DataGrid for WPF
Copyright © 1987-2012 GrapeCity, Inc. All rights reserved.

ComponentOne, a division of GrapeCity
201 South Highland Avenue, Third Floor
Pittsburgh, PA 15206 • USA

Internet: info@ComponentOne.com
Web site: http://www.componentone.com
Sales
E-mail: sales@componentone.com
Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)

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

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

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

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

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

ComponentOne DataGrid for WPF Overview.................................................................................. 1

WPF WPF WPF WPF WPF WPF WPF WPF WPF Help with ComponentOne Studio for WPF............................... 1

Key Features..................................................................................................................................... 3

DataGrid for WPF Quick Start ........................................................................................................ 4
  Step 1 of 3: Adding Grid for WPF to your Project ....................................................................... 4
  Step 2 of 3: Binding the Grid to a Data Source ........................................................................... 5
  Step 3 of 3: Running the Grid Application ................................................................................... 8

Working with DataGrid for WPF .................................................................................................... 11
  Class Hierarchy ............................................................................................................................. 11
  Data Binding ................................................................................................................................ 11
  Defining Columns .......................................................................................................................... 12
    Generating Columns ..................................................................................................................... 12
    Column Types .............................................................................................................................. 12
    Explicitly Defining Columns ........................................................................................................ 13
    Customizing Automatically Generated Columns ......................................................................... 15
  Creating Custom Columns ............................................................................................................ 17
    Customizing Column Cell Content ............................................................................................ 17
    Adding Properties to a Custom Column ..................................................................................... 19
  Creating Custom Rows .................................................................................................................. 20
    Customizing Row Cell Content .................................................................................................. 20
    Adding a Custom Row to the Data Grid ..................................................................................... 23
  Adding Row Details ...................................................................................................................... 25
  Filtering the Grid ............................................................................................................................ 25
    Basic Column Filtering ................................................................................................................. 25
    Filter Row Filtering ..................................................................................................................... 26
    Full Text Grid Filtering ............................................................................................................... 26
    Advanced Filtering ...................................................................................................................... 27
    Column Filter List ....................................................................................................................... 27
    Tab Filter List ............................................................................................................................. 28
  Summarizing the Grid .................................................................................................................... 28
Localizing the Application ................................................................. 29
  Adding Resource Files .................................................................. 29
  Adding Supported Cultures .......................................................... 30
  Setting the Current Culture .......................................................... 31
Enabling or Disabling End User Interaction ..................................... 31
Setting Selection Mode ................................................................... 32
Locking the Grid ............................................................................. 33
Deferred and Real Time Scrolling .................................................... 33
DataGrid for WPF’s Appearance ....................................................... 33
  C1DataGrid Themes ..................................................................... 34
Editing Templates and Styles ........................................................... 36
Table Formatting Options ................................................................ 37
  Setting Row and Column Header Visibility ................................ 38
  Setting Grid Line Visibility ............................................................ 38
  Setting New Row Visibility ........................................................... 38
  Setting Vertical and Horizontal Scrollbar Visibility .................... 38
  Setting Row Details Visibility ...................................................... 39
C1DataGrid Brushes ....................................................................... 39
C1DataGrid ClearStyle .................................................................... 39
C1DataGrid Template Parts ............................................................. 41
RowDetailsTemplate ....................................................................... 42
Run-time Interaction ....................................................................... 43
  Keyboard and Mouse Navigation .................................................. 44
    Keyboard Navigation ................................................................. 44
    Mouse Navigation ..................................................................... 45
    Multiple Row Selection ............................................................. 46
    Custom Keyboard Navigation .................................................... 46
Resizing Columns and Rows ............................................................. 47
Reordering Columns ....................................................................... 48
Filtering Columns .......................................................................... 48
Sorting Columns ............................................................................ 50
Grouping Columns ......................................................................... 51
Freezing Columns ......................................................................... 53
Editing Cells .................................................................................. 54
Adding Rows to the Grid ................................................................. 55
DataGrid for WPF Samples .............................................................. 56
DataGrid for WPF Task-Based Help........................................................................57
Creating a DataGrid...............................................................................................57
 Controlling Grid Interaction ................................................................................59
   Enabling Grouping in the Grid ........................................................................59
   Showing the Grouping Area ............................................................................60
   Disabling Column Reordering .........................................................................60
   Disabling Column and Row Resizing .............................................................61
   Disabling Column Filtering ............................................................................61
   Disabling Column Sorting ..............................................................................62
   Enabling Column Freezing .............................................................................63
   Freezing Grid Rows .......................................................................................63
   Disabling Cell Editing ....................................................................................64
   Disabling Adding Rows ..................................................................................64
   Disabling Row Details Toggling .....................................................................65
 Customizing Grid Appearance ............................................................................66
   Changing the Grid's Background and Foreground Color ..............................66
   Removing the Grid's Alternating Row Colors ..............................................67
   Changing the Grid's Mouse Hover Style ......................................................68
   Changing the Grid's Font Style .....................................................................69
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.

For a list of the latest features added to ComponentOne Studio for WPF, visit What's New in Studio for WPF.

Getting Started

Get started with the following topics:
- Key Features (page 3)
- Quick Start (page 4)
- Task-Based Help (page 57)

Getting Started

For information on installing ComponentOne Studio for WPF, licensing, technical support, namespaces and creating a project with the control, please visit Getting Started with Studio for WPF.

What's New

For a list of the latest features added to ComponentOne Studio for WPF, visit What's New in Studio for WPF.
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 43) 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 51) 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 48) for more information.

- **High Performance**
  DataGrid for WPF 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 12) 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 25).

- **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 55) and Setting New Row Visibility (page 38) 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 39).

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.
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**
     
     ```vbnet
     Imports C1.WPF.DataGrid
     Imports ProjectName.ProductsDSTableAdapters
     ```

   - **C#**
     
     ```csharp
     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**
     
     ```vbnet
     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#**
     
     ```csharp
     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()
         {
```
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**

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

- **C#**

```csharp
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:

```
<cl: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:

![Main Window](image)

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.

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:

3. Resize a column, here the **ProductID** column, by clicking the right edge of the column and dragging the edge to a new location.
4. Click once on a cell, edit the contents of that cell, and press the ENTER key.

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<T>**.
For steps on binding a C1DataGrid control to an XML data source, see the DataGrid for WPF Quick Start (page 4).

**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:

<table>
<thead>
<tr>
<th>Column Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataGridBoundColumn</td>
<td>A column that can bind to a property in the grid's data source. This is the default column type for bound undefined data.</td>
</tr>
<tr>
<td>DataGridTextColumn</td>
<td>A text column. This is the default column type for bound string data.</td>
</tr>
<tr>
<td>DataGridCheckBoxColumn</td>
<td>A check box column. This is the default column type for bound Boolean data.</td>
</tr>
<tr>
<td>DataGridComboBoxColumn</td>
<td>A combo box column. This is the default column type for bound enumeration type data.</td>
</tr>
<tr>
<td>DataGridDateTimeColumn</td>
<td>A date time column (see below for an image). This is the default column type for bound date/time data.</td>
</tr>
<tr>
<td>DataGridImageColumn</td>
<td>An image column.</td>
</tr>
<tr>
<td>DataGridNumericColumn</td>
<td>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).</td>
</tr>
<tr>
<td>DataGridTemplateColumn</td>
<td>A template column for hosting custom content.</td>
</tr>
</tbody>
</table>
CustomColumns

A custom column. See the C1DataGrid_Demo sample for examples of custom 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:

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

For example:

- **XAML**

```xml
<DataGrid x:Name="grid" Grid.Row="1" Grid.ColumnSpan="2" Margin="5" AutoGeneratingColumn="grid_AutoGeneratingColumn" CanUserAddRows="False" ColumnHeaderHeight="30" >
  <DataGrid.Columns>
    <!-- Custom check box column. Adds a check box to the header and listens to its events. -->
    <DataGridCheckBoxColumn Binding="{Binding Available, Mode=TwoWay}" DisplayIndex="0" SortMemberPath="Available" FilterMemberPath="Available" MinWidth="108">
      <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>
      </DataGridColumn.Header>
    </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... -->
  </DataGrid.Columns>
</DataGrid>
```
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
     ```vb
     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#
     ```cs
     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:
   ```xml
   <c1:C1DataGrid x:Name="c1DataGrid1" AutoGenerateColumns="True"
                 AutoGeneratingColumn="c1DataGrid1_AutoGeneratingColumn"/>
   ```

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, and 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**

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

• **C#**

```csharp
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**

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

• **C#**

```csharp
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**

```vbnet
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#**

```csharp
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#

```csharp
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`.

```csharp
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 20) 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:
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
     ```vbasic
     Imports C1.WPF.DataGrid
     Public Class DataGridFilterRow
       Inherits DataGridRow
     End Class
     ```
   
   - C#
     ```csharp
     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**
  ```vbnet
  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#**
  ```csharp
  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**
     ```vbnet
     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#**
     ```csharp
     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))
       ```
2. In **UnbindCellContent** you must remove the text changed handler to avoid leaking memory:

- **Visual Basic**

```vbnet
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#**

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

- **Visual Basic**

  ```vbnet
  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#**

  ```csharp
  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**

  ```xaml
  <c1:C1DataGrid>
    <c1:C1DataGrid.TopRows>
      <local:DataGridFilterRow />  
    </c1:C1DataGrid.TopRows>
    <c1:C1DataGrid.BottomRows>
      <local:DataGridFilterRow/>  
    </c1:C1DataGrid.BottomRows>
  </c1:C1DataGrid>
  ```
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 42) topic.

Using the **RowDetailsVisibilityMode** property the row details section can be displayed for selected rows, displayed for all rows, or it can be collapsed. See [Setting Row Details Visibility](#) (page 39) for more information.

Filtering the Grid

**ComponentOne DataGrid for WPF** includes several options for filtering the grid. You can add column filtering, a filter row, or full-text grid filtering. There's basic filtering or you can use the C1.WPF.DataGrid.Filters.dll assembly which offers more advanced filtering options than are built into the grid. How you choose to filter the grid will depends on your needs – for example if you just want the end user to be able to filter text in a column or if your application requires more advanced custom filtering.

Basic Column Filtering

For basic column filtering, simply set the **CanUserFilter** property to **True**. This will add a filter column element to the grid's user interface allowing end users to filter the grid via a drop-down box in each column's header:

```
<cl:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
CanUserFilter="True" />
```

By default the **CanUserFilter** property will be set to **True** and filtering will be enabled. If you need to manually enable basic filtering, you can use the following markup or code:

- **XAML**
  ```xml
  <cl:C1DataGrid Name="c1datagrid1" Height="180" Width="250"
  CanUserFilter="True" />
  ```

- **Visual Basic**
  ```vbnet
  Me.C1DataGrid1.CanUserFilter = True
  ```

- **C#**
  ```csharp
  this.c1DataGrid1.CanUserFilter = true;
  ```

See the [Filtering Columns](#) (page 48) topic for more details and examples.
Filter Row Filtering

If you choose, you can add a visible filter row column to the top or bottom of your grid. The filter row column appears as a row consisting of text boxes in each cell. When text is entered in a text box, the text of the column and grid is filtered by that text as it is entered:

For example, the following markup adds two filter rows, one at the top and one at the bottom of the grid:

```xml
<cl:C1DataGrid x:Name="grid" Grid.Row="1" CanUserAddRows="False" CanUserFreezeColumns="True" FrozenTopRowsCount="1" FrozenBottomRowsCount="1" RowHeight="30" >
  <cl:C1DataGrid.TopRows>
    <cl:C1DataGridFilterRow />
  </cl:C1DataGrid.TopRows>
  <cl:C1DataGrid.BottomRows>
    <cl:C1DataGridFilterRow/>
  </cl:C1DataGrid.BottomRows>
</cl:C1DataGrid>
```

You can see the C1DataGrid_Demo2010/Filtering/FilterRow/FilterRow.xaml sample for an example.

Full Text Grid Filtering

C1DataGrid also supports full text filtering for the entire grid. Setting an attached property to the data grid allows the end user to filter the whole data grid (all the columns at once) by text entered in an external text box. All the matching results in the grid will be highlighted as the user types:
To use this method of grid filtering, you would need to add a text box control to your application and reference that control in the FullTextSearchBehavior attached property. For example, with the following XAML markup:

```xml
<StackPanel>
    <c1:C1TextBoxBase x:Name="filterTextBox" Width="200" Watermark = "Type here to filter text"/>
    <c1:C1DataGrid x:Name="c1dg" c1:C1NagScreen.Nag="True">
        <c1:C1FullTextSearchBehavior.FullTextSearchBehavior>
            <c1:C1FullTextSearchBehavior Filter="{Binding ElementName=filterTextBox,Path=C1Text}"/>
        </c1:C1FullTextSearchBehavior.FullTextSearchBehavior>
    </c1:C1DataGrid>
</StackPanel>
```

You can see the C1DataGrid_Demo2010/Filtering/OneTextBoxFilter/OneTextBoxFilter.xaml sample for an example.

**Advanced Filtering**

One way to add advanced filtering is by using C1AdvancedFiltersBehavior. C1AdvancedFiltersBehavior adds a range of advanced filters to the C1DataGrid built-in columns. For example, this behavior adds several predefined filters, expanding the options for each column:

```xml
<c1:C1DataGrid>
    <c1:C1AdvancedFiltersBehavior.AdvancedFiltersBehavior/>
    <c1:C1AdvancedFiltersBehavior.AdvancedFiltersBehavior/>
    <c1:C1AdvancedFiltersBehavior.AdvancedFiltersBehavior/>
</c1:C1DataGrid>
```

You can find a sample running in $\SilverlightMain\Samples\SL\C1.Silverlight.DataGrid\C1DataGrid_Demo, file DemoGrid.xaml.

**Column Filter List**

An option for filtering the grid is to add a list of filters to a column in XAML. For example, the following markup adds three filters in a numeric column including a custom filter called RangeFilter:
You can find a sample in $\text{Silverlight}\text{Main}\text{Samples}\text{SL}\text{C1.Silverlight.DataGrid}\text{C1DataGrid_Demo}, file Filtering/CustomFilters/CustomFilters.xaml.

**Tab Filter List**

An option for filtering the grid is to add a list of filters displayed in a tab control:

```xml
<DataGrid NumericColumn Header="Range filter" Binding="\{Binding StandardCost\}" FilterMemberPath="StandardCost">
    <DataGrid NumericColumn.Filter>
        <DataGridContentFilter>
            <DataGridFilterList>
                <DataGrid RangeFilter Minimum="0" Maximum="1000"/>
                <DataGridNumericFilter/>
                <DataGridTextFilter/>
            </DataGridFilterList>
        </DataGridContentFilter>
    </DataGrid NumericColumn.Filter>
</DataGrid NumericColumn>
```

You can find a sample in $\text{Silverlight}\text{Main}\text{Samples}\text{SL}\text{C1.Silverlight.DataGrid}\text{C1DataGrid_Demo}, file Filtering/CustomFilters/CustomFilters.xaml.

**Summarizing the Grid**

**ComponentOne DataGrid for WPF** includes the C1.WPF.DataGrid.Summaries.dll assembly which can enhance the grid by adding a summary row.

The Summaries assembly contains the following features:

- **SummaryRow.** a row that shows the aggregate functions corresponding to each column (See C1DataGrid_Demo2010/Grouping/GrandTotal.xaml)
- **GroupRowWithSummaries** the same as the previous one but the summaries are shown in the group row rather than a regular row. (See C1DataGrid_Demo2010/Grouping/Grouping.xaml)
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 WinForms applications.

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

1. Add resource files for each culture that you wish to support. See Adding Resource Files (page 29).
2. Update your project file's supported cultures. See Adding Supported Cultures (page 30).
3. And, depending on your project, set the current culture. See Setting the Current Culture (page 31).

The following topics describe localizing the grid in more detail.

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:

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

**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:

```xml
<Supported Cultures>fr;es;en;it;ru</Supported Cultures>
```

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**

  ```vbnet
  Public Sub New()
  ' Set desired culture, for example here the French (France) locale.
  ' InitializeComponent() call.
  InitializeComponent()
  ' Add any initialization after the InitializeComponent() call.
  InitializeComponent()
  End Sub
  ```

- **C#**

  ```csharp
  public MainPage()
  {
    // Set desired culture, for example here the French (France) locale.
    // 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:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanUserAddRows</td>
<td>Determines if users can add rows at run time. <strong>True</strong> by default.</td>
</tr>
<tr>
<td>CanUserEditRows</td>
<td>Determines if users can edit rows at run time. <strong>True</strong> by default.</td>
</tr>
<tr>
<td>CanUserFilter</td>
<td>Determines if users can filter columns at run time. If <strong>True</strong>, the filter bar will be visible on columns. <strong>True</strong> by default.</td>
</tr>
<tr>
<td>CanUserGroup</td>
<td>Determines if users can group rows at run time. If <strong>True</strong>, the grouping area of the grid will be visible. <strong>False</strong> by default.</td>
</tr>
<tr>
<td>CanUserRemoveRows</td>
<td>Determines if users can remove rows at run time by pressing the DELETE key. <strong>True</strong> by default.</td>
</tr>
<tr>
<td>CanUserReorderColumns</td>
<td>Determines if users can reorder columns at run time by using a</td>
</tr>
</tbody>
</table>
In each column you can customize the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanUserMove</td>
<td>Determines if users can reorder this column at run time. <strong>True</strong> by default.</td>
</tr>
<tr>
<td>CanUserResize</td>
<td>Determines if users can resize this column at run time. <strong>True</strong> by default.</td>
</tr>
<tr>
<td>CanUserFilter</td>
<td>Determines if users can filter this column at run time. If <strong>True</strong>, the filter bar will be visible on this column. <strong>True</strong> by default.</td>
</tr>
<tr>
<td>CanUserSort</td>
<td>Determines if users can sort this column at run time. <strong>True</strong> by default.</td>
</tr>
</tbody>
</table>

**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:

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

For more information about modifier keys and the **MultiRow** option, see the [Multiple Row Selection](#) (page 46) 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:

```xml
<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
  ```vbnet
  Me.C1DataGrid1.IsReadOnly = True
  ```
- C#
  ```csharp
  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:

```xml
<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
  ```vbnet
  Me.C1DataGrid1.ScrollMode = C1DataGridScrollMode.Deferred
  ```
- C#
  ```csharp
  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 39).
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:

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:

<table>
<thead>
<tr>
<th>Theme Name</th>
<th>Theme Preview</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1ThemeBureauBlack</td>
<td><img src="image" alt="Theme Preview" /></td>
</tr>
</tbody>
</table>
### C1ThemeExpressionDark

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chai</td>
<td>Beverages</td>
<td>10 boxes x 20</td>
</tr>
<tr>
<td>Chang</td>
<td>Beverages</td>
<td>24 - 12 oz bottle</td>
</tr>
<tr>
<td>Aniseed Syrup</td>
<td>Condiments</td>
<td>12 - 550 ml bottle</td>
</tr>
<tr>
<td>Chef Anton’s Cajun Seasoning</td>
<td>Condiments</td>
<td>48 - 6 oz jars</td>
</tr>
<tr>
<td>Chef Anton’s Gumbo Mix</td>
<td>Condiments</td>
<td>36 boxes</td>
</tr>
<tr>
<td>Grandma’s Boysenberry Spread</td>
<td>Condiments</td>
<td>12 - 8 oz jars</td>
</tr>
</tbody>
</table>

### C1ThemeExpressionLight

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chai</td>
<td>Beverages</td>
<td>10 boxes x 20</td>
</tr>
<tr>
<td>Chang</td>
<td>Beverages</td>
<td>24 - 12 oz bottle</td>
</tr>
<tr>
<td>Aniseed Syrup</td>
<td>Condiments</td>
<td>12 - 550 ml bottle</td>
</tr>
<tr>
<td>Chef Anton’s Cajun Seasoning</td>
<td>Condiments</td>
<td>48 - 6 oz jars</td>
</tr>
<tr>
<td>Chef Anton’s Gumbo Mix</td>
<td>Condiments</td>
<td>36 boxes</td>
</tr>
<tr>
<td>Grandma’s Boysenberry Spread</td>
<td>Condiments</td>
<td>12 - 8 oz jars</td>
</tr>
</tbody>
</table>

### C1ThemeRainierOrange

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chai</td>
<td>Beverages</td>
<td>10 boxes x 20</td>
</tr>
<tr>
<td>Chang</td>
<td>Beverages</td>
<td>24 - 12 oz bottle</td>
</tr>
<tr>
<td>Aniseed Syrup</td>
<td>Condiments</td>
<td>12 - 550 ml bottle</td>
</tr>
<tr>
<td>Chef Anton’s Cajun Seasoning</td>
<td>Condiments</td>
<td>48 - 6 oz jars</td>
</tr>
<tr>
<td>Chef Anton’s Gumbo Mix</td>
<td>Condiments</td>
<td>36 boxes</td>
</tr>
<tr>
<td>Grandma’s Boysenberry Spread</td>
<td>Condiments</td>
<td>12 - 8 oz jars</td>
</tr>
</tbody>
</table>
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:

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

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:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Neither row nor column headers are visible in the grid.</td>
</tr>
<tr>
<td>Column</td>
<td>Only column headers are visible in the grid.</td>
</tr>
<tr>
<td>Row</td>
<td>Only row headers are visible in the grid.</td>
</tr>
<tr>
<td>All (default)</td>
<td>Both column and row headers are visible in the grid.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Neither horizontal nor vertical grid lines are visible in the grid.</td>
</tr>
<tr>
<td>Horizontal</td>
<td>Only horizontal grid lines are visible in the grid.</td>
</tr>
<tr>
<td>Vertical</td>
<td>Only vertical grid lines are visible in the grid.</td>
</tr>
<tr>
<td>All (default)</td>
<td>Both horizontal and vertical grid lines are visible in the grid.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top</td>
<td>The Add New row appears at the top of the grid.</td>
</tr>
<tr>
<td>Bottom (default)</td>
<td>The Add New row appears at the bottom of the grid.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disabled</td>
<td>The chosen scrollbar is disabled.</td>
</tr>
<tr>
<td>Auto (default)</td>
<td>The chosen scrollbar appears only when the content of the grid is exceeds the grid window.</td>
</tr>
<tr>
<td>Hidden</td>
<td>The chosen scrollbar appears to be hidden.</td>
</tr>
<tr>
<td>Visible</td>
<td>The chosen scrollbar is always visible.</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VisibleWhenSelected</td>
<td>Row details are only visible when selected.</td>
</tr>
<tr>
<td>Visible</td>
<td>Row details are always visible.</td>
</tr>
<tr>
<td>collapsed (default)</td>
<td>Row details appear collapsed and are not visible.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Brush</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Background</td>
<td>Gets or sets the background brush that is used when rendering. (This brush will be applied to all the parts of the data grid)</td>
</tr>
<tr>
<td>Foreground</td>
<td>Gets or sets the foreground brush that is used when rendering. (This brush will be applied to all the parts of the data grid)</td>
</tr>
<tr>
<td>BorderBrush</td>
<td>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)</td>
</tr>
<tr>
<td>SelectedBrush</td>
<td>Gets or sets the selected brush that is used when rendering selected rows and row and column headers, etc.</td>
</tr>
<tr>
<td>MouseOverBrush</td>
<td>Gets or sets the mouse over brush that is used when mouse is over rows and row and column headers, etc.</td>
</tr>
<tr>
<td>RowBackground</td>
<td>Gets or sets the background brush of a row.</td>
</tr>
<tr>
<td>RowForeground</td>
<td>Gets or sets the foreground brush of a row.</td>
</tr>
<tr>
<td>AlternatingRowBackground</td>
<td>Gets or sets the background brush of an alternating row.</td>
</tr>
<tr>
<td>AlternatingRowForeground</td>
<td>Gets or sets the foreground brush of an alternating row.</td>
</tr>
<tr>
<td>HorizontalGridLinesBrush</td>
<td>Gets of sets the brush applied to the horizontal lines.</td>
</tr>
<tr>
<td>VerticalGridLinesBrush</td>
<td>Gets of sets the brush applied to the vertical lines.</td>
</tr>
</tbody>
</table>

**ComponentOne DataGrid for WPF** uses ClearStyle technology for styling. For details, see [C1DataGrid ClearStyle](#) (page 39).

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:

```xml
<cl: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:

![Image of C1DataGrid with Background set to #FF663366](image)

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

```xml
<cl: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:

![Image of C1DataGrid with Background set to #FF663366 and Foreground set to White](image)

You can even set the **Background** property to a gradient value, for example with the following XAML:
The grid will appear similar to the following image:

![Grid Example](image)

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:
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:

![HorizontalScrollBar (ScrollBar)](Image)

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

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

**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 checkboxes:

```xml
<cl:C1DataGrid.RowDetailsTemplate>
  <!-- Begin row details section. -->
  <DataTemplate>
    <Border BorderBrush="DarkGray" BorderThickness="1"
           Background="Azure">
      <StackPanel Orientation="Horizontal">
        <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:
           Complete:" VerticalAlignment="Center" />
        </StackPanel>
        <TextBlock FontSize="12" Text="Notes:" />
        <TextBox FontSize="12" Text="{Binding Notes, Mode=TwoWay}" Width="420" TextWrapping="Wrap"/>
      </StackPanel>
    </Border>
  </DataTemplate>
  <!-- End row details section. -->
</cl: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:

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOWN Arrow</td>
<td>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.</td>
</tr>
<tr>
<td>UP Arrow</td>
<td>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.</td>
</tr>
<tr>
<td>LEFT ARROW</td>
<td>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.</td>
</tr>
<tr>
<td>RIGHT Arrow</td>
<td>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.</td>
</tr>
<tr>
<td>HOME</td>
<td>Moves the focus to the first cell in the current row.</td>
</tr>
<tr>
<td>END</td>
<td>Moves the focus to the last cell in the current row.</td>
</tr>
<tr>
<td>PAGE DOWN</td>
<td>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.</td>
</tr>
<tr>
<td>PAGE UP</td>
<td>Scrolls the control upward by the number of rows that are displayed.</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Keyboard Shortcut</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAB</td>
<td>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.</td>
</tr>
<tr>
<td>CTRL+DOWN ARROW</td>
<td>Moves the focus to the last cell in the current column.</td>
</tr>
<tr>
<td>CTRL+UP ARROW</td>
<td>Moves the focus to the first cell in the current column.</td>
</tr>
<tr>
<td>CTRL+RIGHT ARROW</td>
<td>Moves the focus to the last cell in the current row.</td>
</tr>
<tr>
<td>CTRL+LEFT ARROW</td>
<td>Moves the focus to the first cell in the current row.</td>
</tr>
<tr>
<td>CTRL+HOME</td>
<td>Moves the focus to the first cell in the control.</td>
</tr>
<tr>
<td>CTRL+PAGE DOWN</td>
<td>Same as PAGE DOWN.</td>
</tr>
<tr>
<td>CTRL+PAGE UP</td>
<td>Same as PAGE UP.</td>
</tr>
<tr>
<td>ENTER</td>
<td>Enter/exit edit mode on a selected cell (if the grid and column's IsReadOnly properties are False).</td>
</tr>
<tr>
<td>F2</td>
<td>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.</td>
</tr>
<tr>
<td>ESC</td>
<td>Cancel editing of a cell or new row.</td>
</tr>
<tr>
<td>DEL</td>
<td>Delete selected row.</td>
</tr>
<tr>
<td>INSERT</td>
<td>Scrolls to the new row and begins editing it.</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Mouse Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click an unselected row</td>
<td>Makes the clicked row the current row.</td>
</tr>
<tr>
<td>Click a cell in the current row</td>
<td>Puts the clicked cell into edit mode.</td>
</tr>
<tr>
<td>Drag a column header cell</td>
<td>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).</td>
</tr>
<tr>
<td>Drag a column header separator</td>
<td>Resizes the column (if the CanUserResizeColumns property is True and the CanUserResize property is True for the current column).</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>CTRL+click a row</th>
<th>Modifies a non-contiguous multi-row selection (if <strong>SelectionMode</strong> support multiple rows, cells, or columns).</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHIFT+click a row</td>
<td>Modifies a contiguous multi-row selection (if <strong>SelectionMode</strong> support multiple rows, cells, or columns).</td>
</tr>
</tbody>
</table>

**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
     ```vbnet
     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#
     ```csharp
     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:
   ```xml
   <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:

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

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 61) 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:

   ![Column Reordering Example](image)

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 60) 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:**

   ![Filter Column Example](image)

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:
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:

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:

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

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

To remove the grouping, simply click the X button next to a grouped column in the grouping area of the grid:
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 59). For more information about showing the grouping area, see the [Showing the Grouping Area](#) (page 60) 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:

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:

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:
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 63) 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.

![Double-click cell](image)

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:

![Edit cell](image)

3. Press ENTER or click away from the cell you are editing for the changes you made to take effect:
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 64) 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:

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

Press ENTER for text to be added to the grid in 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 64) for an example.

DataGrid for WPF Samples

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

Several pages within the ControlExplorer sample installed with ComponentOne Studio for WPF detail the C1DataGrid control's functionality (grouping, filtering, sorting), appearance (conditional formatting, templates, custom columns and rows), behavior (drag-and-drop behavior, validation, export), and more.
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 4) 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 57).

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 <Window> tag. It will appear similar to the following:

   ```xml
   <Window 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.MainWindow" 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:

```xml
<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 MainWindow.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
     ```vbnet
     Imports C1.WPF.DataGrid
     ```
   - C#
     ```csharp
     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
     ```vbnet
     Public Sub New()
       InitializeComponent()
       Dim c1datagrid1 As New C1DataGrid
       c1datagrid1.Height = 180
       c1datagrid1.Width = 250
       LayoutRoot.Children.Add(c1datagrid1)
     End Sub
     ```
   - C#
     ```csharp
     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 51). 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:

```xml
<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**
  ```vbnet
  Me.C1DataGrid1.CanUserGroup = True
  ```

- **C#**
  ```csharp
  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 60) 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 51). 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:

```xml
<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
  ```vb
  Me.C1DataGrid1.ShowGroupingPanel = True
  ```

- C#
  ```csharp
  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 59) 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 48). 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:

```xml
<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:
What You've Accomplished

Run the application and observe that you can no longer reorder columns at run time by performing a drag-and-drop operation. For more information about column reordering, see the Reordering Columns (page 48) 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 47). 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 performing a drag-and-drop operation. For more information about column reordering, see the Resizing Columns and Rows (page 47) topic.

Disabling Column Filtering

By default end users can filter columns in the grid at run time. For more information, see Filtering Columns (page 48). 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:

```xml
<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
  ```vbnet
  Me.C1DataGrid1.CanUserFilter = False
  ```

* C#
  ```csharp
  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 48) topic.

**Disabling Column Sorting**

By default end users can sort columns in the grid at run time. For more information, see Sorting Columns (page 50). 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:

```xml
<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
  ```vbnet
  Me.C1DataGrid1.CanUserSort = False
  ```

* C#
  ```csharp
  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 50) 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 53). 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:

```xml
<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
  ```vbnet
  Me.C1DataGrid1.CanUserFreezeColumns = DataGridColumnFreezing.Left
  ```
- C#
  ```csharp
  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 53) 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 rows 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:

```xml
<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
  ```csharp
  Me.C1DataGrid1.FrozenTopRowsCount = True
  Me.C1DataGrid1.FrozenBottomRowsCount = True
  ```

- C#
  ```csharp
  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 and 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 54). 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:

```xml
<ct: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
  ```csharp
  Me.C1DataGrid1.CanUserEditRows = False
  ```

- C#
  ```csharp
  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 54) 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 55). 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 `CanUserAddRows="False"` to the `<c1:C1DataGrid>` tag so that it appears similar to the following:

```xml
<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
  ```vbnet
  Me.C1DataGrid1.CanUserAddRows = False
  ```
- C#
  ```csharp
  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 55) 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:

```xml
<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
  ```vbnet
  Me.C1DataGrid1.CanUserToggleDetails = False
  ```
- C#
  ```csharp
  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 39) 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 39) 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:

```xml
<cl: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**
  ```vbnet
  ```

- **C#**
  ```csharp
  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.
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 68) for more details.

Removing the Grid's Alternating Row Colors

ComponentOne DataGrid for WPF 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:

```xml
<cl:C1DataGrid Name="cldatagrid1" 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
```vbnet
```
What You've Accomplished

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

### 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:

```xml
<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

```vb
```

- C#
What You've Accomplished

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

### Changing the Grid's Font Style

You may want to update the font style that appears in **DataGrid for WPF** 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:

```xml
<cl: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**
  ```vbnet
  Me.c1datagrid1.FontFamily = New FontFamily("Times New Roman")
  Me.c1datagrid1.FontSize = 10
  ```

- **C#**
  ```csharp
  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.

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