataGridCreatingRowEventArgs)
      'Check if it's an item row (it could be a group row too).
      If e.Type = DataGridRowType.Item Then
            e.Row = New DataGridTemplateRow() With { _
                  .RowTemplate = DirectCast(Resources("TemplateRow"),
DataTemplate) _
            }
      End If
End Sub
```

- C#

```
private void C1DataGrid_CreatingRow(object sender,
DataGridCreatingRowEventArgs e)
{
    //Check if it's an item row (it could be a group row too).
    if (e.Type == DataGridRowType.Item)
    {
        e.Row = new DataGridTemplateRow()
        {
            RowTemplate = (DataTemplate)Resources["TemplateRow"]
        };
    }
}
```

Adding an Extra Row

ComponentOne DataGrid for WPF allows adding one or more rows on top or bottom of data. This functionality is used in the new row, total row, summary row, and filter row scenarios.

For example, in XAML or code:

- XAML

```
<c1:C1DataGrid>
    <c1:C1DataGrid.TopRows>
        <local:DataGridFilterRow />
    </c1:C1DataGrid.TopRows>
    <c1:C1DataGrid.BottomRows>
        <local:DataGridFilterRow/>
    </c1:C1DataGrid.BottomRows>
</c1:C1DataGrid>
```

- Visual Basic

```
grid.Rows.TopRows.Add(New DataGridFilterRow())
```

- C#

```
    grid.Rows.TopRows.Add(new DataGridFilterRow());
```

Adding Row Details

Each grid row in **ComponentOne DataGrid for WPF** can be expanded to display a row details section. This row details section can display more details information about a specific row's content. The row details section is defined by a **DataTemplate**, **RowDetailsTemplate**, that specifies the appearance of the section and the data to be displayed. For an example, see the RowDetailsTemplate (page 51) topic.

Using the **RowDetailsVisibilityMode** property the row details section can be displayed for selected rows, displayed for all rows, or it can be collapsed. Setting Row Details Visibility (page 47) for more information.

Localizing the Application

You can localize (translate) end user visible strings in **ComponentOne DataGrid for WPF**. Localization in **DataGrid for WPF** is based on the same approach as the standard localization of .NET Windows forms.

To localize your application, you will need to complete the following steps:

1. Add resource files for each culture that you wish to support.
2. Update your project file's supported cultures.
3. And, depending on your project, set the current culture.

Adding Resource Files

As with Windows Forms, you can create a set of resource files for the **DataGrid for WPF** assembly. You can create separate resource files, with the extension .resx, for each required culture. When the application runs you can switch between those resources and between languages. Note that all parts of your application using components from a **DataGrid for WPF** DLL must use the same localization resource.

Localization Conventions

To localize the grid you would need to set up resource files for each localized culture. The following conventions are recommended when creating .resx resource files:

- All .resx files should be placed in the **Resources** subfolder of your project.
- Files should be named as follows:

  XXX.YYY.resx, where:

  - XXX is the name of the ComponentOne assembly.
  - YYY is the culture code of the resource. If your translation is only for the invariant culture, the .resx file does not need to contain a culture suffix.

  For example:

  - C1.WPF.DataGrid.de.resx – German (de) resource for the C1.WPF.DataGrid assembly.
  - C1.WPF.DataGrid.resx – Invariant culture resource for the C1.WPF.DataGrid assembly.

Localization Strings

The following table lists strings that can be added to an .resx file to localize your application:

String	Default Value	Description
AddNewRow	Click here to add a new row	Text that appears in the add new row.
CheckBoxFilter_Checked	Checked :	Text that appears in the filter for check box columns to indicate if the column should be filtered for checked or unchecked items.
ComboBoxFilter_SelectAll	Select All	Text that appears in the filter for check box columns to select all items.
DateTimeFilter_End	End	Text that appears in the filter for date time columns for the end of the date time range.
DateTimeFilter_Start	Start	Text that appears in the filter for date time columns for the start of the date time range.
EmptyGroupPanel	Drag a column here to group by that column	Text that appears in the grouping area of the grid when no columns are grouped.
Filter_Clear	Clear	Text that appears in the filter bar to clear the filter condition.
Filter_Filter	Filter	Text that appears in the filter bar to add a filter condition.
NumericFilter_And	And	Text that appears in the filter bar for numeric columns to indicate multiple filter conditions.
NumericFilter_Equals	Equals	Text that appears in the filter bar for numeric columns to indicate the filter condition should apply to exact matches only.
NumericFilter_GraterOrEquals	Greater/Equals	Text that appears in the filter bar for numeric columns to indicate the filter condition should apply to items with higher values than the condition value or exact matches only.
NumericFilter_Greater	Greater	Text that appears in the filter bar for numeric columns to indicate the filter condition should apply to items with higher values than the condition value.
NumericFilter_Less	Less	Text that appears in the filter bar for numeric columns to indicate the filter condition should apply to items with lower values than the condition value.
NumericFilter_LessOrEquals	Less/Equals	Text that appears in the filter bar for numeric columns to indicate the filter condition should apply to items with lower values than the condition value or exact matches only.
NumericFilter_NotEquals	Not Equals	Text that appears in the filter bar for numeric columns to indicate the filter condition should apply to items that are not an exact match.
NumericFilter_Or	Or	Text that appears in the filter bar for numeric columns to indicate multiple filter conditions.
TextFilter_Contains	Contains	Text that appears in the filter for text columns to indicate if the filter condition should apply to items that contain the value of the condition.
TextFilter_StartsWith	Starts With	Text that appears in the filter for text columns to indicate if the filter condition should apply to items that start with the value of the condition.
TextFilter_Equals	Equals	Text that appears in the filter bar for text columns to indicate the filter condition should apply to exact matches only.

TextFilter_NotEquals	Not Equals	Text that appears in the filter bar for text columns to indicate the filter condition should apply to items that are not an exact match.

Adding Supported Cultures

Once you've created resource files for your application, you will need to set the supported cultures for your project. To do so, complete the following steps:

1. In the Solution Explorer, right-click your project and select **Unload Project**.

   The project will appear grayed out and unavailable.

2. Right click the project again, and select the **Edit ProjectName.csproj** option (or **Edit ProjectName.vbproj**, where *ProjectName* is the name of your project).

3. In the .csproj file, locate the `<SupportedCultures></SupportedCultures>` tags. In between the tags, list the cultures you want to be supported, separating each with a semicolon.

   For example:

   ```
   <SupportedCultures>fr;es;en;it;ru</SupportedCultures>
   ```

   This will support French, Spanish, English, Italian, and Russian.

4. Save and close the .csproj or .vbproj file.

5. In the Solution Explorer, right-click your project and choose **Reload Project** from the content menu.

   The project will be reloaded and will now support the specified cultures.

Setting the Current Culture

The **C1DataGrid** control will use localization files automatically according to the culture selected in the application as long as you haven't moved files to another location or excluded files from the project. By default, the current culture is designated as **System.Threading.Thread.CurrentThread.CurrentUICulture**. If you want to use a culture other than the current culture, you can set the desired culture in your application using the following code:

- Visual Basic

  ```
  Public Sub New()
        ' Set desired culture, for example here the French (France) locale.
        System.Threading.Thread.CurrentThread.CurrentUICulture = New
  System.Globalization.CultureInfo("fr-FR")
        ' InitializeComponent() call.
        ' Add any initialization after the InitializeComponent() call.
        InitializeComponent()
  End Sub
  ```

- C#

  ```
  public MainPage()
  {
      // Set desired culture, for example here the French (France) locale.
      System.Threading.Thread.CurrentThread.CurrentUICulture = new
  System.Globalization.CultureInfo("fr-FR");
      // InitializeComponent() call.
      InitializeComponent();
      // Add any initialization after the InitializeComponent() call.
  }
  ```

Enabling or Disabling End User Interaction

You can customize how much control end users have over the grid at run time. For example you can enable grouping, and prevent actions such as filtering columns and resizing rows. The following table lists properties that you can use to customize how users interact with the grid:

Property	**Description**
CanUserAddRows	Determines if users can add rows at run time. **True** by default.
CanUserEditRows	Determines if users can edit rows at run time. **True** by default.
CanUserFilter	Determines if users can filter columns at run time. If **True**, the filter bar will be visible on columns. **True** by default.
CanUserGroup	Determines if users can group rows at run time. If **True** the grouping area of the grid will be visible. **False** by default.
CanUserRemoveRows	Determines if users can remove rows at run time by pressing the DELETE key. **True** by default.
CanUserReorderColumns	Determines if users can reorder columns at run time by using a drag-and-drop operation. **True** by default.
CanUserResizeColumns	Determines if users can resize columns at run time. **True** by default.
CanUserResizeRows	Determines if users can resize rows at run time. **False** by default.
CanUserSort	Determines if users can sort columns at run time by clicking on a column's header. **True** by default.
CanUserToggleDetails	Determines if users can toggle the row details section's visibility. **True** by default.
CanUserFreezeColumns	Determines if users can change the number of frozen columns by dragging the freezing separator at run time. **None** by default.

In each column you can customize the following properties:

Property	**Description**
CanUserMove	Determines if users can reorder this column at run time. **True** by default.
CanUserResize	Determines if users can resize this column at run time. **True** by default.
CanUserFilter	Determines if users can filter this column at run time. If **True**, the filter bar will be visible on this column. **True** by default.
CanUserSort	Determines if users can sort this column at run time. **True** by default.

Note: The properties set in the grid take precedence over those set in columns.

Setting Selection Mode

You can set the grid's selection mode behavior by setting the **SelectionMode** property. You can change how users interact with the grid, but setting the **SelectionMode** property to one of the following values:

Option	Description
None	The user can not select any item.
SingleCell	The user can select only one cell at a time.
SingleRow	The user can select only one row at a time.
SingleColumn	The user can select only one column at a time.
SingleRange	The user can select only one cells range at a time. (A range is the rectangle delimited by two cells)
MultiRow (Default)	The user can select multiple rows while holding down the corresponding modifier key.
MultiColumn	The user can select multiple columns while holding down the corresponding modifier key.
MultiRange	The user can select multiple cells ranges while holding down the corresponding modifier key.

For more information about modifier keys and the **MultiRow** option, see the Multiple Row Selection (page 54) topic.

Locking the Grid

By default users can interact and edit the grid and columns in the grid. If you choose, you can set the grid or specific columns in the grid to not be editable with the **IsReadOnly** property.

In XAML

To lock the grid from being edited, add `IsReadOnly="True"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid x:Name="c1DataGrid1" IsReadOnly="True">
```

In Code

To lock the grid from editing, set the **IsReadOnly** property to **True**. For example:

- Visual Basic

```
Me.C1DataGrid1.IsReadOnly = True
```

- C#

```
this.c1DataGrid1.IsReadOnly = true;
```

Deferred and Real Time Scrolling

ComponentOne DataGrid for WPF supports both real time and deferred scrolling. By default, real time scrolling is used and as a user moves the thumb button or clicks the scroll button the grid scrolls. In deferred scrolling, the grid is not scrolled until the user releases the scrollbar thumb; the grid does not move as the scrollbar thumb is moved. You might want to implement deferred scrolling in your application if the grid contains a large amount of data or to optimize scrolling.

You can determine how the grid is scrolled by setting the ScrollMode property. You can set the ScrollMode property to a C1DataGridScrollMode enumeration option, either **RealTime** (default) or **Deferred**. The example below set the grid to deferred scrolling mode.

In XAML

To set the grid to deferred scrolling mode, add `ScrollMode="Deferred"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid x:Name="c1DataGrid1" ScrollMode="Deferred">
```

In Code

To set the grid to deferred scrolling mode, set the ScrollMode property to **Deferred**. For example:

- Visual Basic

  ```
  Me.C1DataGrid1.ScrollMode = C1DataGridScrollMode.Deferred
  ```

- C#

  ```
  this.c1DataGrid1.ScrollMode = C1DataGridScrollMode.Deferred;
  ```

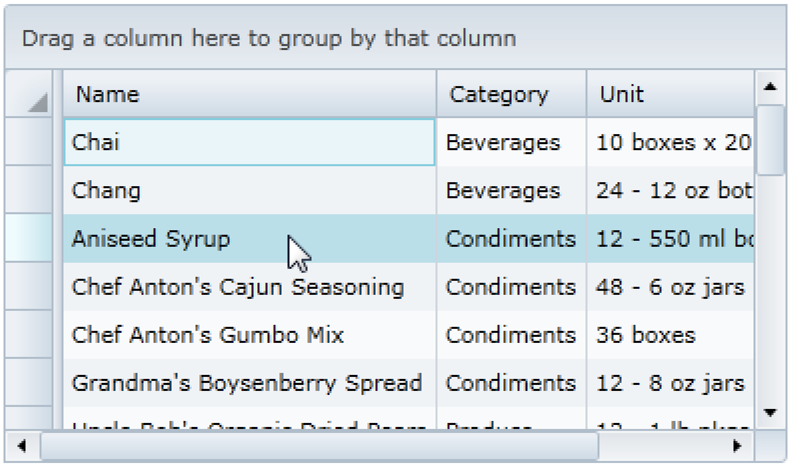
DataGrid for WPF's Appearance

The **C1DataGrid** control supports common table formatting options, such as alternating row backgrounds and the ability to show or hide headers, grid lines, and scroll bars. Additionally, the control provides several brush, style and template properties that you can use to completely change the appearance of the control and its rows, columns, headers, and cells.

Note that **ComponentOne DataGrid for WPF** uses ClearStyle technology for styling. For details, see C1DataGrid ClearStyle (page 48).

C1DataGrid Themes

ComponentOne DataGrid for WPF incorporates several themes that allow you to customize the appearance of your grid. When you first add a **C1DataGrid** control to the page, it appears similar to the following image:



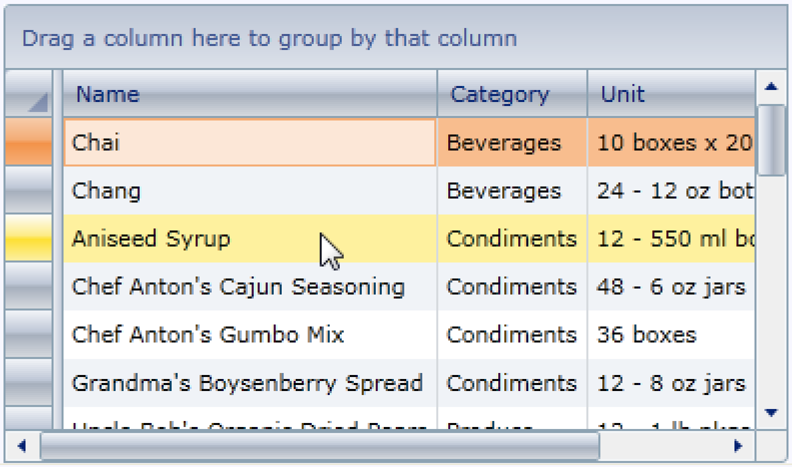
Drag a column here to group by that column

Name	Category	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bot
Aniseed Syrup	Condiments	12 - 550 ml b
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars
Chef Anton's Gumbo Mix	Condiments	36 boxes
Grandma's Boysenberry Spread	Condiments	12 - 8 oz jars

This is the control's default appearance. You can change this appearance by using one of the built-in themes or by creating your own custom theme. All of the built-in themes are based on WPF Toolkit themes. 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:

Theme Name	**Theme Preview**

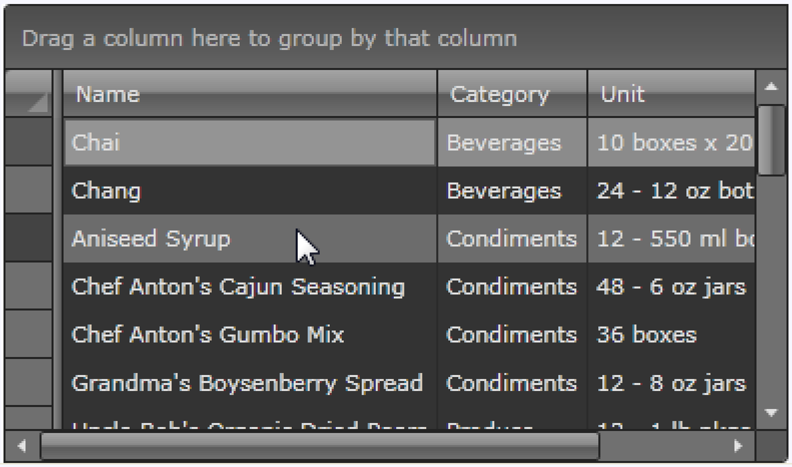
C1ThemeBureauBlack



Drag a column here to group by that column

Name	Category	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bot
Aniseed Syrup	Condiments	12 - 550 ml b
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars
Chef Anton's Gumbo Mix	Condiments	36 boxes
Grandma's Boysenberry Spread	Condiments	12 - 8 oz jars

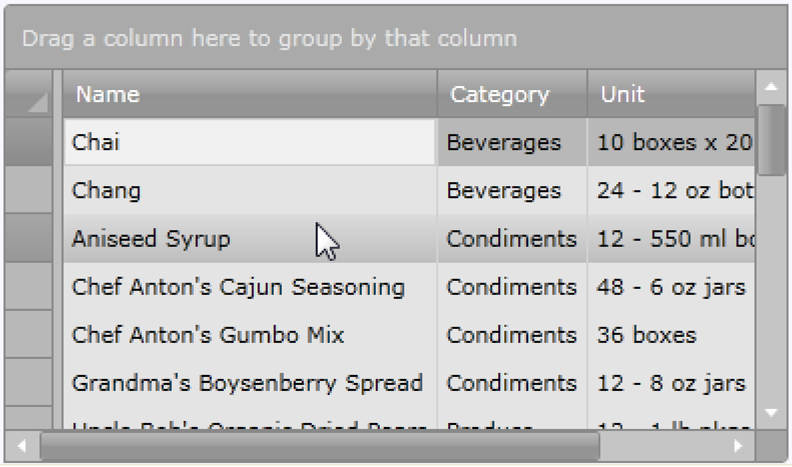
C1ThemeExpressionDark



Drag a column here to group by that column

Name	Category	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bot
Aniseed Syrup	Condiments	12 - 550 ml b
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars
Chef Anton's Gumbo Mix	Condiments	36 boxes
Grandma's Boysenberry Spread	Condiments	12 - 8 oz jars

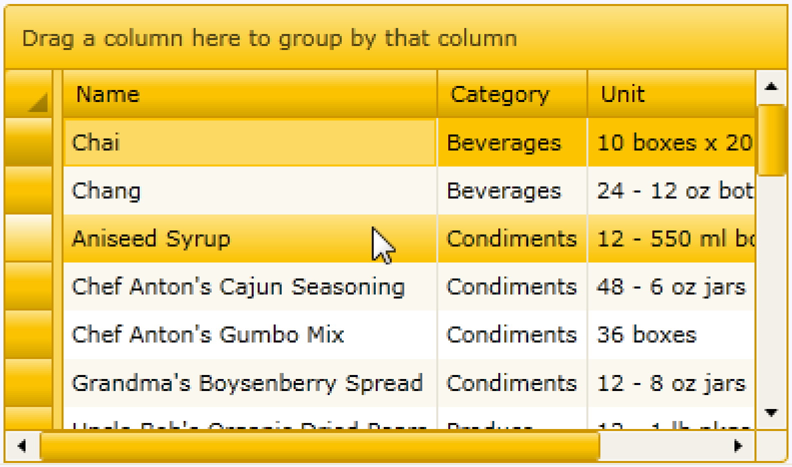
C1ThemeExpressionLight



Drag a column here to group by that column

Name	Category	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bot
Aniseed Syrup	Condiments	12 - 550 ml b
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars
Chef Anton's Gumbo Mix	Condiments	36 boxes
Grandma's Boysenberry Spread	Condiments	12 - 8 oz jars

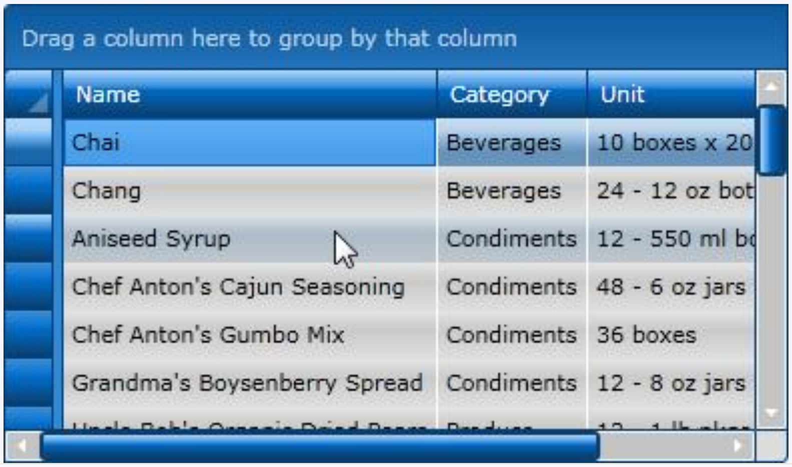
C1ThemeRainierOrange



Drag a column here to group by that column

Name	Category	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bot
Aniseed Syrup	Condiments	12 - 550 ml b
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars
Chef Anton's Gumbo Mix	Condiments	36 boxes
Grandma's Boysenberry Spread	Condiments	12 - 8 oz jars

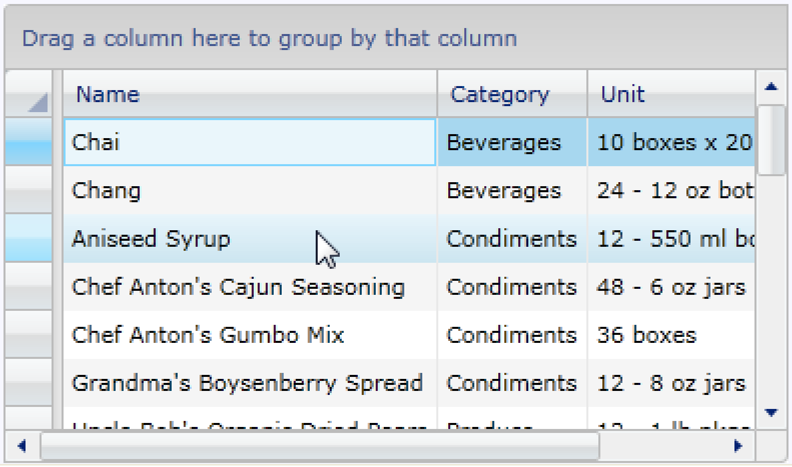
C1ThemeShinyBlue



Drag a column here to group by that column

Name	Category	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bot
Aniseed Syrup	Condiments	12 - 550 ml b
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars
Chef Anton's Gumbo Mix	Condiments	36 boxes
Grandma's Boysenberry Spread	Condiments	12 - 8 oz jars

C1ThemeWhistlerBlue



Drag a column here to group by that column

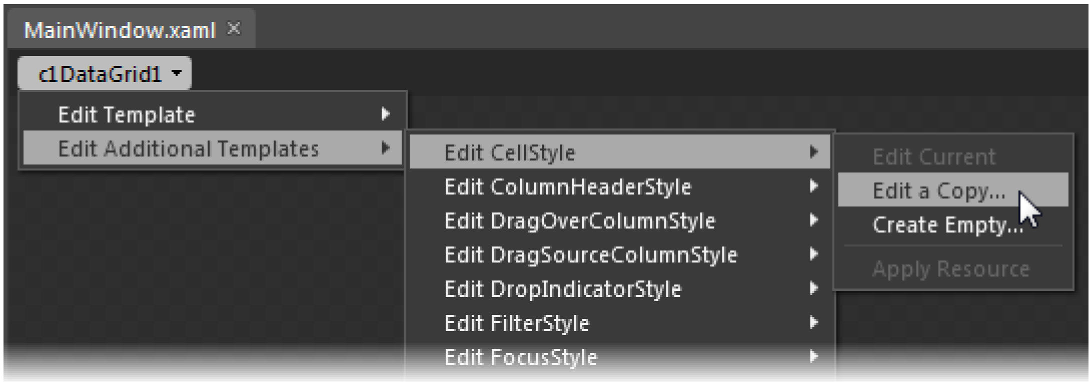
Name	Category	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bot
Aniseed Syrup	Condiments	12 - 550 ml b
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars
Chef Anton's Gumbo Mix	Condiments	36 boxes
Grandma's Boysenberry Spread	Condiments	12 - 8 oz jars

Editing Templates and Styles

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 DataGrid 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. **DataGrid 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.

ComponentOne DataGrid for WPF's **C1DataGrid** control provides several style properties that you can use to completely change the appearance of the control and its rows, columns, headers, and cells. Some of the included styles are described in the table below:

Style	**Description**
CellStyle	Gets or sets the style that is used when rendering the cells.
ColumnHeaderStyle	Gets or sets the style that is used when rendering the column headers.
DragOverColumnStyle	Style applied to a **ContentControl** element used to show the dragged column while it is moved.
DragSourceColumnStyle	Style applied to a **ContentControl** that is placed over the source column when it starts the drag-and-drop operation.
DropIndicatorStyle	Style applied to a **ContentControl** element used to indicate the position where the dragged column will be dropped.
FilterStyle	Gets or sets the style used for the filter control container.
FocusStyle	Sets the style of the internal **Rectangle** used to show the focus on the **C1DataGrid**.
GroupColumnHeaderStyle	Gets or sets the style that is used when rendering the column headers in the group panel.
GroupRowHeaderStyle	Gets of sets the style of the header of the group row.
GroupRowStyle	Gets of sets the style of the group row.
NewRowHeaderStyle	Gets or sets the style that is used when rendering the row header for entering new items.
NewRowStyle	Gets or sets the style that is used when rendering the row for entering new items.
RowHeaderStyle	Gets or sets the style that is used when rendering the row

	headers.
RowStyle	Gets or sets the style that is used when rendering the rows.

Table Formatting Options

The following topics detail table formatting options, including grid headers and placement of table objects.

Setting Row and Column Header Visibility

By default row and column headers are visible in the grid. However, if you choose, you can set one or both of the headers to be hidden by setting the **HeadersVisibility** property. You can set the **HeadersVisibility** property to one of the following options:

Option	Description
None	Neither row nor column headers are visible in the grid.
Column	Only column headers are visible in the grid.
Row	Only row headers are visible in the grid.
All (default)	Both column and row headers are visible in the grid.

Setting Grid Line Visibility

By default vertical and horizontal grid lines are visible in the grid. However, if you choose, you can set one or both sets of grid lines to be hidden by setting the **GridLinesVisibility** property. You can set the **GridLinesVisibility** property to one of the following options:

Option	Description
None	Neither horizontal nor vertical grid lines are visible in the grid.
Horizontal	Only horizontal grid lines are visible in the grid.
Vertical	Only vertical grid lines are visible in the grid.
All (default)	Both horizontal and vertical grid lines are visible in the grid.

Setting New Row Visibility

By default the Add New row is located at the bottom of the grid. However, if you choose, you can change its location by setting the **NewRowVisibility** property. You can set the **NewRowVisibility** property to one of the following options:

Option	Description
Top	The Add New row appears at the top of the grid.
Bottom (default)	The Add New row appears at the bottom of the grid.

Setting Vertical and Horizontal Scrollbar Visibility

By default the grid's horizontal and vertical scrollbars are only visible when the height or width of grid content exceeds the size of the grid. However, if you choose, you can set the scrollbars to be always or never visible, and even disable them altogether, by setting the **VerticalScrollbarVisibility** and **HorizontalScrollbarVisibility** properties. You can set the **VerticalScrollbarVisibility** and **HorizontalScrollbarVisibility** properties to one of the following options:

Option	Description
Disabled	The chosen scrollbar is disabled.
Auto (default)	The chosen scrollbar appears only when the content of the grid is exceeds the grid window.
Hidden	The chosen scrollbar appears to be hidden.
Visible	The chosen scrollbar is always visible.

Setting Row Details Visibility

By default row details are collapsed and not visible. You can use the **RowDetailsVisibilityMode** property to set if and when row details are visible. You can set the **RowDetailsVisibilityMode** property to one of the following options:

Option	Description
VisibleWhenSelected	Row details are only visible when selected.
Visible	Row details are always visible.
Collapsed (default)	Row details appear collapsed and are not visible.

C1DataGrid Brushes

ComponentOne DataGrid for WPF's **C1DataGrid** control provides several brush properties that you can use to completely change the appearance of the control and its rows, columns, headers, and cells. Some of the included brushes are described in the table below:

Brush	Description
Background	Gets or sets the background brush that is used when rendering. (This brush will be applied to all the parts of the data grid)
Foreground	Gets or sets the foreground brush that is used when rendering. (This brush will be applied to all the parts of the data grid)
BorderBrush	Gets or sets the border brush that is used when rendering. (This brush will be applied to some of the parts of the data grid depending on the theme)
SelectedBrush	Gets or sets the selected brush that is used when rendering selected rows and row and column headers, etc.
MouseOverBrush	Gets or sets the mouse over brush that is used when mouse is over rows and row and column headers, etc.
RowBackground	Gets or sets the background brush of a row.
RowForeground	Gets or sets the foreground brush of a row.
AlternatingRowBackground	Gets or sets the background brush of an alternating row.
AlternatingRowForeground	Gets or sets the foreground brush of an alternating row.
HorizontalGridLinesBrush	Gets of sets the brush applied to the horizontal lines.
VerticalGridLinesBrush	Gets of sets the brush applied to the vertical lines.

ComponentOne DataGrid for WPF uses ClearStyle technology for styling. For details, see C1DataGrid ClearStyle (page 48).

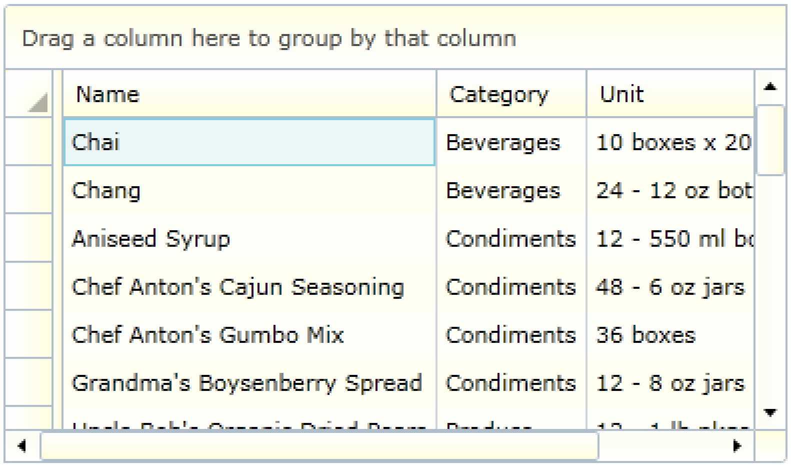
C1DataGrid ClearStyle

DataGrid for WPF supports ComponentOne's new ClearStyle technology that allows you to easily change control colors without having to change control templates. By just setting a few color properties you can quickly style the entire grid.

You can completely change the appearance of the **C1DataGrid** control by simply setting a few properties, such as the **C1DataGrid.Background** property which sets the color scheme of the **C1DataGrid** control. For example, if you set the **Background** property to "#FF663366" so the XAML markup appears similar to the following:

```
<c1:C1DataGrid HorizontalAlignment="Left" Margin="10,10,0,0"
Name="c1DataGrid1" VerticalAlignment="Top" CanUserFreezeColumns="Left"
CanUserGroup="True" Background="#FFFFFFCC"/>
```

The grid will appear similar to the following image:



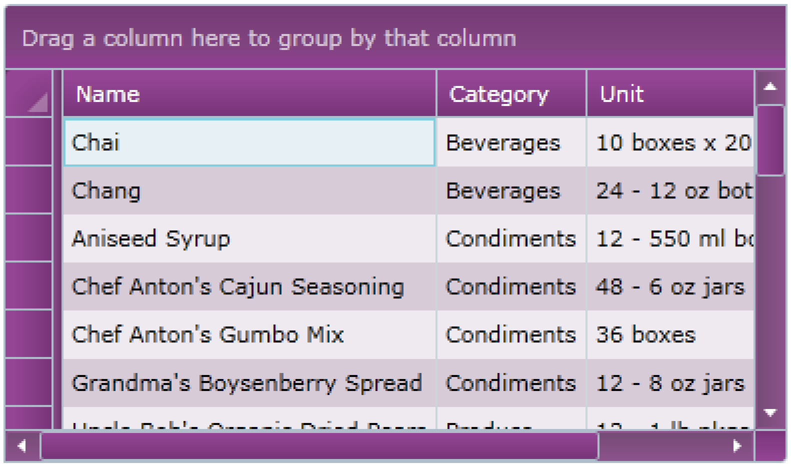
Drag a column here to group by that column

Name	Category	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bot
Aniseed Syrup	Condiments	12 - 550 ml b
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars
Chef Anton's Gumbo Mix	Condiments	36 boxes
Grandma's Boysenberry Spread	Condiments	12 - 8 oz jars

If you set the **Background** property to "#FF663366" and the **Foreground** property to "White", so the XAML markup appears similar to the following:

```
<c1:C1DataGrid HorizontalAlignment="Left" Margin="10,10,0,0"
Name="c1DataGrid1" VerticalAlignment="Top" CanUserFreezeColumns="Left"
CanUserGroup="True" Background="#FF663366" Foreground="White"/>
```

The grid will appear similar to the following image:



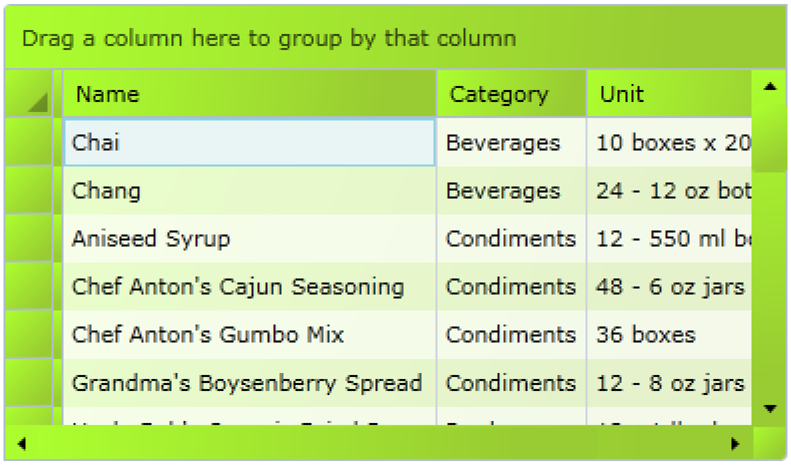
Drag a column here to group by that column

Name	Category	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bot
Aniseed Syrup	Condiments	12 - 550 ml b
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars
Chef Anton's Gumbo Mix	Condiments	36 boxes
Grandma's Boysenberry Spread	Condiments	12 - 8 oz jars

You can even set the **Background** property to a gradient value, for example with the following XAML:

```
<c1:C1DataGrid x:Name="c1DataGrid1" HorizontalAlignment="Left"
Margin="10,10,0,0" VerticalAlignment="Top" CanUserFreezeColumns="Left"
CanUserGroup="True">
    <c1:C1DataGrid.Background>
        <LinearGradientBrush StartPoint="0,0" EndPoint="1,1">
            <GradientStop Color="GreenYellow" Offset="0.0" />
            <GradientStop Color="YellowGreen" Offset="0.85" />
        </LinearGradientBrush>
    </c1:C1DataGrid.Background>
</c1:C1DataGrid>
```

The grid will appear similar to the following image:

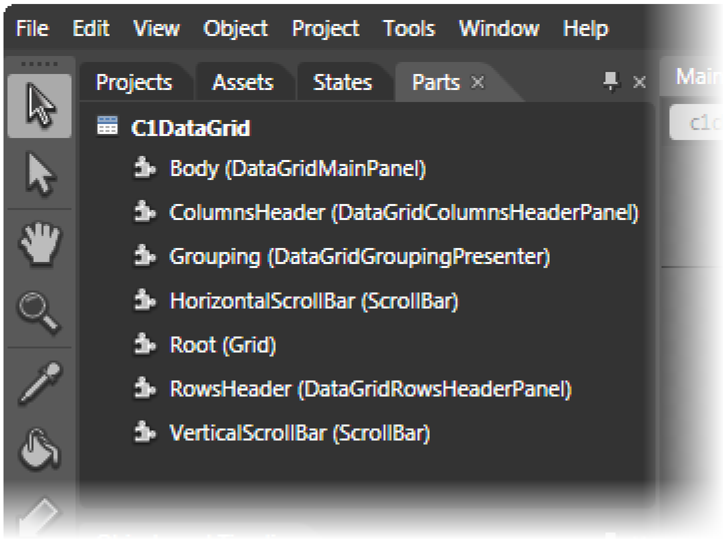


Drag a column here to group by that column

Name	Category	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bot
Aniseed Syrup	Condiments	12 - 550 ml b
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars
Chef Anton's Gumbo Mix	Condiments	36 boxes
Grandma's Boysenberry Spread	Condiments	12 - 8 oz jars

C1DataGrid Template Parts

In Microsoft Expression Blend, you can view and edit template parts by creating a new template (for example, click the **C1DataGrid** control to select it and choose **Object | Edit Template | Edit a Copy**). Once you've created a new template, the parts of the template will appear in the **Parts** window:


File
Edit
View
Object
Project
Tools
Window
Help
Projects
Assets
States
Parts
C1DataGrid
Body (DataGridMainPanel)
ColumnsHeader (DataGridColumnsHeaderPanel)
Grouping (DataGridGroupingPresenter)
HorizontalScrollBar (ScrollBar)
Root (Grid)
RowsHeader (DataGridRowsHeaderPanel)
VerticalScrollBar (ScrollBar)

Note that you may have to select the **ControlTemplate** for its parts to be visible in the **Parts** window.

In the Parts window, you can double-click any element to create that part in the template. Once you have done so, the part will appear in the template and the element's icon in the **Parts** pane will change to indicate selection:

Template parts available in the **C1DataGrid** control include:

Name	**Type**	**Description**
Body	**DataGridMainPanel**	Panel that contains the body of the grid.
ColumnsHeader	DataGridColumnsHeaderPanel	Panel that contains a collection of DataGridColumnsHeaderPanel.
Grouping	DataGridGroupingPresenter	Presenter that displays the grouping panel or another element if there is no columns in the grouping panel.
HorizontalScrollBar	ScrollBar	Represents a control that provides a scroll bar that has a sliding Thumb whose position corresponds to a value.
Root	Grid	Defines a flexible grid area that consists of columns and rows.
RowsHeader	DataGridRowsHeaderPanel	Panel that contains DataGridRowsHeaderPanel.
VerticalScrollBar	ScrollBar	Represents a control that provides a scroll bar that has a sliding Thumb whose position corresponds to a value.

RowDetailsTemplate

The **RowDetailsTemplate** template controls the appearance of the row details area. The row details section appears below a row and can display additional information.

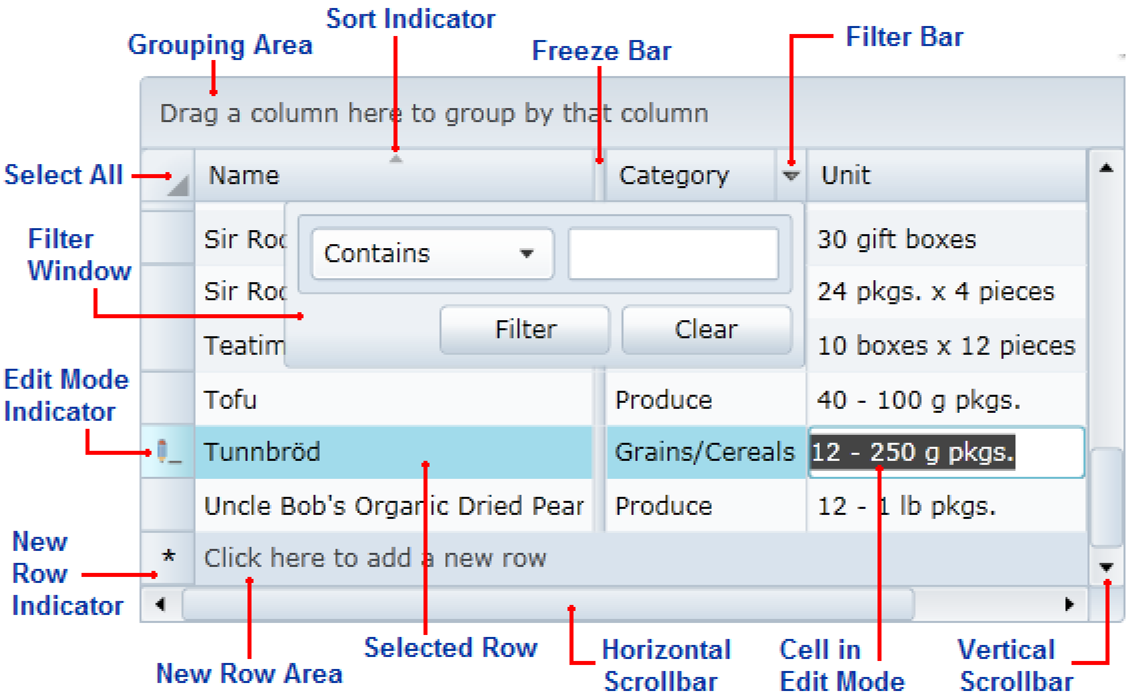
In Expression Blend, you can create an empty template at design time by selecting the **C1DataGrid** control and then clicking **Object | Edit Other Templates | Edit RowDetailsTemplate | Create Empty**.

You can include text, controls, and more in the **RowDetailsTemplate**, including controls bound to data. For example, the following template includes bound and unbound text and check boxes:

```
<c1:C1DataGrid.RowDetailsTemplate>
    <!-- Begin row details section. -->
    <DataTemplate>
        <Border BorderBrush="DarkGray" BorderThickness="1"
Background="Azure">
        <StackPanel Orientation="Horizontal">
            <StackPanel>
                <StackPanel Orientation="Horizontal">
                <!-- Controls are bound to properties. -->
                <TextBlock FontSize="16" Foreground="MidnightBlue"
Text="{Binding Name}" Margin="0,0,10,0" VerticalAlignment="Bottom" />
                <TextBlock FontSize="12" Text="Order Date: "
VerticalAlignment="Bottom"/>
                <TextBlock FontSize="12" Text="    Complete:"
VerticalAlignment="Bottom" />
                <CheckBox IsChecked="{Binding Complete, Mode=TwoWay}"
VerticalAlignment="Center" />
                </StackPanel>
                <TextBlock FontSize="12" Text="Notes: " />
                <TextBox FontSize="12" Text="{Binding Notes,
Mode=TwoWay}" Width="420" TextWrapping="Wrap"/>
            </StackPanel>
        </StackPanel>
        </Border>
    </DataTemplate>
    <!-- End row details section. -->
</c1:C1DataGrid.RowDetailsTemplate>
```

Run-time Interaction

The image below highlights some of the run-time interactions possible in a **ComponentOne DataGrid for WPF C1DataGrid** control:



The following topics detail these run-time features including filtering, sorting, and grouping data.

Keyboard and Mouse Navigation

ComponentOne DataGrid for WPF supports several run-time keyboard and mouse navigation options that provide increased accessibility. The following topics detail some of these end-user interactions.

Keyboard Navigation

The following table lists several keyboard shortcuts that can be used to navigate and manipulate the grid at run time. Note that on Apple computers, end users should use the Command (or Apple) key in place of the CTRL key:

Key Combination	Description
DOWN Arrow	Moves the focus to the cell directly below the current cell. If the focus is in the last row, pressing the DOWN ARROW does nothing.
UP Arrow	Moves the focus to the cell directly above the current cell. If the focus is in the first row, pressing the UP ARROW does nothing.
LEFT ARROW	Moves the focus to the previous cell in the row. If the focus is in the first cell in the row, pressing the LEFT ARROW does nothing.
RIGHT Arrow	Moves the focus to the next cell in the row. If the focus is in the last cell in the row, pressing the RIGHT ARROW does nothing.
HOME	Moves the focus to the first cell in the current row.
END	Moves the focus to the last cell in the current row.
PAGE DOWN	Scrolls the control downward by the number of rows that are displayed. Moves the focus to the last displayed row without changing columns. If the last row is only partially displayed, scrolls the grid to fully display the last row.
PAGE UP	Scrolls the control upward by the number of rows that are displayed.

	Moves focus to the first displayed row without changing columns. If the first row is only partially displayed, scrolls the grid to fully display the first row.
TAB	If the current cell is in edit mode, moves the focus to the next editable cell in the current row. If the focus is already in the last cell of the row, commits any changes that were made and moves the focus to the first editable cell in the next row. If the focus is in the last cell in the control, moves the focus to the next control in the tab order of the parent container. If the current cell is not in edit mode, moves the focus to the next control in the tab order of the parent container.
SHIFT+TAB	If the current cell is in edit mode, moves the focus to the previous editable cell in the current row. If the focus is already in the first cell of the row, commits any changes that were made and moves the focus to the last cell in the previous row. If the focus is in the first cell in the control, moves the focus to the previous control in the tab order of the parent container. If the current cell is not in edit mode, moves the focus to the previous control in the tab order of the parent container.
CTRL+DOWN ARROW	Moves the focus to the last cell in the current column.
CTRL+UP ARROW	Moves the focus to the first cell in the current column.
CTRL+RIGHT ARROW	Moves the focus to the last cell in the current row.
CTRL+LEFT ARROW	Moves the focus to the first cell in the current row.
CTRL+HOME	Moves the focus to the first cell in the control.
CTRL+PAGE DOWN	Same as PAGE DOWN.
CTRL+PAGE UP	Same as PAGE UP.
ENTER	Enter/exit edit mode on a selected cell (if the grid and column's **IsReadOnly** properties are **False**).
F2	Enter edit mode on a selected cell (if the grid and column's **IsReadOnly** properties are **False**). If the focus is on the new row, the grid begins editing the first editable cell of the new row.
ESC	Cancel editing of a cell or new row.
DEL	Delete selected row.
INSERT	Scrolls to the new row and begins editing it.

Mouse Navigation

The following table lists several mouse and keyboard shortcuts that can be used to navigate and manipulate the grid at run time. Note that on Apple computers, end users should use the Command (or Apple) key in place of the CTRL key:

Mouse Action	Description
Click an unselected row	Makes the clicked row the current row.
Click a cell in the current row	Puts the clicked cell into edit mode.
Drag a column header cell	Moves the column so that it can be dropped into a new position (if the **CanUserReorderColumns** property is **True** and the current column's **CanUserReorder** property is **True**).
Drag a column header separator	Resizes the column (if the **CanUserResizeColumns** property is **True** and the **CanUserResize** property is True for the current column).

Click a column header cell	If the property **ColumnHeaderClickAction** is set to **Sort**, when the user clicks the column header it sorts the column (if the **CanUserSortColumns** property is **True** and the **CanUserSort** property is **True** for the current column). Clicking the header of a column that is already sorted will reverse the sort direction of that column. Pressing the CTRL key while clicking multiple column headers will sort by multiple columns in the order clicked. If the property **ColumnHeaderClickAction** is set to **Select** the column will be selected if **SelectionMode** supports column selection.
CTRL+click a row	Modifies a non-contiguous multi-row selection (if **SelectionMode** support multiple rows, cells, or columns).
SHIFT+click a row	Modifies a contiguous multi-row selection (if **SelectionMode** support multiple rows, cells, or columns).

Multiple Row Selection

If the **SelectionMode** property is set to **MultiRow**, the navigation behavior does not change, but navigating with the keyboard and mouse while pressing SHIFT (including CTRL+SHIFT) will modify a multi-row selection. Before navigation starts, the control marks the current row as an anchor row. When you navigate while pressing SHIFT, the selection includes all rows between the anchor row and the current row.

Selection Keys

The following selection keys modify multi-row selection:

- SHIFT+DOWN ARROW
- SHIFT+UP ARROW
- SHIFT+PAGE DOWN
- SHIFT+PAGE UP
- CTRL+SHIFT+DOWN ARROW
- CTRL+SHIFT+UP ARROW
- CTRL+SHIFT+PAGE DOWN
- CTRL+SHIFT+PAGE UP

Mouse Selection

If the **SelectionMode** property is set to **MultiRow**, clicking a row while pressing CTRL or SHIFT will modify a multi-row selection.

When you click a row while pressing SHIFT, the selection includes all rows between the current row and an anchor row located at the position of the current row before the first click. Subsequent clicks while pressing SHIFT changes the current row, but not the anchor row.

If the CTRL key is pressed when navigating, the arrow keys will navigate to the border cells; for example, if you are in the first row and you press CTRL + DOWN you will navigate to the last row, if the SHIFT key is pressed, all the rows will be selected though.

Custom Keyboard Navigation

You can add your own custom navigation to the **C1DataGrid** control. Custom keyboard navigation enables you to control how users interact with the grid. For example, you can prevent users from navigating to read-only columns or cells with null values. In a hierarchical grid, you could set up navigation between parent and child grids. To add custom keyboard navigation you would need to handle the **KeyDown** event and then add code to override the default navigation with your customized navigation.

Adding the KeyDown Event Handler

Complete the following steps to add the **KeyDown** event handler:

1. Switch to Code view and add an event handler for the **KeyDown** event, for example:
    - Visual Basic

    ```
    Private Sub C1DataGrid1_KeyDown(ByVal sender As System.Object, ByVal e
    As System.Windows.Input.KeyEventArgs) Handles C1DataGrid1.KeyDown
        ' Add code here.
    End Sub
    ```

    - C#

    ```
    private void c1DataGrid1_KeyDown(object sender, KeyEventArgs e)
    {
        // Add code here.
    }
    ```

2. Switch to Source view and add the event handler to instances of the **C1DataGrid** control, for example:

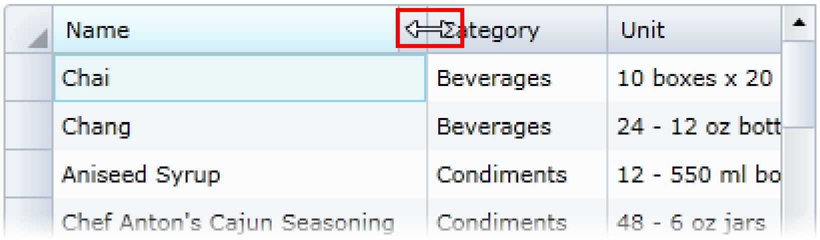
    ```
    <c1:C1DataGrid x:Name="c1DataGrid1" AutoGenerateColumns="True"
    KeyDown="c1DataGrid1_KeyDown"></c1:C1DataGrid>
    ```

You can now add code to the **KeyDown** event handler to customize the default navigation. For an example, you can take a look at the hierarchical grid example (**C1_MDSL_RowDetail**) in the **ControlExplorer** sample.

Resizing Columns and Rows

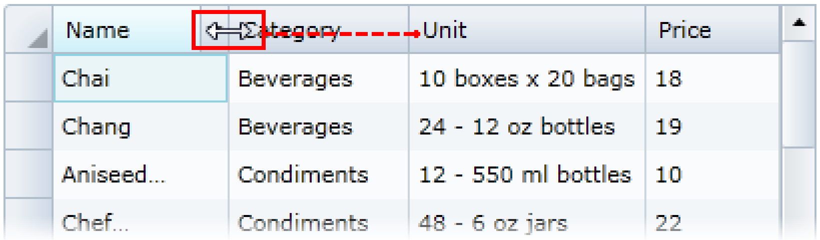
Users can easily resize columns and rows at run time through a drag-and-drop operation. To resize columns at run time, complete the following steps:

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



Name	Category	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bott
Aniseed Syrup	Condiments	12 - 550 ml bo
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars

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



Name	Category	Unit	Price
Chai	Beverages	10 boxes x 20 bags	18
Chang	Beverages	24 - 12 oz bottles	19
Aniseed...	Condiments	12 - 550 ml bottles	10
Chef...	Condiments	48 - 6 oz jars	22

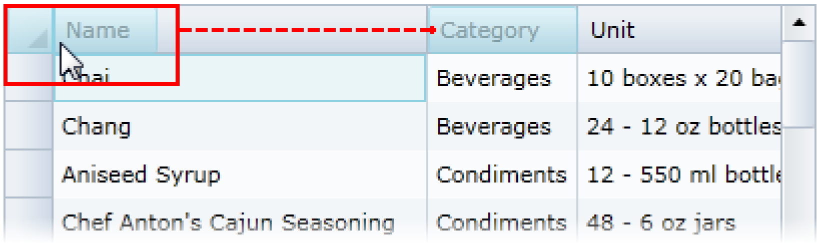
3. Release the mouse to complete the column resize operation.

Resize rows in a similar way by dragging the row indicator column. Note that the **CanUserResizeColumns** and **CanUserResizeRows** properties must be set to **True** (default) for column and row resizing to be possible. See the Disabling Column and Row Resizing (page 69) topic for more details.

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.
2. Drag the column header to where you wish the column to be ordered. Notice that a line will appear if you can place the column in that location:



Name	Category	Unit
Chai	Beverages	10 boxes x 20 ba
Chang	Beverages	24 - 12 oz bottles
Aniseed Syrup	Condiments	12 - 550 ml bottl
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars

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

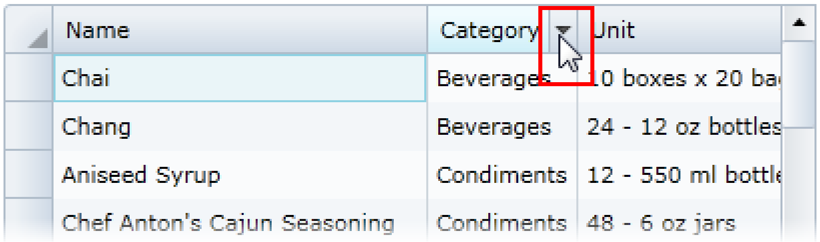
Note that the **CanUserReorderColumns** property must be set to **True** (default) for column reordering to be possible. See the Disabling Column Reordering (page 68) topic for more details.

Filtering Columns

ComponentOne DataGrid 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, complete the following steps:

1. Click the drop-down arrow in a text column's header:



Name	Category	Unit
Chai	Beverages	10 boxes x 20 ba
Chang	Beverages	24 - 12 oz bottles
Aniseed Syrup	Condiments	12 - 550 ml bottl
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars

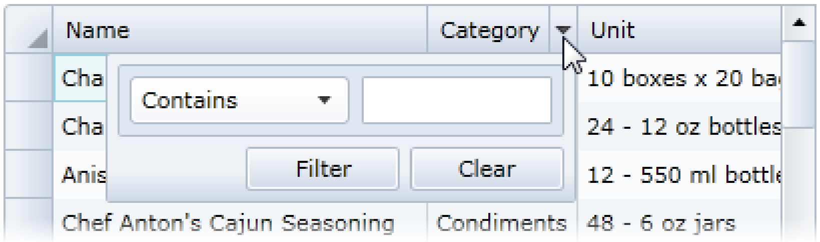
2. Enter the text in the filter text box that you want the column to be filtered by, and click the **Filter** button.

   The column will be sorted.

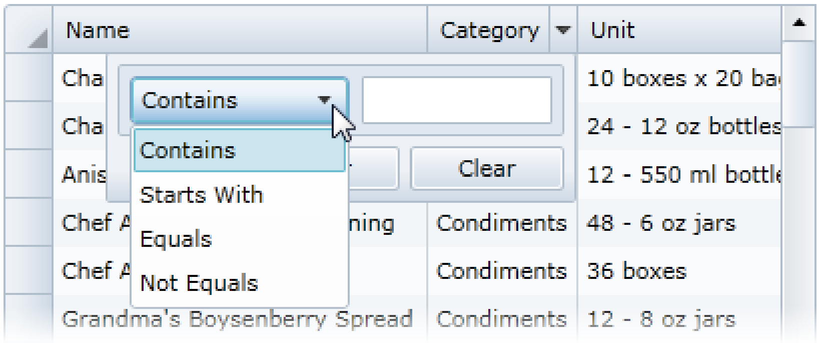
Filter options vary depending on the column type. The following filter options may be included:

- **Text Columns**

  In text columns, the filter bar appears similar to the following:

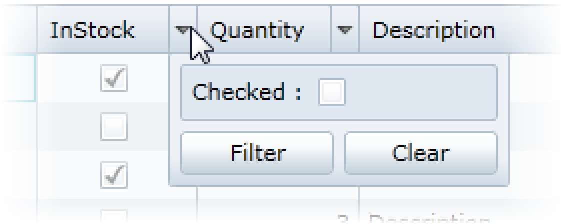


You can filter the column by whether items in the column contain, start, are equivalent to, or are not equivalent to the filter condition:



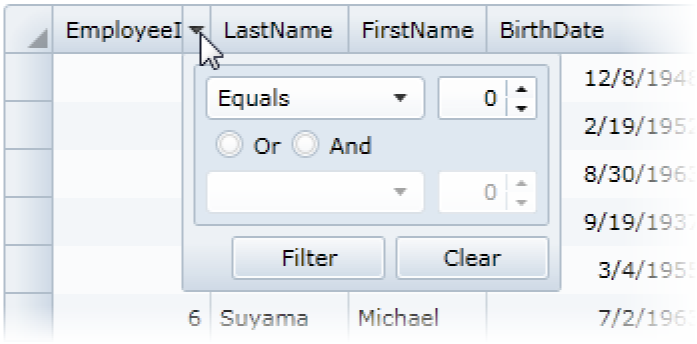
- **Boolean Columns**

  Boolean check box columns can be filtered by whether items in the column are checked or not:

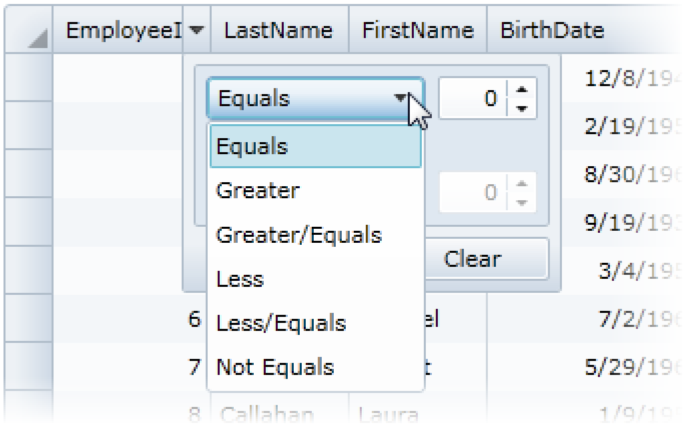


- **Numeric Columns**

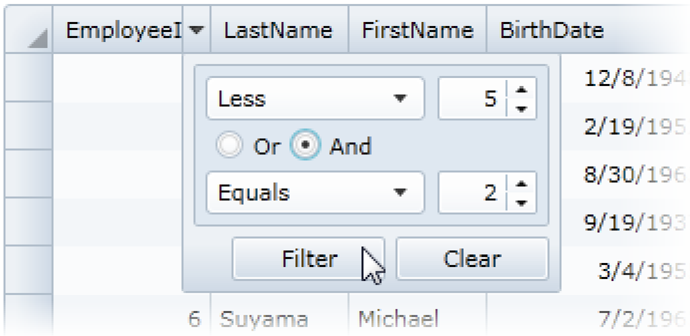
  Numeric columns offer several options for filtering:



You can filter the column by specific condition:



And you can use the **And** and **Or** radio buttons to filter by multiple conditions:

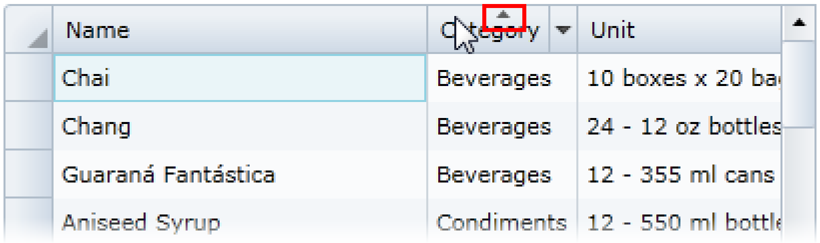


Note that the **CanUserFilter** property must be set to **True** (default) for filtering to be possible.

Sorting Columns

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

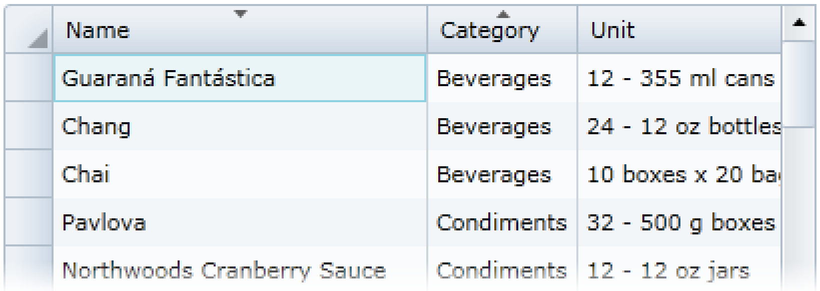
You will notice that the sort glyph, a sort direction indicator, appears when a column is sorted:



Name	Category	Unit
Chai	Beverages	10 boxes x 20 ba
Chang	Beverages	24 - 12 oz bottles
Guaraná Fantástica	Beverages	12 - 355 ml cans
Aniseed Syrup	Condiments	12 - 550 ml bottle

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 *Category* column was first sorted, and then the *Name* column was reverse sorted:

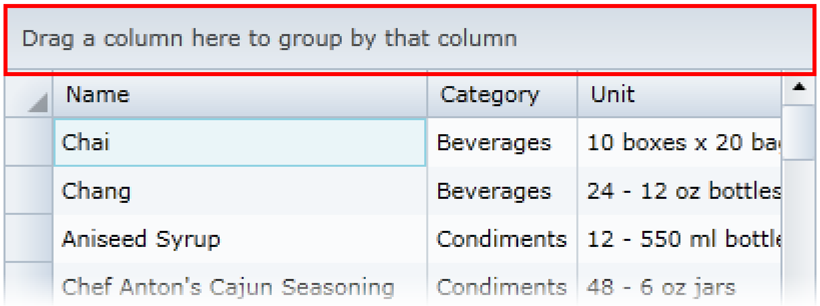


Name	Category	Unit
Guaraná Fantástica	Beverages	12 - 355 ml cans
Chang	Beverages	24 - 12 oz bottles
Chai	Beverages	10 boxes x 20 ba
Pavlova	Condiments	32 - 500 g boxes
Northwoods Cranberry Sauce	Condiments	12 - 12 oz jars

Note that the **CanUserSort** property must be set to **True** (default) for sorting to be possible.

Grouping Columns

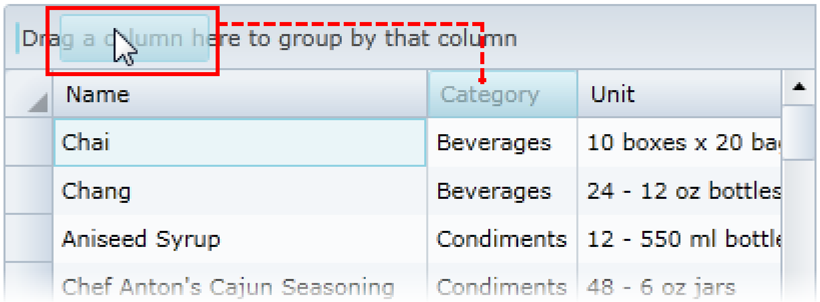
Users can group columns in your grid at run time to better organize information. The grouping area at the top of the grid allows you to easily group columns through a simple drag-and-drop operation:



Drag a column here to group by that column

Name	Category	Unit
Chai	Beverages	10 boxes x 20 ba
Chang	Beverages	24 - 12 oz bottles
Aniseed Syrup	Condiments	12 - 550 ml bottle
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars

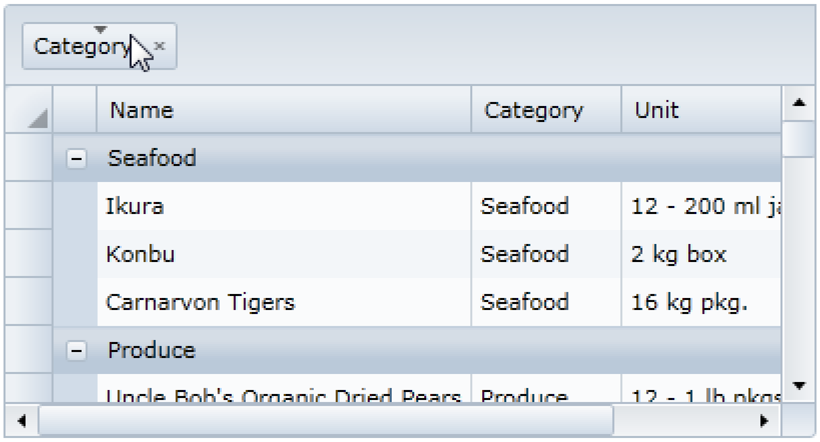
To group a column, drag a column header onto the grouping area:



Drag a column here to group by that column

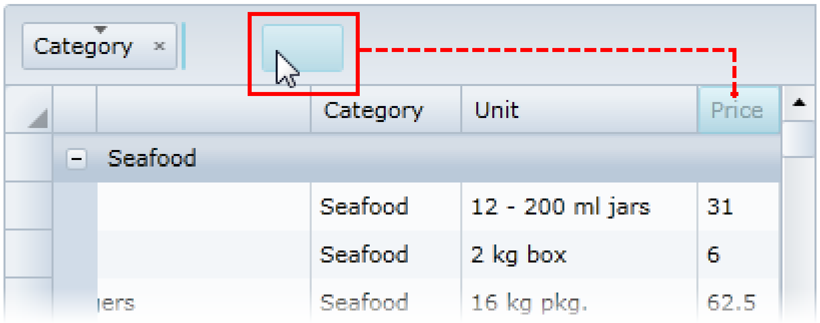
Name	Category	Unit
Chai	Beverages	10 boxes x 20 ba
Chang	Beverages	24 - 12 oz bottles
Aniseed Syrup	Condiments	12 - 550 ml bottle
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars

You can sort the display of grouped items, by clicking the column header in the grouping area. In the following image the grouped column has been reverse sorted:



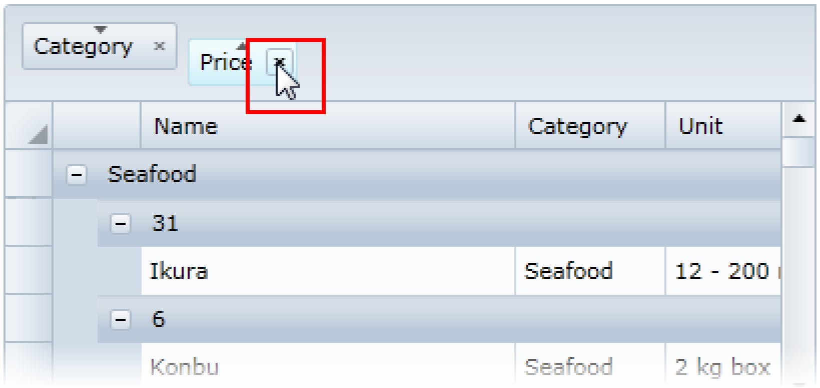
Name	Category	Unit
Seafood		
Ikura	Seafood	12 - 200 ml j
Konbu	Seafood	2 kg box
Carnarvon Tigers	Seafood	16 kg pkg.
Produce		
Uncle Bob's Organic Dried Pears	Produce	12 - 1 lb pkgs

You can group multiple columns by performing a drag-and-drop operation to drag additional columns to the grouping area:



	Category	Unit	Price
Seafood			
	Seafood	12 - 200 ml jars	31
	Seafood	2 kg box	6
ers	Seafood	16 kg pkg.	62.5

To remove the grouping, simply click the **X** button next to a grouped column in the grouping area of the grid:

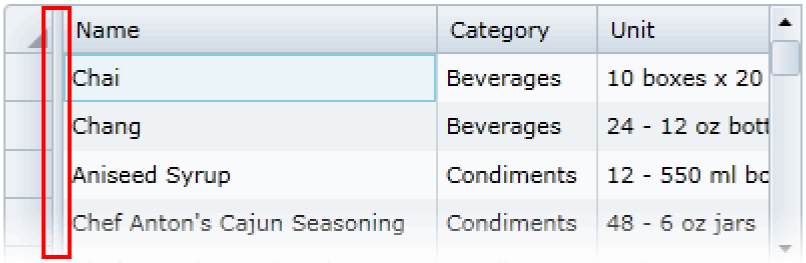


Name	Category	Unit
Seafood		
31		
Ikura	Seafood	12 - 200
6		
Konbu	Seafood	2 kg box

Note that the **CanUserGroup** property must be set to **True** for the grouping area to be visible and grouping to be possible (by default it is set to **False**). For more information, see Enabling Grouping in the Grid (page 67). For more information about showing the grouping area, see the Showing the Grouping Area (page 68) topic.

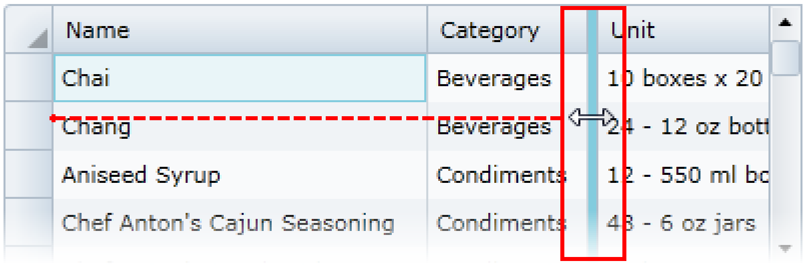
Freezing Columns

Users can freeze columns at run time to prevent them from being scrolled horizontally. This is useful as it keeps specific columns visible when the grid is resized or scrolled. The freeze bar enables users to freeze columns. When visible, the freeze bar appears to the left of the first columns by default:



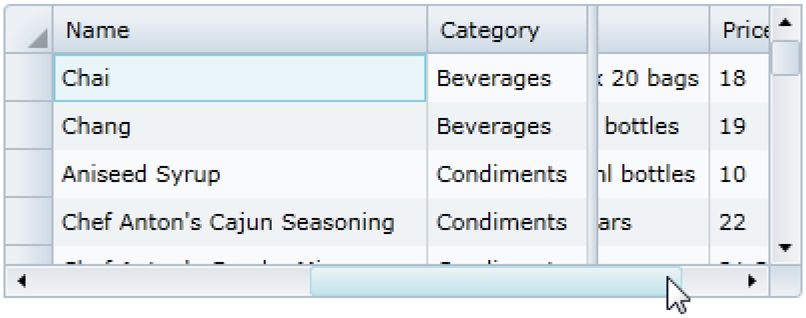
Name	Category	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bott
Aniseed Syrup	Condiments	12 - 550 ml b
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars

To freeze specific columns, move the freeze bar to the right of the column(s) you want to freeze. For example, in the following image the freeze bar was moved to the right of the second columns:



Name	Category	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bott
Aniseed Syrup	Condiments	12 - 550 ml b
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars

Once columns are frozen, they are not scrolled when the grid is scrolled horizontally. For example, in the following image the first two columns are frozen:



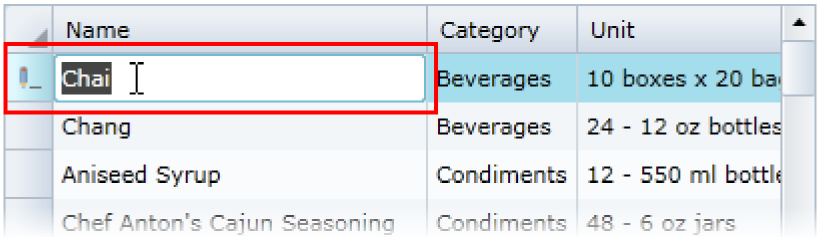
Name	Category		Price
Chai	Beverages	20 bags	18
Chang	Beverages	bottles	19
Aniseed Syrup	Condiments	l bottles	10
Chef Anton's Cajun Seasoning	Condiments	ars	22

Note that the **ShowVerticalFreezingSeparator** property must be set to **Left** (by default **None**) for the freeze bar to be visible and the **CanUserFreezeColumns** property must be set to **Left** (by default **None**) to allow users to freeze columns are run time. See Enabling Column Freezing (page 71) for an example.

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:

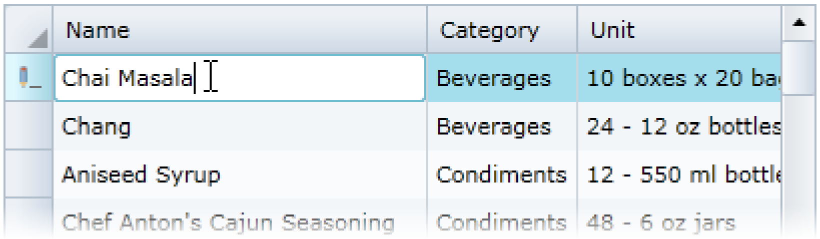
1. Double-click the cell you would like to edit.



Name	Category	Unit
Chai	Beverages	10 boxes x 20 ba
Chang	Beverages	24 - 12 oz bottles
Aniseed Syrup	Condiments	12 - 550 ml bottl
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars

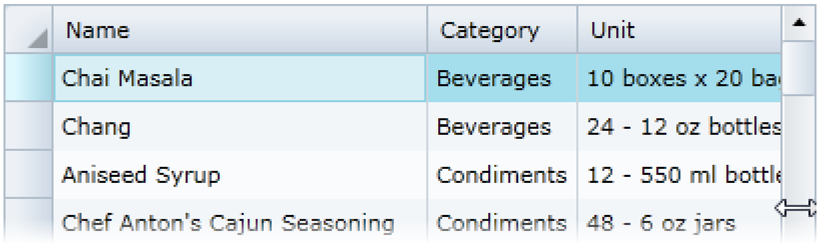
A cursor will appear in that cell indicating that it can be edited and a pencil icon will appear in the row indicator column, indicating that a cell in that row is in edit mode.

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



Name	Category	Unit
Chai Masala	Beverages	10 boxes x 20 ba
Chang	Beverages	24 - 12 oz bottles
Aniseed Syrup	Condiments	12 - 550 ml bottl
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars

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



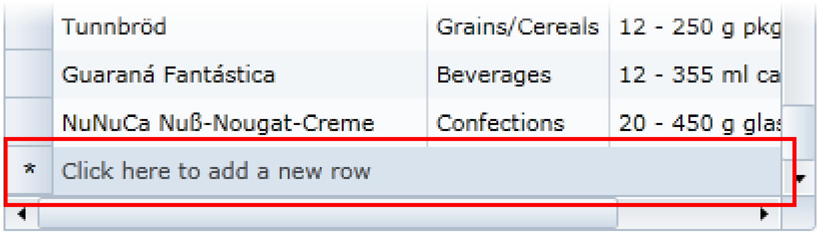
Name	Category	Unit
Chai Masala	Beverages	10 boxes x 20 ba
Chang	Beverages	24 - 12 oz bottles
Aniseed Syrup	Condiments	12 - 550 ml bottl
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars

The pencil icon indicating editing will no longer be visible.

Note that the **CanUserEditRows** property must be set to **True** (default) for editing to be possible. See Disabling Cell Editing (page 72) for an example.

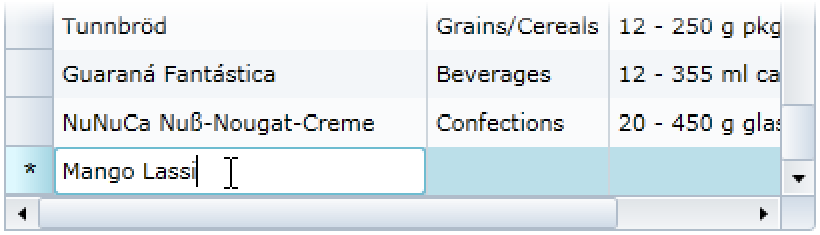
Adding Rows to the Grid

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



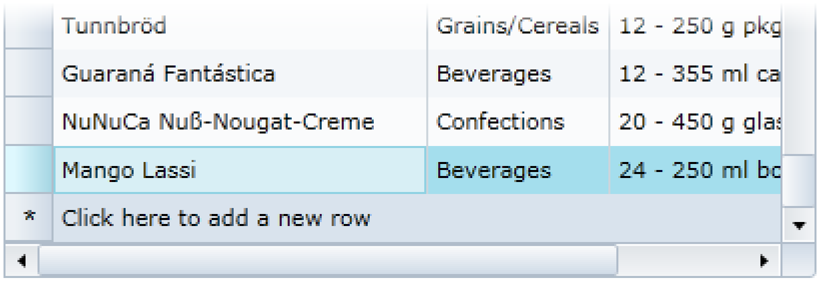
	Tunnbröd	Grains/Cereals	12 - 250 g pkg
	Guaraná Fantástica	Beverages	12 - 355 ml ca
	NuNuCa Nuß-Nougat-Creme	Confections	20 - 450 g glas
*	Click here to add a new row		

To add a new row, simply type text into the new row bar:



	Tunnbröd	Grains/Cereals	12 - 250 g pkg
	Guaraná Fantástica	Beverages	12 - 355 ml ca
	NuNuCa Nuß-Nougat-Creme	Confections	20 - 450 g glas
*	Mango Lassi		

Press ENTER for text to be added to the grid in a new row:



	Tunnbröd	Grains/Cereals	12 - 250 g pkg
	Guaraná Fantástica	Beverages	12 - 355 ml ca
	NuNuCa Nuß-Nougat-Creme	Confections	20 - 450 g glas
	Mango Lassi	Beverages	24 - 250 ml bo
*	Click here to add a new row		

Note that the **CanUserAddRows** property must be set to **True** (default) for row adding to be possible. See Disabling Adding Rows (page 72) for an example.

DataGrid for WPF Task-Based Help

The following task-based help topics assume that you are familiar with Visual Studio and Expression Blend and know how to use the C1DataGrid control in general. If you are unfamiliar with the **ComponentOne DataGrid for WPF** product, please see the DataGrid for WPF Quick Start (page 17) first.

Each topic in this section provides a solution for specific tasks using the **ComponentOne DataGrid for WPF** product. Most task-based help topics also assume that you have created a new WPF project and added a C1DataGrid control to the project – for information about creating the control, see Creating a DataGrid (page 65).

Creating a DataGrid

You can easily create a C1DataGrid control at design time in Expression Blend, in XAML, and in code. Note that if you create a C1DataGrid control as in the following steps, it will appear empty. You will need to bind the grid or populate it with data.

At Design Time in Blend

To create a C1DataGrid control in Blend, complete the following steps:

1. Navigate to the **Projects** window and right-click the **References** folder in the project files list. In the context menu choose **Add Reference**, locate and select the **C1.WPF.DataGrid.dll** assembly, and click **Open**.

   The dialog box will close and the references will be added to your project and the controls will be available in the Asset Library.

2. In the Toolbox click on the **Assets** button (the double chevron icon) to open the **Assets** dialog box.
3. In the **Asset Library** dialog box, choose the **Controls** item in the left pane, and then click on the **C1DataGrid** icon in the right pane:

   The **C1DataGrid** icon will appear in the Toolbox under the **Assets** button.

4. Click once on the design area of the **UserControl** to select it. Unlike in Visual Studio, in Blend you can add WPF controls directly to the design surface as in the next step.
5. Double-click the **C1DataGrid** icon in the Toolbox to add the control to the panel. The C1DataGrid control will now exist in your application.
6. If you choose, can customize the control by selecting it and setting properties in the Properties window. For example, set the C1DataGrid control's **Name** property to "c1datagrid1" the **Height** property to "180", and the **Width** property to "250".

In XAML

To create a C1DataGrid control using XAML markup, complete the following steps:

1. In the Visual Studio Solution Explorer, right-click the **References** folder in the project files list. In the context menu choose **Add Reference**, select the **C1.WPF.DataGrid.dll** assembly, and click **OK**.
2. Add a XAML namespace to your project by adding `xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"` to the initial `<UserControl>` tag. It will appear similar to the following:

```
<UserControl
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
x:Class="C1DataGrid.MainPage" Width="640" Height="480">
```

3. Add a `<c1:C1DataGrid>` tag to your project within the `<Grid>` tag to create a C1DataGrid control. The markup will appear similar to the following:

   ```
   <Grid x:Name="LayoutRoot" Background="White">
       <c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250" />
   </Grid>
   ```

   This markup will create an empty C1DataGrid control named "c1datagrid1" and set the control's size.

In Code

To create a C1DataGrid control in code, complete the following steps:

1. In the Visual Studio Solution Explorer, right-click the **References** folder in the project files list. In the context menu choose **Add Reference**, select the **C1.WPF.dll** and **C1.WPF.DataGrid.dll** assemblies, and click **OK**.
2. Right-click within the **MainPage.xaml** window and select **View Code** to switch to Code view
3. Add the following import statements to the top of the page:
   - Visual Basic

     ```
     Imports C1.WPF.DataGrid
     ```

   - C#

     ```
     using C1.WPF.DataGrid;
     ```

4. Add code to the page's constructor to create the C1DataGrid control. It will look similar to the following:
   - Visual Basic

     ```
     Public Sub New()
         InitializeComponent()
         Dim c1datagrid1 As New C1DataGrid
         c1datagrid1.Height = 180
         c1datagrid1.Width = 250
         LayoutRoot.Children.Add(c1datagrid1)
     End Sub
     ```

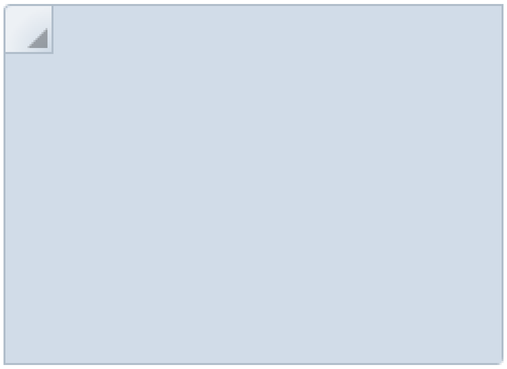
   - C#

     ```
     public MainPage()
     {
         InitializeComponent();
         C1DataGrid c1datagrid1 = new C1DataGrid();
         c1datagrid1.Height = 180;
         c1datagrid1.Width = 250;
         LayoutRoot.Children.Add(c1datagrid1);
     }
     ```

   This code will create an empty C1DataGrid control named "c1datagrid1", set the control's size, and add the control to the page.

What You've Accomplished

Run your application and observe that you've created a C1DataGrid control.



Note that when you create a C1DataGrid control as in the above steps, it will appear empty. You can add items to the control that can be interacted with at run time.

Controlling Grid Interaction

The following task-based help topics detail how you can enhance your users' interaction with **DataGrid for WPF**. For example, you can allow users to filter, sort, reorder, delete, and edit the grid through code and XAML.

Enabling Grouping in the Grid

You can enable grouping and the grouping area of the grid so that users can group columns in your grid at run time to better organize information. For more information, see Grouping Columns (page 59). By default, user cannot group columns in the grid but you can enable this function by setting the CanUserGroup property to **True**.

At Design Time

To enable grouping, complete the following steps:

1. Click the C1DataGrid control once to select it.
2. Navigate to the Properties window and locate the CanUserGroup property.
3. Check the check box next to the CanUserGroup property.

In XAML

For example to enable grouping, add `CanUserGroup="True"` to the `< c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
CanUserGroup=True" />
```

In Code

For example, to enable grouping, add the following code to your project:

- Visual Basic

```
Me.C1DataGrid1.CanUserGroup = True
```

- C#

```
this.c1DataGrid1.CanUserGroup = true;
```

What You've Accomplished

Run the application and notice that the grouping area appears at the top of the grid. Note that you can also customize the visibility of the grouping area. For more information about the grouping area, see the Showing the Grouping Area (page 68) topic.

Showing the Grouping Area

By default grouping in the grid is disabled and the grouping area is not visible. For more information, see Grouping Columns (page 59). When the CanUserGroup property is set to **True** and grouping is enabled the grouping area is made visible. But if you choose you can show or hide the grouping area whether or not grouping is enabled. By default, the grouping area is not visible when grouping is not enabled but you can make the area visible by setting the ShowGroupingPanel property to **True**.

At Design Time

To show the grouping area, complete the following steps:

1. Click the C1DataGrid control once to select it.
2. Navigate to the Properties window and locate the ShowGroupingPanel property.
3. Check the check box next to the ShowGroupingPanel property.

In XAML

For example to show the grouping area, add `ShowGroupingPanel="True"` to the `< c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1DataGrid1" Height="180" Width="250"
ShowGroupingPanel="True" />
```

In Code

For example, to show the grouping area, add the following code to your project:

- Visual Basic

```
Me.C1DataGrid1.ShowGroupingPanel = True
```

- C#

```
this.c1DataGrid1.ShowGroupingPanel = true;
```

What You've Accomplished

Run the application and notice that the grouping area appears at the top of the grid. Note that even if the grouping area is visible, grouping will not be enabled if the CanUserGroup property is **False**. For more information, see the Enabling Grouping in the Grid (page 67) topic.

Disabling Column Reordering

By default end users can easily reorder columns in the grid at run time. For more information, see Reordering Columns (page 56). If you choose, however, you can disable the column reordering feature by setting the CanUserReorderColumns property to **False**.

At Design Time

To disable column reordering, complete the following steps:

1. Click the C1DataGrid control once to select it.
2. Navigate to the Properties window and locate the CanUserReorderColumns property.
3. Clear the check box next to the CanUserReorderColumns property.

In XAML

For example to disable column reordering, add `CanUserReorderColumns="False"` to the `< c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
CanUserReorderColumns="False" />
```

In Code

For example, to disable column reordering, add the following code to your project:

- Visual Basic
```
Me.C1DataGrid1.CanUserReorderColumns = False
```
- C#
```
this.c1DataGrid1.CanUserReorderColumns = false;
```

What You've Accomplished

Run the application and observe that you can no longer reorder columns at run time by preforming a drag-and-drop operation. For more information about column reordering, see the Reordering Columns (page 56) topic.

Disabling Column and Row Resizing

By default end users can resize columns and rows in the grid at run time. For more information, see Resizing Columns and Rows (page 55). If you choose, however, you can disable the column and row resizing feature by setting the CanUserResizeColumns and CanUserResizeRows properties to **False**.

At Design Time

To disable column and row resizing, complete the following steps:

1. Click the C1DataGrid control once to select it.
2. Navigate to the Properties window and locate the CanUserResizeColumns property.
3. Clear the check box next to the CanUserResizeColumns property.
4. In the Properties window, locate the CanUserResizeRows property.
5. Clear the check box next to the CanUserResizeRows property.

In XAML

For example to disable column and row resizing, add `CanUserResizeColumns="False" CanUserResizeRows="False"` to the `< c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
CanUserResizeColumns="False" CanUserResizeRows="False"/>
```

In Code

For example, to disable column and row resizing, add the following code to your project:

- Visual Basic
```
Me.C1DataGrid1.CanUserResizeColumns = False
Me.C1DataGrid1.CanUserResizeRows = False
```
- C#
```
this.c1DataGrid1.CanUserResizeColumns = false;
this.c1DataGrid1.CanUserResizeRows = false;
```

What You've Accomplished

Run the application and observe that you can no longer resize columns or rows at run time by preforming a drag-and-drop operation. For more information about column reordering, see the Resizing Columns and Rows (page 55) topic.

Disabling Column Filtering

By default end users can filter columns in the grid at run time. For more information, see Filtering Columns (page 56). If you choose, however, you can disable the column filtering feature by setting the CanUserFilter property to **False**.

At Design Time

To disable column filtering, complete the following steps:

1. Click the C1DataGrid control once to select it.
2. Navigate to the Properties window and locate the CanUserFilter property.
3. Clear the check box next to the CanUserFilter property.

In XAML

For example to disable column filtering, add `CanUserFilter="False"` to the `< c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
CanUserFilter="False" />
```

In Code

For example, to disable column filtering, add the following code to your project:

- Visual Basic

```
Me.C1DataGrid1.CanUserFilter = False
```

- C#

```
this.c1DataGrid1.CanUserFilter = false;
```

What You've Accomplished

Run the application and observe that you can no longer filter columns at run time; the drop-down arrow to display the filter box is no longer visible at run time. For more information about column filtering, see the Filtering Columns (page 56) topic.

Disabling Column Sorting

By default end users can sort columns in the grid at run time. For more information, see Sorting Columns (page 58). If you choose, however, you can disable the column sorting feature by setting the CanUserSort property to **False**.

At Design Time

To disable column sorting, complete the following steps:

1. Click the C1DataGrid control once to select it.
2. Navigate to the Properties window and locate the CanUserSort property.
3. Clear the check box next to the CanUserSort property.

In XAML

For example to disable column sorting, add `CanUserSort="False"` to the `< c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
CanUserSort="False" />
```

In Code

For example, to disable column sorting, add the following code to your project:

- Visual Basic

```
Me.C1DataGrid1.CanUserSort = False
```

- C#

```
this.c1DataGrid1.CanUserSort = false;
```

What You've Accomplished

Run the application and observe that you can no longer sort columns at run time. Clicking on a column's header at run time will not sort the grid and the sort indicator is not visible in the column header. For more information about column sorting, see the Sorting Columns (page 58) topic.

Enabling Column Freezing

You may want to freeze columns in the grid at run time so that they are always visible even when the grid is scrolled horizontally. For more information, see Freezing Columns (page 61). This feature is not enabled by default, but if you choose you can enable the column freezing feature by setting the CanUserFreezeColumns property to **Left**.

At Design Time

To enable column freezing, complete the following steps:

1. Click the C1DataGrid control once to select it.
2. Navigate to the Properties window and locate the CanUserFreezeColumns property.
3. Click the drop-down arrow next to the CanUserFreezeColumns property and select **Left**.

In XAML

For example to enable column freezing, add `CanUserFreezeColumns="Left"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
CanUserFreezeColumns="Left" />
```

In Code

For example, to enable column freezing, add the following code to your project:

- Visual Basic

```
Me.C1DataGrid1.CanUserFreezeColumns = DataGridColumnFreezing.Left
```

- C#

```
this.c1DataGrid1.CanUserFreezeColumns = DataGridColumnFreezing.Left;
```

What You've Accomplished

Run the application and observe that the freeze bar is visible at run time. The freeze bar can be moved to select which columns to freeze; columns to the left of the bar will be frozen so that they are always visible even when the grid is scrolled horizontally. For more information about column freezing, see the Freezing Columns (page 61) topic.

Freezing Grid Rows

You may want to freeze the top or bottom rows in the grid at so that they are always visible even when the grid is scrolled vertically at run time. This feature is not enabled by default, but if you choose you can enable the row freezing feature by setting the FrozenTopRowsCount and FrozenBottomRowsCount properties.

At Design Time

To freeze the top and bottom two rows, complete the following steps:

1. Click the C1DataGrid control once to select it and navigate to the Properties window.
2. In the Properties window, locate the FrozenTopRowsCount property, click in the text box next to the property, and enter "2" to set the number of top tows that will be frozen.
3. Locate the FrozenBottomRowsCount property, click in the text box next to the property, and enter "2" to set the number of bottom rows that will be frozen.

In XAML

For example to freeze the top and bottom two rows, add `FrozenTopRowsCount="2" FrozenBottomRowsCount="2"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
FrozenTopRowsCount="2" FrozenBottomRowsCount="2" />
```

In Code

For example, to freeze the top and bottom two rows, add the following code to your project:

- Visual Basic

```
Me.C1DataGrid1.FrozenTopRowsCount = True
Me.C1DataGrid1.FrozenBottomRowsCount = True
```

- C#

```
this.c1DataGrid1.FrozenTopRowsCount = true;
this.c1DataGrid1.FrozenBottomRowsCount = true;
```

What You've Accomplished

Run the application and observe that the two top and bottom rows are frozen. Scroll the grid vertically and notice that the top two an bottom two rows do not scroll and are locked in place. By default the Add New row appears as the last row in the grid and so will be one of the frozen rows.

Disabling Cell Editing

By default end users edit content in the grid at run time. For more information, see Editing Cells (page 62). If you choose, however, you can disable the cell editing feature by setting the CanUserEditRows property to **False**.

At Design Time

To disable cell editing, complete the following steps:

1. Click the C1DataGrid control once to select it.
2. Navigate to the Properties window and locate the CanUserEditRows property.
3. Clear the check box next to the CanUserEditRows property.

In XAML

For example to disable cell editing, add `CanUserEditRows="False"` to the `< c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
CanUserEditRows="False" />
```

In Code

For example, to disable cell editing, add the following code to your project:

- Visual Basic

```
Me.C1DataGrid1.CanUserEditRows = False
```

- C#

```
this.c1DataGrid1.CanUserEditRows = false;
```

What You've Accomplished

Run the application and double-click a cell; observe that the cell does not move into edit mode and you can no longer edit grid content at run time. For more information about cell editing, see the Editing Cells (page 62) topic.

Disabling Adding Rows

By default end users add new rows and content to the grid at run time. A new row bar appears at the bottom of the grid, users can enter text in the bar to add new content to the grid. For more information, see Adding Rows to the Grid (page 63). If you choose, however, you can disable the new row bar feature by setting the CanUserAddRows property to **False**.

At Design Time

To disable adding rows, complete the following steps:

1. Click the C1DataGrid control once to select it.
2. Navigate to the Properties window and locate the CanUserAddRows property.
3. Clear the check box next to the CanUserAddRows property.

In XAML

For example to disable adding rows, add `CanUserEditRows="False"` to the `< c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
CanUserAddRows="False" />
```

In Code

For example, to disable adding rows, add the following code to your project:

- Visual Basic

```
Me.C1DataGrid1.CanUserAddRows = False
```

- C#

```
this.c1DataGrid1.CanUserAddRows = false;
```

What You've Accomplished

Run the application and scroll to the end of the grid, if needed. Observe that the new row bar no longer appears in the grid and that users can no longer add new rows and content to the grid. For more information about cell editing, see the Adding Rows to the Grid (page 63) topic.

Disabling Row Details Toggling

When the grid includes a child grid or you've created a master-detail grid, by default the row details can be toggled so that they are visible or collapsed. If you choose, however, you can disable the toggling the details row feature by setting the CanUserToggleDetails property to **False**. Note that you will need to have a grid with row details to view the change in this example.

At Design Time

To disable toggling row details, complete the following steps:

1. Click the C1DataGrid control once to select it.
2. Navigate to the Properties window and locate the CanUserToggleDetails property.
3. Clear the check box next to the CanUserToggleDetails property.

In XAML

For example to disable toggling row details, add `CanUserToggleDetails="False"` to the `< c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
CanUserToggleDetails="False" />
```

In Code

For example, to disable toggling row details, add the following code to your project:

- Visual Basic

```
Me.C1DataGrid1.CanUserToggleDetails = False
```

- C#

```
this.c1DataGrid1.CanUserToggleDetails = false;
```

What You've Accomplished

Run the application and observe that you can no longer toggle the row details in the grid at run time. The arrow icon in the row header that indicates that row details can be toggled is no longer visible so toggling rows is not an option.

Customizing Grid Appearance

The following task-based help topics detail how you can customize **DataGrid for WPF** by changing the grid's appearance. **DataGrid for WPF** includes several appearance options that incorporate ComponentOne's unique ClearStyle technology. For example, you can change the background color of the grid or the alternating row background. Note for more information about ClearStyle technology, see the C1DataGrid ClearStyle (page 48) topic. The follow topics also detail changing the layout of the grid, including how to set the location of the header and add new row bar.

Changing the Grid's Background and Foreground Color

ComponentOne DataGrid for WPF includes ComponentOne's unique ClearStyle technology that enables you to change the entire appearance of the grid simply and flawlessly. The following steps will detail how to set the **C1DataGrid.Background** property to completely change the appearance of the grid. For more details about ComponentOne's ClearStyle technology, see the C1DataGrid ClearStyle (page 48) topic.

At Design Time

To change the grid's foreground and background color so that it appears green, complete the following steps:

1. Click the C1DataGrid control once to select it.
2. Navigate to the Properties window and click the drop-down arrow next to the **Background** property.
3. Click the drop-down arrow in the box the hex code appears in, and choose **Green**.
4. Navigate to the Properties window and click the drop-down arrow next to the **Foreground** property.
5. Click the drop-down arrow in the box the hex code appears in, and choose **White**.

In XAML

For example to change the grid's foreground and background color so that it appears green, add `Background="Green" Foreground="White"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
Background="Green" Foreground="White" />
```

In Code

For example, to change the grid's foreground and background color so that it appears green, add the following code to your project:

- Visual Basic

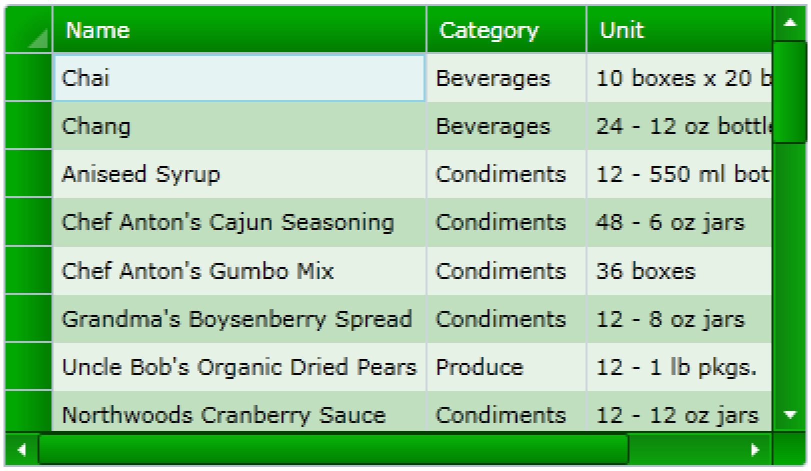
```
Me.C1DataGrid1.Background = New
System.Windows.Media.SolidColorBrush(Colors.Green)
Me.C1DataGrid1.ForeGround = New
System.Windows.Media.SolidColorBrush(Colors.White)
```

- C#

```
this.c1DataGrid1.Background = new System.Windows.Media.
SolidColorBrush(Colors.Green);
this.c1DataGrid1.Foreground = new System.Windows.Media.
SolidColorBrush(Colors.White);
```

What You've Accomplished

Run the application and observe that the grid now appears green with white text in the grid header.



Name	Category	Unit
Chai	Beverages	10 boxes x 20 b
Chang	Beverages	24 - 12 oz bottl
Aniseed Syrup	Condiments	12 - 550 ml bot
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars
Chef Anton's Gumbo Mix	Condiments	36 boxes
Grandma's Boysenberry Spread	Condiments	12 - 8 oz jars
Uncle Bob's Organic Dried Pears	Produce	12 - 1 lb pkgs.
Northwoods Cranberry Sauce	Condiments	12 - 12 oz jars

Note that with the C1DataGrid control's ClearStyle technology, the color of the grid, the grid's scrollbars, and the alternating row background of the grid all changed to reflect the green background. Highlight an item in the grid and notice the mouse hover style did not change; you can customize these styles as well if you choose. See Changing the Grid's Mouse Hover Style (page 76) for more details.

Removing the Grid's Alternating Row Colors

ComponentOne DataGrid for Silverlight appears with alternating row colors by default. Alternating row colors are when alternate lines appear in a different color than the base color of the grid. This is helpful so that rows are easier to follow across the grid, but if you choose you can make the appearance of the grid uniform by removing the alternating row colors.

At Design Time

To remove alternating row colors and set it so all rows appear white, complete the following steps:

1. Click the C1DataGrid control once to select it.
2. Navigate to the Properties window and click the drop-down arrow next to the **RowBackground** property.
3. Click the drop-down arrow in the box the hex code appears in, and choose **White**.
4. Navigate to the Properties window and click the drop-down arrow next to the **AlternatingRowBackground** property.
5. Click the drop-down arrow in the box the hex code appears in, and choose **White**.

In XAML

To remove alternating row colors and set it so all rows appear white, add `RowBackground="White" AlternatingRowBackground="White"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
RowBackground="White" AlternatingRowBackground="White" />
```

In Code

To remove alternating row colors and set it so all rows appear white, add the following code to your project:

- Visual Basic

```
Me.C1DataGrid1.RowBackground = New
System.Windows.Media.SolidColorBrush(Colors.White)
```

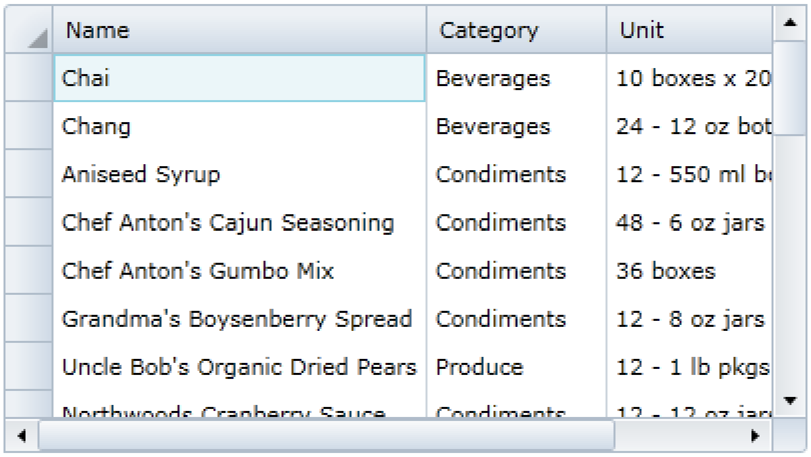
```
Me.C1DataGrid1.AlternatingRowBackground = New
System.Windows.Media.SolidColorBrush(Colors.White)
```

- C#

```
this.c1DataGrid1.RowBackground = new System.Windows.Media.
SolidColorBrush(Colors.White);
this.c1DataGrid1.AlternatingRowBackground = new System.Windows.Media.
SolidColorBrush(Colors.White);
```

What You've Accomplished

Run the application and observe that all rows in the grid now appear white.



Name	Category	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bot
Aniseed Syrup	Condiments	12 - 550 ml b
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars
Chef Anton's Gumbo Mix	Condiments	36 boxes
Grandma's Boysenberry Spread	Condiments	12 - 8 oz jars
Uncle Bob's Organic Dried Pears	Produce	12 - 1 lb pkgs

Changing the Grid's Mouse Hover Style

By default, columns and rows that are moused over appear in a different color to indicate to users what area of the grid they are interacting with. If you choose you can customize the appearance of cells that are moused over. For example, you may want to highlight these cells even more or remove this effect.

At Design Time

To set the mouse over effect to yellow, complete the following steps:

1. Click the C1DataGrid control once to select it.
2. Navigate to the Properties window and click the drop-down arrow next to the **MouseOverBrush** property.
3. Click the drop-down arrow in the box the hex code appears in, and choose **Yellow**.

In XAML

To set the mouse over effect to yellow, add `MouseOverBrush="Yellow"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
MouseOverBrush="Yellow" />
```

In Code

To set the mouse over effect to yellow, add the following code to your project:

- Visual Basic

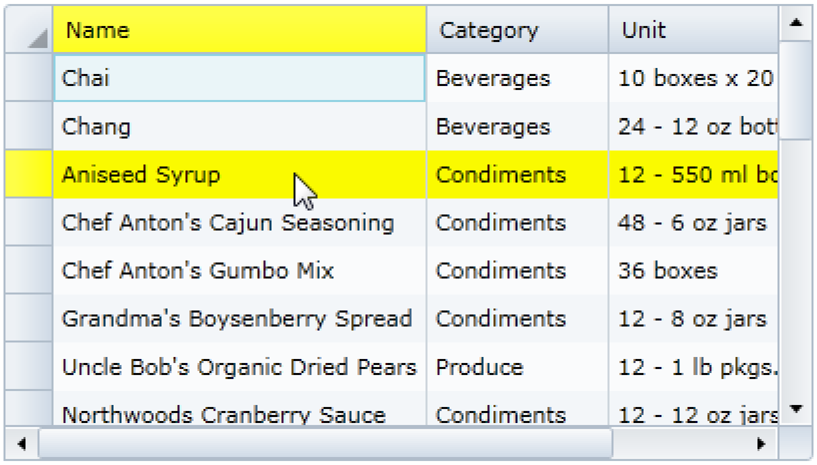
```
Me.c1datagrid1.MouseOverBrush = New
System.Windows.Media.SolidColorBrush(Colors.Yellow)
```

- C#

```
this.c1datagrid1.MouseOverBrush = new
System.Windows.Media.SolidColorBrush(Colors.Yellow);
```

What You've Accomplished

Run the application and observe that all highlighted rows and columns in the grid now appear yellow.



Name	Category	Unit
Chai	Beverages	10 boxes x 20
Chang	Beverages	24 - 12 oz bot
Aniseed Syrup	Condiments	12 - 550 ml bc
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars
Chef Anton's Gumbo Mix	Condiments	36 boxes
Grandma's Boysenberry Spread	Condiments	12 - 8 oz jars
Uncle Bob's Organic Dried Pears	Produce	12 - 1 lb pkgs.
Northwoods Cranberry Sauce	Condiments	12 - 12 oz jars

Changing the Grid's Font Style

You may want to update the font style that appears in **DataGrid for Silverlight** when the control is run. For example, you may want to change the style of the grid, an element of which is the font style, to match your application's appearance.

At Design Time

To change the font style, complete the following steps:

1. Click the C1DataGrid control once to select it.
2. Navigate to the Properties window and click the drop-down arrow next to the **FontFamily** property and choose **Times New Roman**.
3. Navigate to the Properties window and click the drop-down arrow next to the **FontSize** property and choose **10**.

In XAML

To change the font style, add `FontFamily="Times New Roman" FontSize="10"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:

```
<c1:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
FontFamily="Times New Roman" FontSize="10" />
```

In Code

To remove alternating row colors and set it so all rows appear white, add the following code to your project:

- Visual Basic

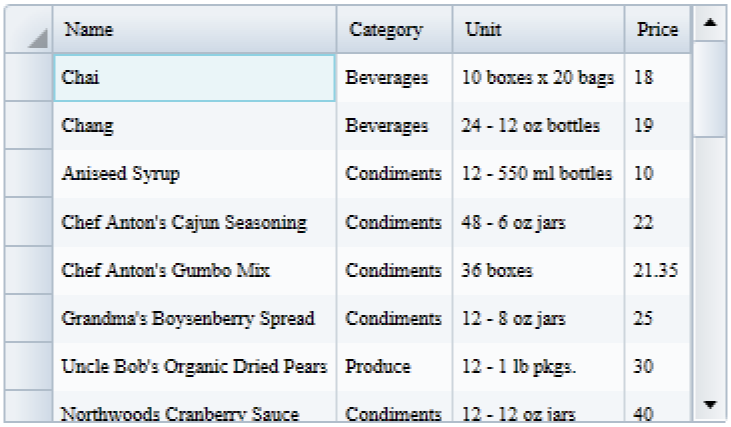
```
Me.c1datagrid1.FontFamily = New FontFamily("Times New Roman")
Me.c1datagrid1.FontSize = 10
```

- C#

```
this.c1datagrid1.FontFamily = new FontFamily("Times New Roman");
this.c1datagrid1.FontSize = 10;
```

What You've Accomplished

Run the application and observe that all rows in the grid appear in the Times New Roman font.



Name	Category	Unit	Price
Chai	Beverages	10 boxes x 20 bags	18
Chang	Beverages	24 - 12 oz bottles	19
Aniseed Syrup	Condiments	12 - 550 ml bottles	10
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars	22
Chef Anton's Gumbo Mix	Condiments	36 boxes	21.35
Grandma's Boysenberry Spread	Condiments	12 - 8 oz jars	25
Uncle Bob's Organic Dried Pears	Produce	12 - 1 lb pkgs.	30
Northwoods Cranberry Sauce	Condiments	12 - 12 oz jars	40