ComponentOne

DataGrid for WPF and Silverlight

GrapeCity US

GrapeCity
201 South Highland Avenue, Suite 301
Pittsburgh, PA 15206
Tel: 1.800.858.2739 | 412.681.4343
Fax: 412.681.4384
Website: https://www.grapecity.com/en/
E-mail: us.sales@grapecity.com

Trademarks

Warranty

ComponentOne warrants that the media on which the software is delivered is 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 media 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 the defective media by sending it and a check for $25 (to cover postage and handling) to ComponentOne.

Except for the express warranty of the original media on which the software is delivered is 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. ComponentOne is 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

Table of Contents

DataGrid for WPF and Silverlight Overview 5-6
Help with WPF and Silverlight Edition 6
Data Grid Feature Comparison Matrix 6-9
Getting Started 9
Quick Start 9
Step 1 of 4: Adding C1DataGrid to your Project 9-10
Step 2 of 4: Creating the Data Model 10-11
Step 3 of 4: Setting the ItemsSource 11
Step 4 of 4: Running the Grid Application 11-13
Class hierarchy 13-14
Creating a C1DataGrid in Expression Blend 14
Main Concepts and Properties 14
DataGrid Features 14
Add New Row 14-15
Disable Adding New Rows 15-16
Column and Row Resizing 16-17
Disabling resizing of rows and columns 17
Column Reordering 17
Column Types 17-18
Custom Columns 18
Customizing Column Cell Content 18-20
Custom Rows 20
Customizing Row Cell Content 20-23
Adding a Custom Row 23-24
Adding Row Details 24
Data Binding 24-25
New Topic 4 25
Deferred Scrolling 25
Defining Columns 25-26
Generating Columns 26
Editing 26
Editing Cells 26-27
Disabling Cell Editing 27
Locking the Grid 27-28

Filtering 28
Basic Column Filtering 28
Filtering Columns 28-30
Filter Row Filtering 31
Full Text Grid Filtering 31-32
Advanced Filtering 32
Column Filter List 32
Disabling Column Filtering 32-33
Tab Filter List 33
Freezing 33-34
Enabling Column Freezing 34-35
Freezing Grid Rows 35-36
Grouping 36
Showing the Grouping Area 36-37
Grouping Columns 37-38
Group Summaries 38-39
Keyboard and Mouse Navigation 39
Keyboard Navigation 39-40
Mouse Navigation 40-41
Multiple Row Selection 41
Custom Keyboard Navigation 41-42
Localizing the Application 42
Adding Resource Files 42-44
Adding Supported Cultures 44
Setting the Current Culture 44-45
Row Details 45
Row Details Template 45
Disabling Row Details Toggling 45-46
Sorting 46-47
Disabling Column Sorting 47
DataGrid Appearance 47
C1DataGrid Themes 47-49
Editing Templates and Styles 49-50
Table Formatting Options 50
Setting Row and Column Header Visibility 50-51
Setting Grid Line Visibility 51

Setting New Row Visibility 51
Setting Vertical and Horizontal Scrollbar Visibility 51-52
Setting Row Details Visibility 52
C1DataGrid Brushes 52-53
C1DataGrid Clear Style 53-55
C1DataGrid Template Parts 55-56
Customizing Grid Appearance 56
Changing the Grid's Background and Foreground Color 56-57
Removing the Grid's Alternating Row Colors 57
Changing the Grid's Mouse Hover Style 57-58
Changing the Grid's Font Style 58-59
Run-time Interaction 59
Selection Mode 59-60
Customizing Automatically Generated Columns 60-62
DataGrid Samples 62
Tutorials 62
New Topic 5 62
Step 1 of 3: Creating the User Interface 62-64
Step 2 of 3: Adding a Web Service 64-68
Step 3 of 3: Connecting the Web Service 68-71
New Topic 6 71
Creating a Master Detail View 71-72
Step 1 of 3: Setting up the Master/Detail Grid 72-73
Step 2 of 3: Adding a Data Source to the Project 73
Step 3 of 3: Setting up Row Details 73-75
Localizing the Grid 75
Step 1 of 3: Setting up the Localized Grid 75-78
Step 2 of 3: Adding a Resource File 78-80
Step 3 of 3: Setting the Culture 80-82
Binding the Grid to a WCF RIA Services Data Source 82
Step 1 of 3: Creating the Application and Adding the Data Source 82-83
Step 2 of 3: Adding the C1DataGrid Control 83-89
手順 3:アプリケーションの実行 89-90
Implementing Stealth Paging 90
Step 1 of 3: Creating the User Interface 90-92
Step 2 of 3: Adding a Web Service 92-96

Step 3 of 3: Connecting the Web Service and Adding Stealth Paging 96-100
Task-Based Help 100
Creating a DataGrid 100-102

DataGrid for WPF and Silverlight Overview

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

DataGrid for WPF and Silverlight includes several key features, such as:

- **Fully Interactive Grid**

  Enhance the end-user experience by creating a fully interactive grid. **C1DataGrid** has many built-in interactive features such as column resizing and reordering, editing, sorting, filtering, grouping, freezing, and selecting. See Run-time Interaction for more information.

- **Data Grouping and Totals**

  C1DataGrid 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 for details.

- **Excel-like Filtering**

  By default, **C1DataGrid** 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 for more information.

- **High Performance**

  C1DataGrid utilizes both row and column recycling (UI Virtualization) to achieve optimal performance when handling large data sets.

- **Several Built-in Column Types**

  C1DataGrid 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 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.

- **Top and Bottom Row Templates**

  With **C1DataGrid**'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 **C1DataGrid**'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 **C1DataGrid** by displaying an empty new row at either the top or bottom of the grid. See Adding Rows to the Grid and Setting New Row Visibility 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**

  C1DataGrid 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.

Help with WPF and Silverlight Edition

Getting Started

- For information on installing **ComponentOne Studio WPF Edition**, licensing, technical support, namespaces, theming and localization, and creating a project with the control, please visit Getting Started with WPF Edition.
- For information on installing **ComponentOne Studio Silverlight Edition**, licensing, technical support, namespaces, theming and localization, and creating a project with the control, please visit Getting Started with Silverlight Edition.

Data Grid Feature Comparison Matrix

Explore all of the features offered by the C1DataGrid control. You can download the matrix in PDF.

Data Binding

Features	C1FlexGrid	C1DataGrid	MS DataGrid
Bound mode	✓	✓	✓
Unbound mode	✓		

Layout and Appearance

Features	C1FlexGrid	C1DataGrid	MS DataGrid
Themes	17 Theme	17 Theme	
ClearStyle		✓	

Presentation

Features	C1FlexGrid	C1DataGrid	MS DataGrid
Autogenerate Columns	✓	✓	✓
Text Column	✓	✓	✓
CheckBox Column	✓	✓	✓
ComboBox Column	✓	✓	✓
Hyperlink Column		✓	✓
DateTime Column		✓	

Numeric Column		✓	
Image Column		✓	
Frozen Columns	✓	✓	✓
Frozen Rows	✓	✓	
Custom Columns	✓	✓	✓
Custom Rows		✓	
Custom Cells (Cell Factory)	✓		
Add New Row	✓	✓	✓
Merged Cells	✓	✓	
Row Details		✓	✓
Hierarchical View	✓	with custom code	

Sorting

Features	C1FlexGrid	C1DataGrid	MS DataGrid
ICollectionView	✓	✓	✓

Filtering

Features	C1FlexGrid	C1DataGrid	MS DataGrid
ICollectionView	✓	✓	✓
Excel-like Filtering	✓	✓	
Filter Row	with custom code	✓	
Custom Filters		✓	
Full-text Search		✓	

Grouping

Features	C1FlexGrid	C1DataGrid	MS DataGrid
ICollectionView	✓	✓	✓
Drag and Drop Grouping	✓	✓	
Subtotals	✓	✓	

Editing

Features	C1FlexGrid	C1DataGrid	MS DataGrid
In-cell Editing	✓	✓	✓
Validation	✓	✓	✓
IDataErrorInfo	✓	✓	✓
IEditableObject		✓	✓

ICustomTypeDescriptor	✓	✓	✓
Data Annotations	✓	✓	

Printing

Feature	C1FlexGrid	C1DataGrid	MS DataGrid
Printing	✓	✓	

Export

Features	C1FlexGrid	C1DataGrid	MS DataGrid
Excel	✓	✓	
Text	✓		
HTML	✓		

Ux

Features	C1FlexGrid	C1DataGrid	MS DataGrid
Keyboard Navigation	✓	✓	✓
RTL Support	✓	✓	✓
Touch Support	✓	✓	✓
Clipboard Support	✓	✓	✓
Multiple Selection Modes	✓	✓	✓

Localization

Features	C1FlexGrid	C1DataGrid	MS DataGrid
.NET Localization Support	✓	✓	✓
Included Translations	25 Languages	25 Languages	✓
Regional Settings (Number, date, currency)	✓	✓	✓

Performance

Features	C1FlexGrid	C1DataGrid	MS DataGrid
Deferred Scrolling	✓	✓	✓
UI	✓	✓	✓

Virtualization			
Server-side Data Virtualization with C1DataSource	✓	✓	

Other

Features	C1FlexGrid	C1DataGrid	MS DataGrid
Design-time Support	✓	✓	✓
WPF/Silverlight Compatibility	✓	✓	✓
UI Automation	✓	✓	✓
Assembly Size	287 KB	797 KB	part of PresentationFramework.dll

Getting Started

Quick Start

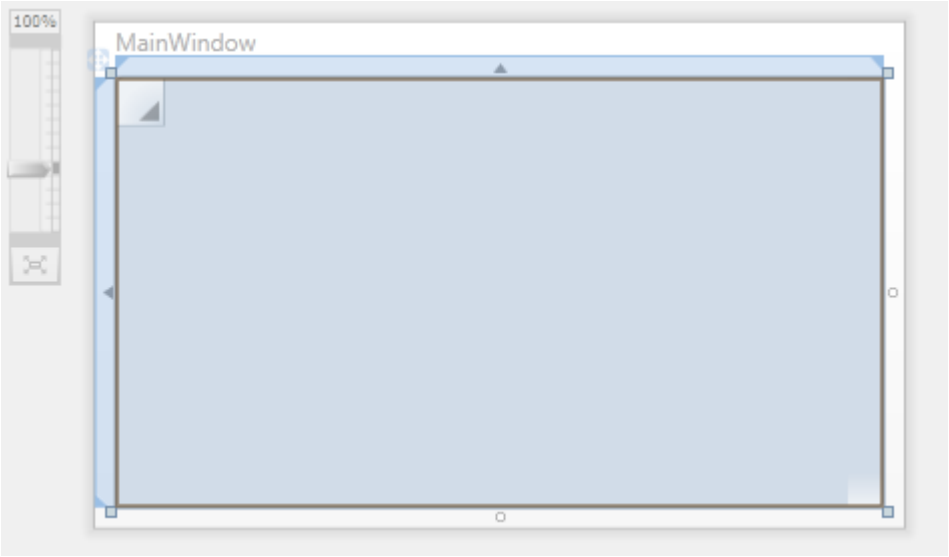
The following quick start guide is intended to get you up and running with **DataGrid for WPF and Silverlight**. In this quick start you'll start in Visual Studio and create a new project, add **C1DataGrid** to your application, and add a data source. You'll then move to Microsoft Expression Blend to complete binding the grid to the data source, customize the grid, and run the grid application to observe run-time interactions.

Step 1 of 4: Adding C1DataGrid 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:

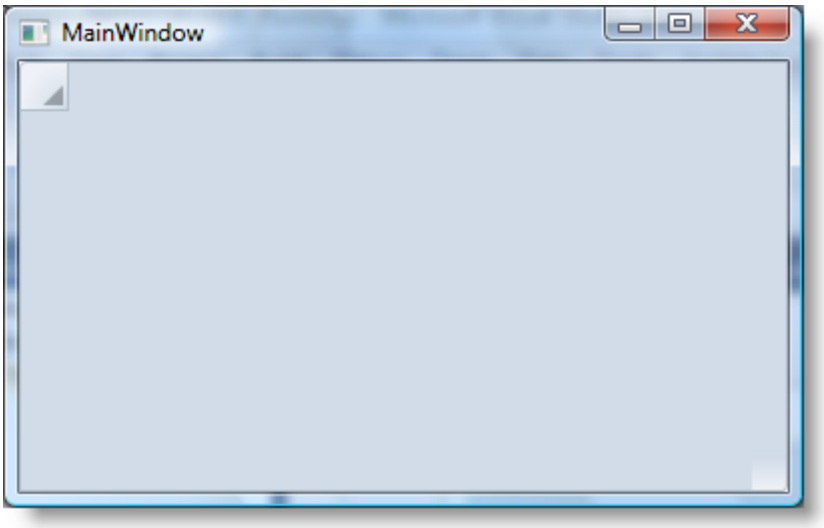
1. Create a new WPF project in Visual Studio.
2. Navigate to the Toolbox and double-click the **C1DataGrid** icon to add the grid control to Window1.
3. Resize the Window and the **C1DataGrid** within the Window; it should now look similar to the following:



4. Name the **C1DataGrid** control by setting its Name property to "ProductsDataGrid".

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 4: Creating the Data Model

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 add a data model to your project that you will later use to generate data to display in the **C1DataGrid** control.

1. To add a data model, complete the following steps:
2. Right-click the project node and select **Add | Class....**
3. Name the class – Product.cs and click **OK**.
4. Replace the generated Product class code with the following:

C#

```
public class Product
{
    static Random _rnd = new Random();
    static string[] _names = "Macko|Surfair|Pocohey|Studeby".Split('|');
    static string[] _lines = "Computers|Washers|Stoves|Cars".Split('|');
    static string[] _colors = "Red|Green|Blue|White".Split('|');

    public Product()
    {
        Name = _names[_rnd.Next() % _names.Length];
        Line = _lines[_rnd.Next() % _lines.Length];
        Color = _colors[_rnd.Next() % _colors.Length];
        Price = 30 + _rnd.NextDouble() * 1000;
        Cost = 3 + _rnd.NextDouble() * 300;
        Discontinued = _rnd.NextDouble() < .2;
        Introduced = DateTime.Today.AddDays(_rnd.Next(-600, 0));
    }

    public string Name { get; set; }
    public string Color { get; set; }
    public string Line { get; set; }
    public double Price { get; set; }
    public double Cost { get; set; }
    public DateTime Introduced { get; set; }
    public bool Discontinued { get; set; }
}
```

Step 3 of 4: Setting the ItemsSource

In the last step you added a data model named Product. In this step you will generate a collection of data objects using this model, and set this list to display in the C1DataGrid control.

To display a collection of data objects in the **C1DataGrid** control, complete the following steps:

1. Open MainWindow.cs or MainWindow.vb.
2. After the boiler-plate initialization code, add the following code which generates a list of 100 random products:

   C#

   ```
   List<Product> _products = new List<Product>();
   for(int i = 0; i < 100; i++)
   {
       _products.Add(new Product());
   }
   ```

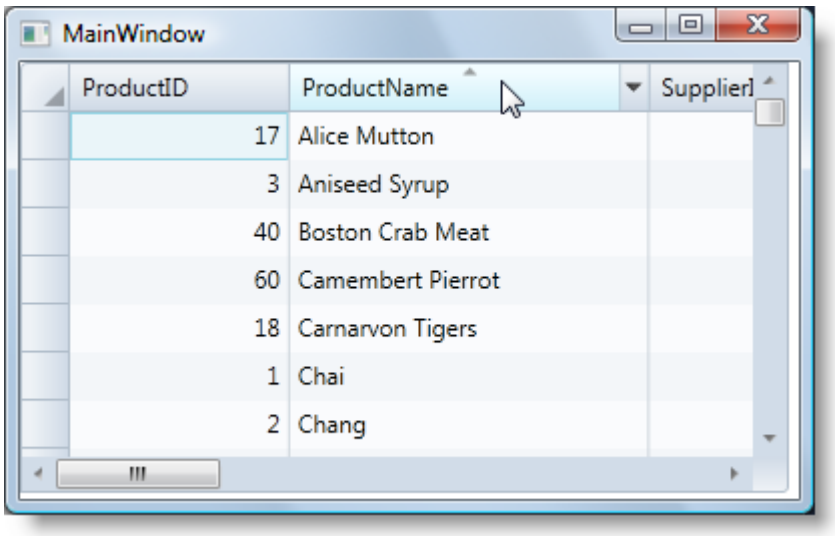
3. Set the ItemsSource property of the **C1DataGrid** to the collection of products. We named our **C1DataGrid** control as “ProductsDataGrid” back in step 1.

   ProductsDataGrid.ItemsSource = _products.

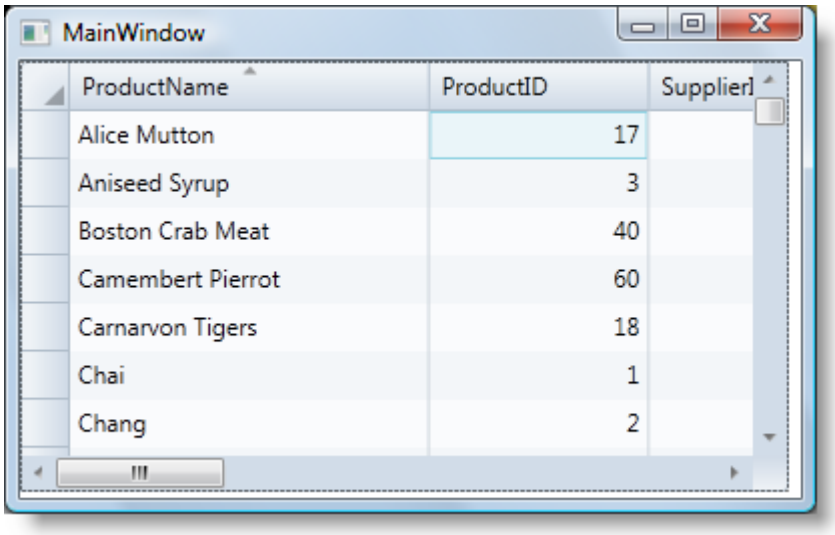
Step 4 of 4: Running the Grid Application

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:

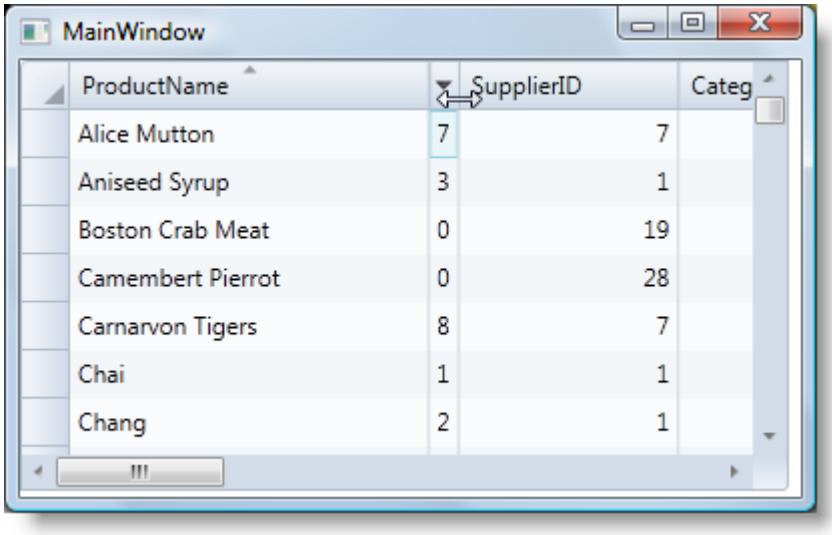
1. From the Debug menu, select **Start Debugging** to view how your grid application will appear at run time.
2. Click the **ProductName** header to sort the grid by product name. Notice that a sort indicator glyph appears to indicate the column being sorted and the direction of the sort.



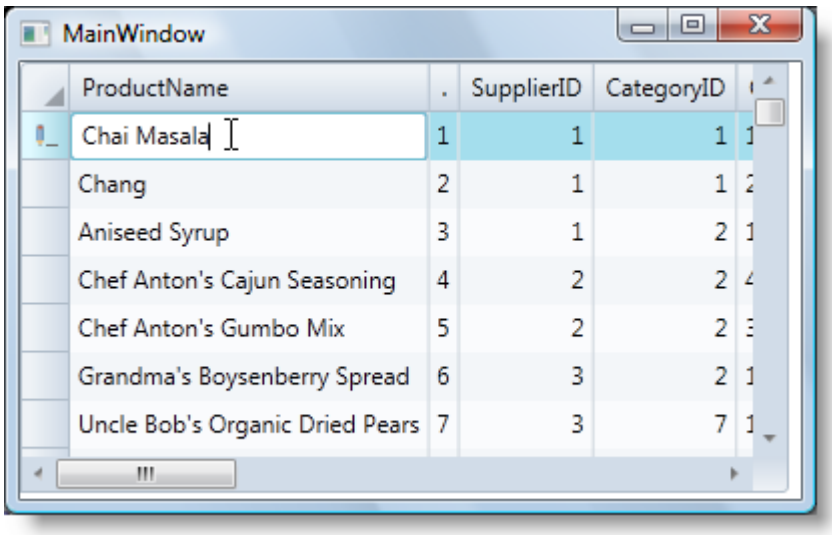
3. Re-order the columns by clicking the ProductName column header and dragging it in front of the ProductID column header. The ProductName column will now appear as the first column in the grid:



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



5. 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.

Class Hierarchy

The following list summarizes the class relationships between the more important classes included in **C1DataGrid**:

- **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.DataGridRowCollection : System.Object**
  Collection of rows.
- **C1.WPF.DataGrid.DataGridCell : System.Object**
  Represents an individual cell.

Creating a C1DataGrid in Expression 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".

Main Concepts and Properties

In order to use the C1DataGrid to create an application that enables you to read and write to most databases it is useful to understand how the main properties map into datagrid elements.

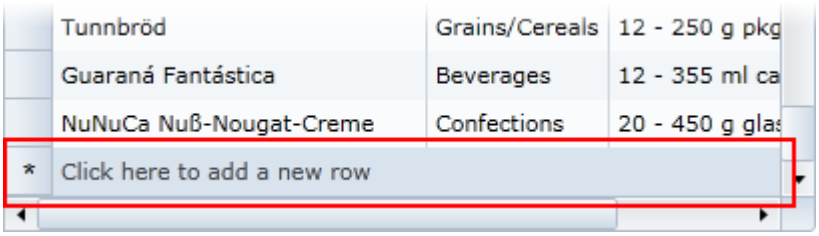
The steps involved in displaying and editing the datagrid are:

1. You can automatically generate the column's headers in the grid by either binding the grid by setting the ItemsSource property to an IEnumberable implementation or by using the **DataGridColumn Collection Editor**.
2. Once the data is set for the grid you can determine whether or not you want to generate the columns automatically or configure the columns explicitly. Set the AutoGenerateColumns property to False to configure the columns explicitly or True to generate the columns automatically.
3. You can then edit (delete, rearrange, and add) the datagrid's columns. For more information see Run-time Interaction.

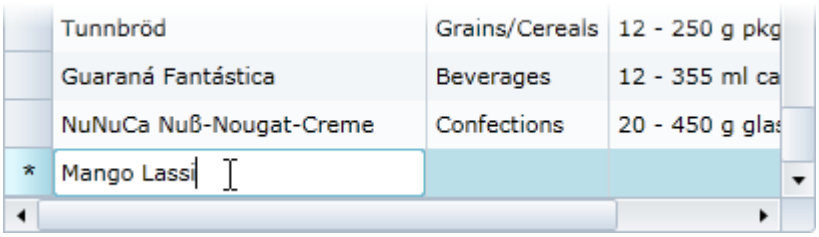
DataGrid Features

Add New Row

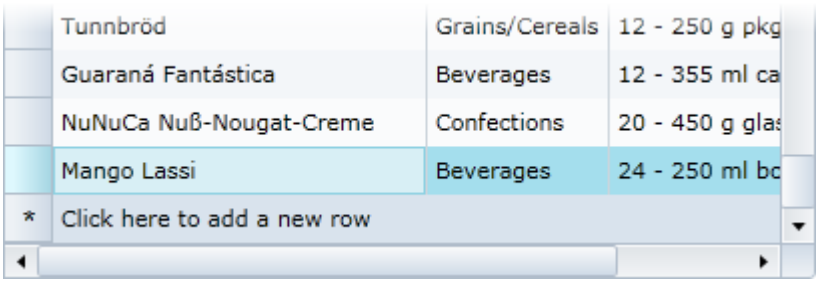
You can add rows to the grid at run time using the new row bar. The new row bar, located at the bottom of the grid by default and indicated by an asterisk symbol (*), allows you to type in new information to add to the grid at run time:



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: The **CanUserAddRows** property must be set to **True** (default) for row adding to be possible. See Disabling Adding Rows for an example.

Disable Adding New 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. If you choose, however, you can disable the new row bar feature by setting the CanUserAddRows property to False.

At Design Time

To disable adding rows, complete the following steps:

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

In XAML

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

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

In Code

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

Visual Basic
Me.C1DataGrid1.CanUserAddRows = False
C#
this.c1DataGrid1.CanUserAddRows = false;

What You've Accomplished

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

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. If you choose, however, you can disable the column and row resizing feature by setting the C1DataGrid.CanUserResizeColumns and C1DataGrid.CanUserResizeRows properties to **False**.

At Design Time

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

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

In XAML

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

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

In Code

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

Visual Basic
`Me.C1DataGrid1.CanUserResizeColumns = False` `Me.C1DataGrid1.CanUserResizeRows = False`

C#
`this.c1DataGrid1.CanUserResizeColumns = false;` `this.c1DataGrid1.CanUserResizeRows = false;`

What You've Accomplished

Run the application and observe that you can no longer resize columns or rows at run time by preforming a drag-

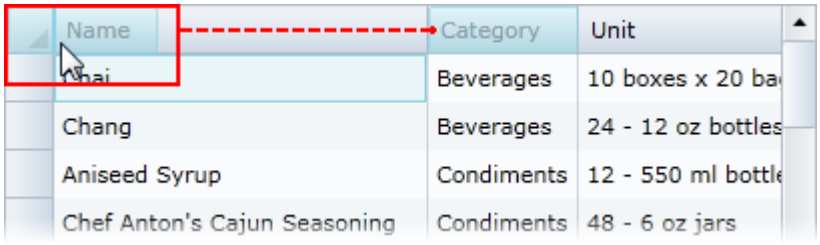
and-drop operation. For more information about column reordering, see the Resizing Columns and Rows topic.

Disabling resizing of rows and columns

Column Reordering

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:



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

Note: The **CanUserReorderColumns** property must be set to True (default) for column reordering to be possible.

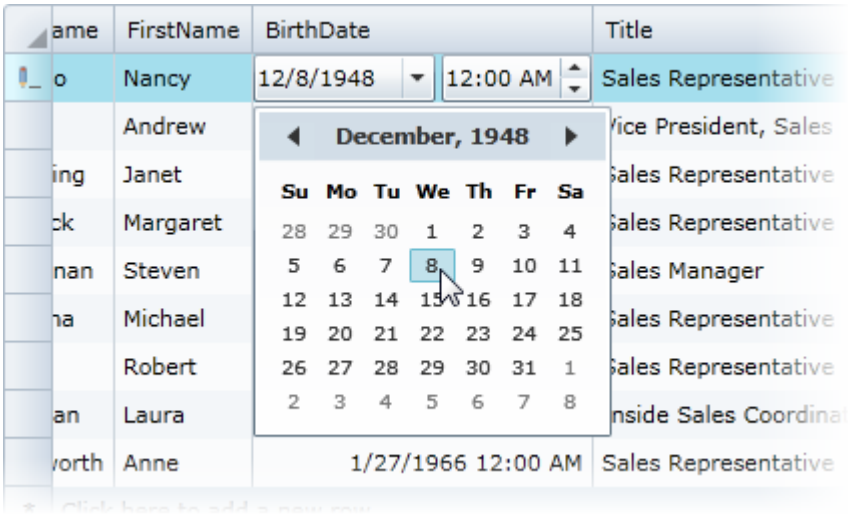
Column Types

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

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

CustomColumns	A custom column. See the **C1DataGrid_Demo** sample for examples 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:



Custom Columns

Customizing Column Cell Content

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

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

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

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

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

Visual Basic

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

```
End Function
```

C#

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

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

Visual Basic

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

C#

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

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

Visual Basic

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

C#

```
public override void BindCellContent(FrameworkElement cellContent, DataGridRow row)
{
```

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

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

Visual Basic

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

C#

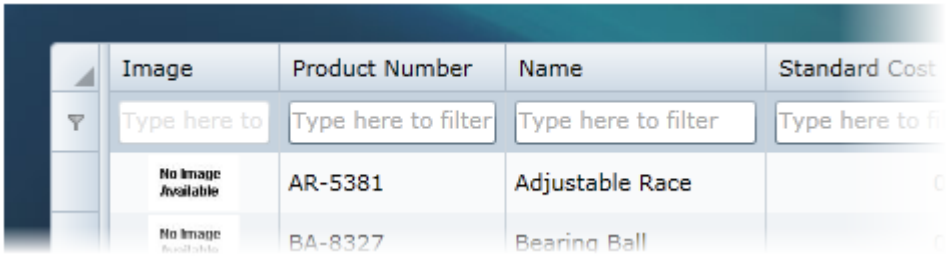
```
Hyperlink.DataContext = row.DataItem;
```

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

Custom Rows

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

```
Imports C1.WPF.DataGrid
Public Class DataGridFilterRow
   Inherits DataGridRow
End Class
```

C#

```
using C1.WPF.DataGrid;
public class DataGridFilterRow : DataGridRow
{

}
```

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

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

Overriding Methods

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

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

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

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

Visual Basic

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

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

C#

```
protected override object GetCellContentRecyclingKey(DataGridColumn column)
{
```

```
    return typeof(TextBox);
}

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

Implementing Filtering

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

1. Add the following code to the BindCellContent method:

Visual Basic

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

C#

```
protected override void BindCellContent(FrameworkElement cellContent,
DataGridColumn column)
{
    var filterTextBox = (TextBox)cellContent;
    //If the column doesn't have a FilterMemberPath specified
    //it won't allow entering text in the TextBox;
    if (string.IsNullOrEmpty(column.FilterMemberPath))
    {
        filterTextBox.IsEnabled = false;
        filterTextBox.Text = "Not available";
    }
    else
    {
        filterTextBox.Text = "";
        filterTextBox.IsEnabled = true;
    }
```

```
    // Handle TextChanged to apply the filter to the column.
    filterTextBox.TextChanged += new EventHandler<TextChangedEventArgs>
(filterTextBox_TextChanged);
}
```

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

Visual Basic

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

C#

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

Adding a Custom Row

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.

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

Visual Basic

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

C#

```
private void C1DataGrid_CreatingRow(object sender, DataGridCreatingRowEventArgs e)
```

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

Adding an Extra Row

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

For example, in XAML or code:

XAML

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

Visual Basic

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

C#

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

Adding Row Details

Each grid row in **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 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 for more information.

Data Binding

The C1DataGrid control can be bound to any object that implements the System.Collections.IEnumerable interface (such as XmlDataProvider, ObjectDataProvider, DataSet, DataView, and so on). You can use the C1DataGrid.ItemsSource property to bind the C1DataGrid.

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

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

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

New Topic 4

Deferred Scrolling

DataGrid for WPF and Silverlight 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. The ScrollMode property accepts the following values from the DataGridScrollMode enumeration.

- RealTime
- Deferred

The example below sets 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:

XAML

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

In Code

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

Visual Basic

```
Me.C1DataGrid1.ScrollMode = DataGridScrollMode.Deferred
```

C#

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

Defining Columns

You can use DataGrid for WPF and Silverlight'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.

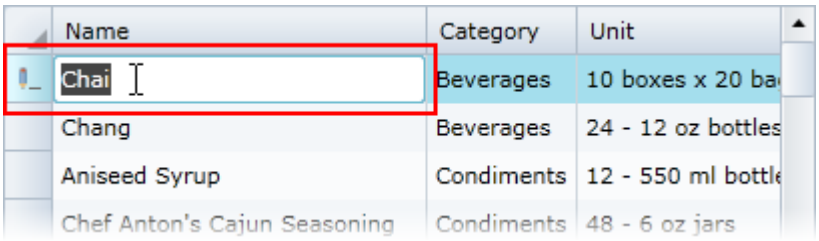
Editing

The following topics describe how to use **C1DataGrid**'s editing features.

Editing Cells

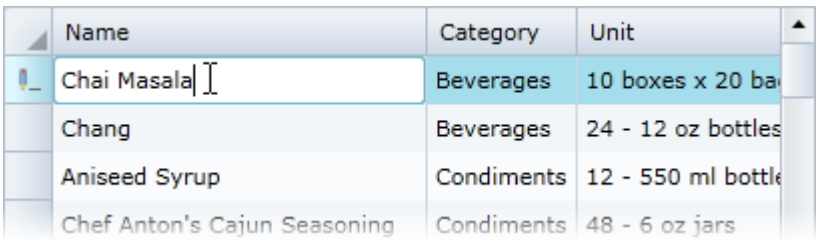
Users can easily edit cell content at run time. Editing content is as simple as selecting a cell and deleting or changing the content in that cell. Complete the following steps to edit cell content:

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



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

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



1. 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** for an example.

Disabling Cell Editing

By default end users edit content in the grid at run time. For more information, see Editing Cells. If you choose, however, you can disable the cell editing feature by setting the **C1DataGrid.CanUserEditRows** property to **False**.

At Design Time

To disable cell editing, complete the following steps:

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

In XAML

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

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

In Code

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

Visual Basic

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

C#

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

What You've Accomplished

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

Locking the Grid

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

In XAML

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

<c1:C1DataGrid x:Name="C1DataGrid1" IsReadOnly="True">

In Code

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

Visual Basic

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

C#

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

Filtering

DataGrid for WPF and Silverlight 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.

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

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

Visual Basic

```
Me.C1DataGrid1.CanUserFilter = True
```

C#

```
this.c1DataGrid1.CanUserFilter = true;
```

See the Filtering Columns topic for more details and examples.

Filtering Columns

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:

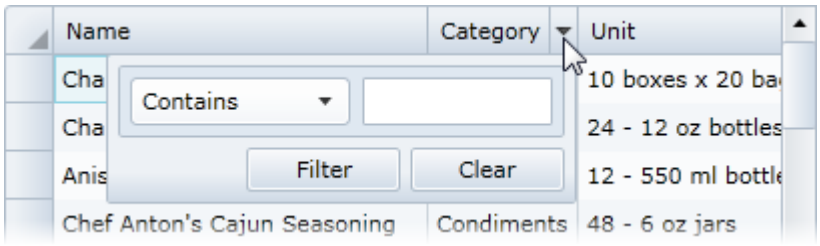
2. Enter the text in the filter text box that you want the column to be filtered by, and click the **Filter** button.

   The column will be sorted.

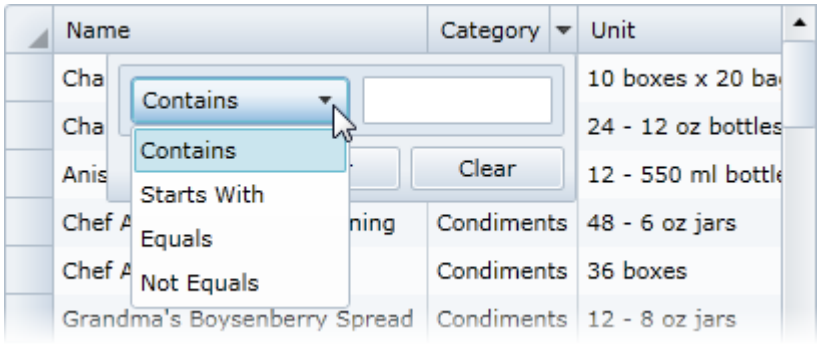
Filter options vary depending on the column type. The following filter options may be included:

- **Text Columns**

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

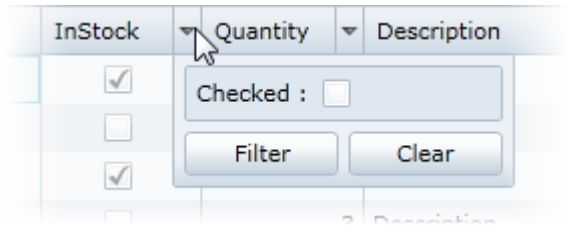


  You can filter the column by whether items in the column contain, start, are equivalent to, or are not equivalent to the filter condition:



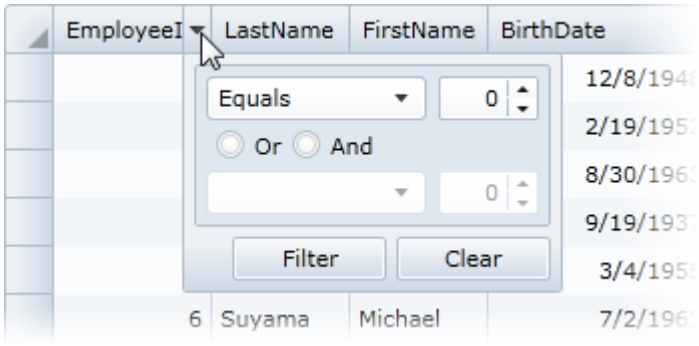
- **Boolean Columns**

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

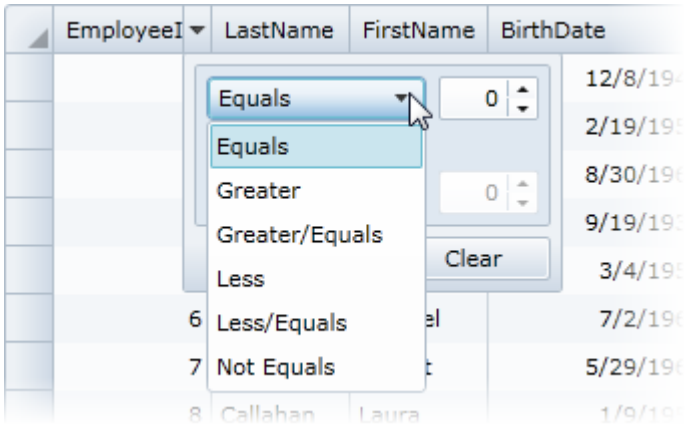


- **Numeric Columns**

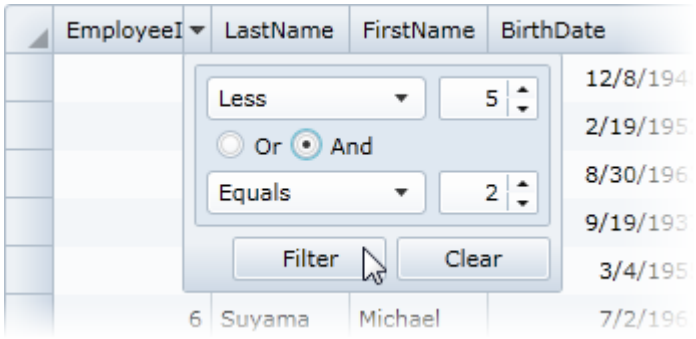
Numeric columns offer several options for filtering:



You can filter the column by specific condition:



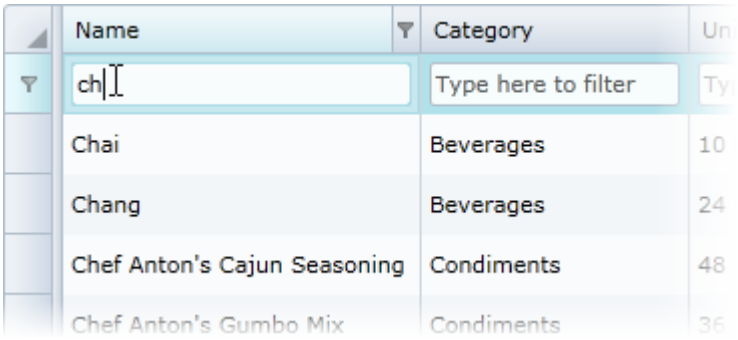
And you can use the **And** and **Or** radio buttons to filter by multiple conditions:



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

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:

XAML

```
<c1:C1DataGrid x:Name="grid" Grid.Row="1" CanUserAddRows="False"
CanUserFreezeColumns="True" FrozenTopRowsCount="1" FrozenBottomRowsCount="1"
RowHeight="30" >
    <c1:C1DataGrid.TopRows>
        < c1:DataGridFilterRow />
    </c1:C1DataGrid.TopRows>
    <c1:C1DataGrid.BottomRows>
        < c1:DataGridFilterRow/>
    </c1:C1DataGrid.BottomRows>
</c1: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:

XAML

```
<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 was 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:

XAML

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

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**:

XAML

```
<c1:DataGridNumericColumn Header="Range filter" Binding="{Binding StandardCost}"
FilterMemberPath="StandardCost">
    < c1:DataGridNumericColumn.Filter>
        < c1:DataGridContentFilter>
            < c1:DataGridFilterList>
                <local:DataGridRangeFilter Minimum="0" Maximum="1000"/>
                < c1:DataGridNumericFilter/>
                < c1:DataGridTextFilter/>
            < /c1:DataGridFilterList>
        < /c1:DataGridContentFilter>
    < /c1:DataGridNumericColumn.Filter>
</c1:DataGridNumericColumn>
```

You can to refer to pre-installed product samples through the following path:

Documents | ComponentOne Samples | Silverlight

Disabling Column Filtering

By default end users can filter columns in the grid at run time. For more information, see Filtering Columns. If you choose, however, you can disable the column filtering feature by setting the C1DataGrid.CanUserFilter property to **False**.

At Design Time

To disable column filtering, complete the following steps:

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

In XAML

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

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

In Code

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

Visual Basic

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

C#

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

What You've Accomplished

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

Tab Filter List

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

XAML

```
<c1:DataGridNumericColumn Header="Filters inside a tab control" Binding="{Binding
StandardCost}" FilterMemberPath="StandardCost">
    < c1:DataGridNumericColumn.Filter>
        < c1:DataGridContentFilter>
            < local:DataGridTabFilters Width="250">
                < local:DataGridRangeFilter Minimum="0" Maximum="1000"/>
                < c1:DataGridNumericFilter/>
                < c1:DataGridTextFilter/>
            < /local:DataGridTabFilters>
        < /c1:DataGridContentFilter>
    < /c1:DataGridNumericColumn.Filter>
</c1:DataGridNumericColumn>
```

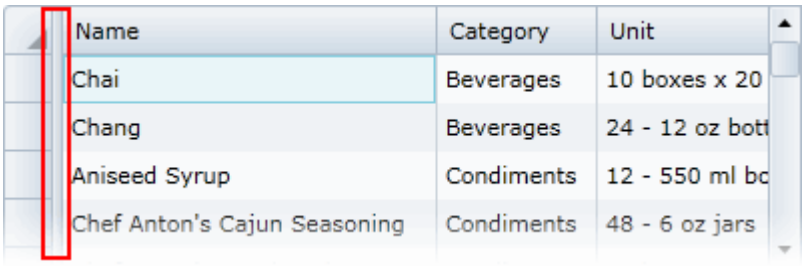
You can refer to pre-installed product samples through the following path:

Documents | ComponentOne Samples | Silverlight

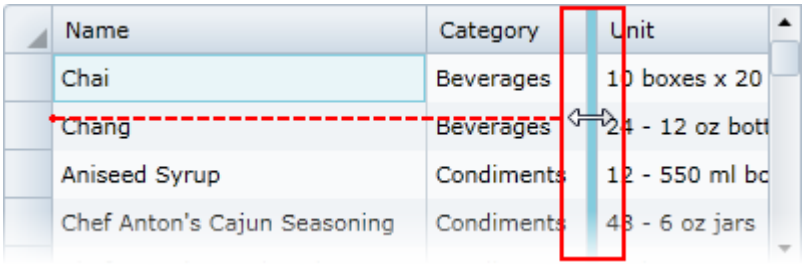
Freezing

Users can freeze columns at run time to prevent them from being scrolled horizontally. This is useful as it keeps

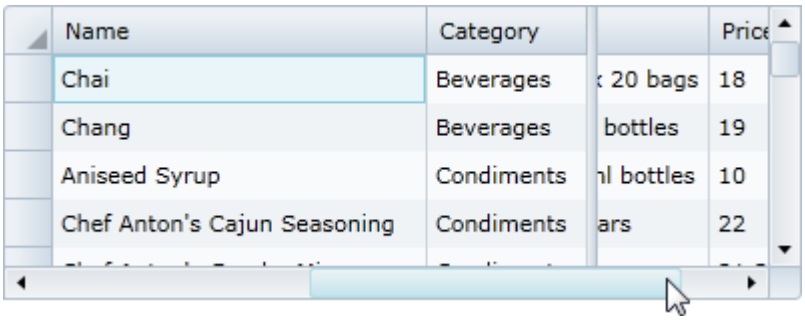
specific columns visible when the grid is resized or scrolled. The freeze bar enables users to freeze columns. When visible, the freeze bar appears to the left of the first columns by default:



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 for an example.

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. This feature is not enabled by default, but if you choose you can enable the column freezing feature by setting the CanUserFreezeColumns property to **Left**.

At Design Time

To enable column freezing, complete the following steps:

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

In XAML

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

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

In Code

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

Visual Basic

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

C#

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

What You've Accomplished

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

Freezing Grid Rows

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

At Design Time

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

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

In XAML

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

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

In Code

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

Visual Basic

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

C#

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

What You've Accomplished

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

Grouping

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. By default, user cannot group columns in the grid but you can enable this function by setting the **C1DataGrid.CanUserGroup** property to **True**.

At Design Time

To enable grouping, complete the following steps:

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

In XAML

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

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

In Code

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

Visual Basic

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

C#

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

What You've Accomplished

Run the application and notice that the grouping area appears at the top of the grid. Note that you can also customize the visibility of the grouping area. For more information about the grouping area, see the Showing the Grouping Area 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. When the CanUserGroup property is set to **True** and grouping is enabled the grouping area is made visible. But if you choose you can show or hide the grouping area whether or not grouping is enabled. By default, the grouping area is not visible when grouping is not enabled but you can make the area visible by setting the ShowGroupingPanel property to **True**.

At Design Time

To show the grouping area, complete the following steps:

1. Click the C1DataGrid control once to select it.

2. Navigate to the Properties window and locate the **ShowGroupingPanel** property.
3. Check the check box next to the **ShowGroupingPanel** property.

In XAML

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

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

In Code

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

Visual Basic

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

C#

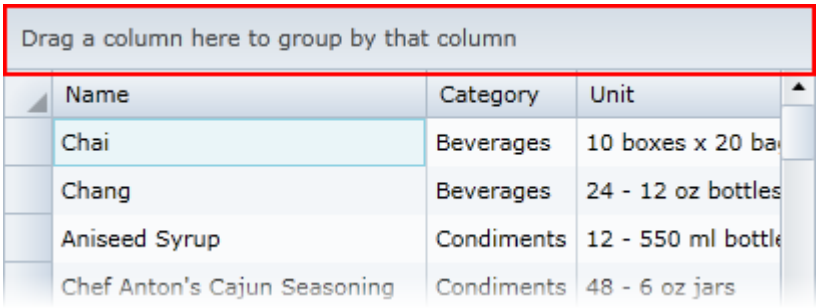
```
this.c1DataGrid1.ShowGroupingPanel = true;
```

What You've Accomplished

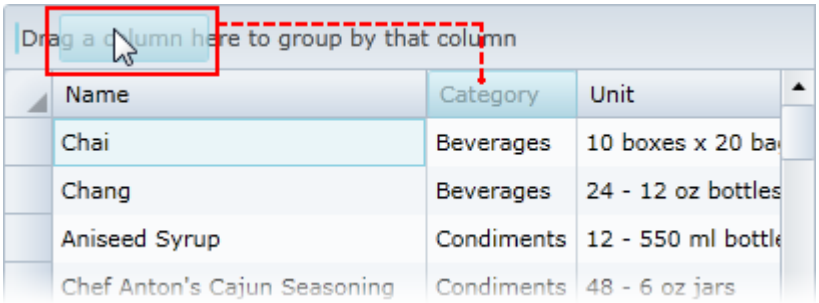
Run the application and notice that the grouping area appears at the top of the grid. Note that even if the grouping area is visible, grouping will not be enabled if the **CanUserGroup** property is **False**. For more information, see the Enabling Grouping in the Grid topic.

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:

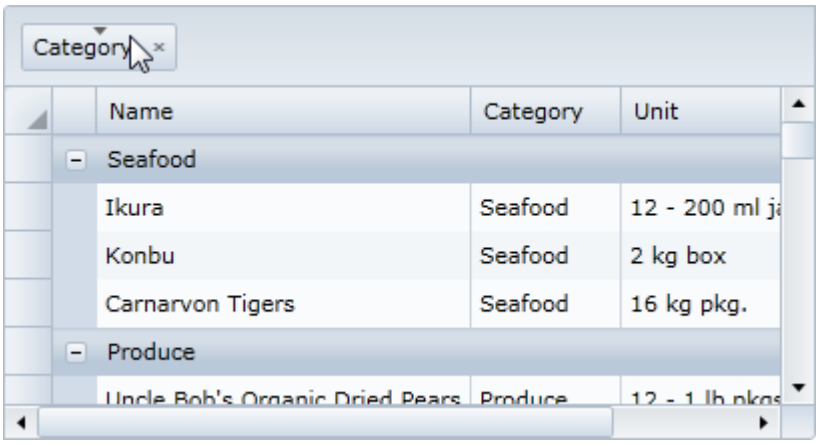


To group a column, drag a column header onto the grouping area:

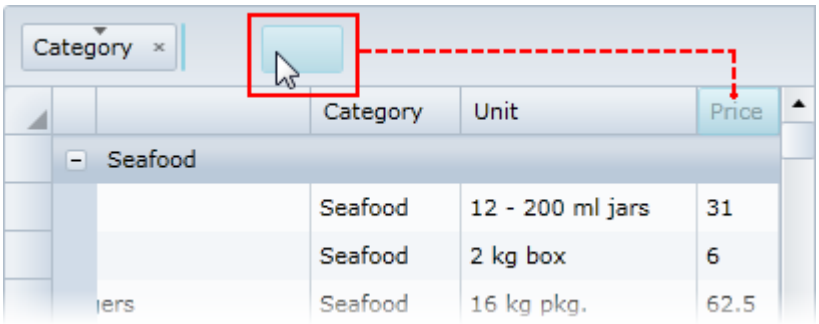


You can sort the display of grouped items, by clicking the column header in the grouping area. In the following image

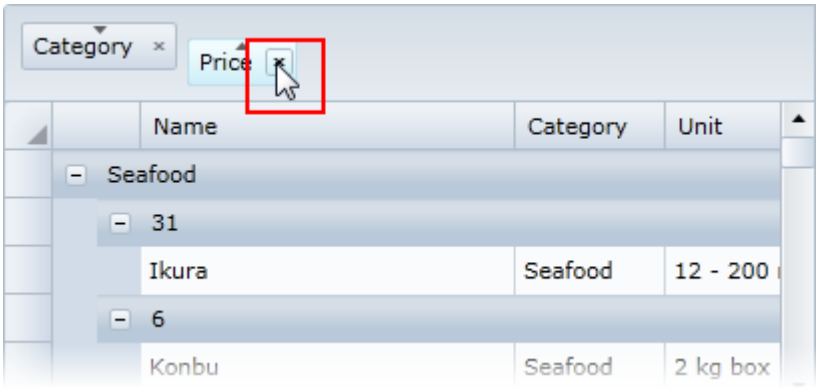
the grouped column has been reverse sorted:



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. For more information about showing the grouping area, see the **Showing the Grouping Area** topic.

Group Summaries

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**)

Keyboard and Mouse Navigation

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

Keyboard Navigation

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

Key Combination	Description
DOWN Arrow	Moves the focus to the cell directly below the current cell. If the focus is in the last row, pressing the DOWN ARROW does nothing.
UP Arrow	Moves the focus to the cell directly above the current cell. If the focus is in the first row, pressing the UP ARROW does nothing.
LEFT Arrow	Moves the focus to the previous cell in the row. If the focus is in the first cell in the row, pressing the LEFT ARROW does nothing.
RIGHT Arrow	Moves the focus to the next cell in the row. If the focus is in the last cell in the row, pressing the RIGHT ARROW does nothing.
HOME	Moves the focus to the first cell in the current row.
END	Moves the focus to the last cell in the current row.
PAGE DOWN	Scrolls the control downward by the number of rows that are displayed. Moves the focus to the last displayed row without changing columns. If the last row is only partially displayed, scrolls the grid to fully display the last row.
PAGE UP	Scrolls the control upward by the number of rows that are displayed. Moves focus to the first displayed row without changing columns. If the first row is only partially displayed, scrolls the grid to fully display the first row.
TAB	If the current cell is in edit mode, moves the focus to the next editable cell in the current row. If the focus is already in the last cell of the row, commits any changes that were made and moves the focus to the first editable cell in the next row. If the focus is in the last cell in the control, moves the focus to the next control in the tab order of the parent container. If the current cell is not in edit mode, moves the focus to the next control in the tab order of the parent container.

SHIFT+TAB	If the current cell is in edit mode, moves the focus to the previous editable cell in the current row. If the focus is already in the first cell of the row, commits any changes that were made and moves the focus to the last cell in the previous row. If the focus is in the first cell in the control, moves the focus to the previous control in the tab order of the parent container. If the current cell is not in edit mode, moves the focus to the previous control in the tab order of the parent container.
CTRL + DOWN ARROW	Moves the focus to the last cell in the current column.
CTRL + UP ARROW	Moves the focust to the last cell in the current column.
CTRL + RIGHT ARROW	Moves the focus to the last cell in the current row.
CTRL + LEFT ARROW	Moves the focus to the first cell in the current row.
CTRL + HOME	Moves the focus to the first cell in the control.
CTRL + PAGE DOWN	Same as PAGE DOWN.
CTRL + PAGE UP	Same as PAGE UP.
ENTER	Shifts the focus to the next row.
F2	Enter/exit edit mode on a selected cell (if the grid and column's **IsReadOnly** properties are **False**).
ESC	Cancel editing of a cell or new row.
DEL	Delete selected row.
INSERT	Scrolls to the new row and begins editing it.

Mouse Navigation

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

Mouse Action	Description
Click an unselected row	Makes the clicked row the current row.
Click a cell in the current row	Puts the clicked cell into edit mode.
Drag a column header cell	Moves the column so that it can be dropped into a new position (if the **CanUserReorderColumns** property is **True** and the current column's **CanUserRorder** property is **True**).
Drag a column header separator	Resizes the column (if the **CanUserResizeColumns** property is **True** and the **CanUserResize** property is True for the current column).
Click a column header cell	If the property **ColumnHeaderClickAction** is set to **Sort**, when the user clicks the column header it sorts the column (if the **CanUserSortColumns** property is **True** and the **CanUserSort** property is **True** for the current column). Clicking the header of a column that is already sorted will reverse the sort direction of that column.

	Pressing the CTRL key while clicking multiple column headers will sort by multiple columns in the order clicked. If the property **ColumnHeaderClickAction** is set to **Select** the column will be selected if **SelectionMode** supports column selection.
CTRL + click a row	Modifies a non-contiguous multi-row selection (if **SelectionMode** support multiple rows, cells, or columns).
SHIFT + click a row	Modifies a contiguous multi-row selection (if **SelectionMode** support multiple rows, cells, or columns).

Multiple Row Selection

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

Selection Keys

The following selection keys modify multi-row selection:

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

Mouse Selection

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

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

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

Custom Keyboard Navigation

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

Adding the KeyDown Event Handler

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

1. Switch to Code view and add an event handler for the KeyDown event, for example:
2. Switch to Source view and add the event handler to instances of the C1DataGrid control, for example:

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

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

Localizing the Application

You can localize (translate) end user visible strings in **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.
2. Update your project file's supported cultures. See Adding Supported Cultures.
3. And, depending on your project, set the current culture. See Setting the Current Culture.

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:

String	Default Value	Description
AddNewRow	Click here to add a new row	Text that appears in the add new row.

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

		filter condition should apply to items that start with the value of the condition.

Adding Supported Cultures

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

1. In the Solution Explorer, right-click your project and select **Unload Project**. The project will appear grayed out and unavailable.
2. Right click the project again, and select the **Edit ProjectName.csproj** option (or **Edit ProjectName.vbproj**, where *ProjectName* is the name of your project).
3. In the .csproj file, locate the *<SupportedCultures></SupportedCultures>* tags. In between the tags, list the cultures you want to be supported, separating each with a semicolon.

   For example:

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

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

4. Save and close the .csproj or .vbproj file.
5. In the Solution Explorer, right-click your project and choose **Reload Project** from the content menu.

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

Setting the Current Culture

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

Visual Basic

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

C#

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

```
    // InitializeComponent() call.
    InitializeComponent();
    // Add any initialization after the InitializeComponent() call.
}
```

Row Details

Row Details Template

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

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

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

XAML

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

Disabling Row Details Toggling

When the grid includes a child grid or you've created a master-detail grid, by default the row details can be toggled so that they are visible or collapsed. If you choose, however, you can disable the toggling the details row feature by

setting the CanUserToggleDetails property to **False**. Note that you will need to have a grid with row details to view the change in this example.

At Design Time

To disable toggling row details, complete the following steps:

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

In XAML

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

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

In Code

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

Visaul Basic

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

C#

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

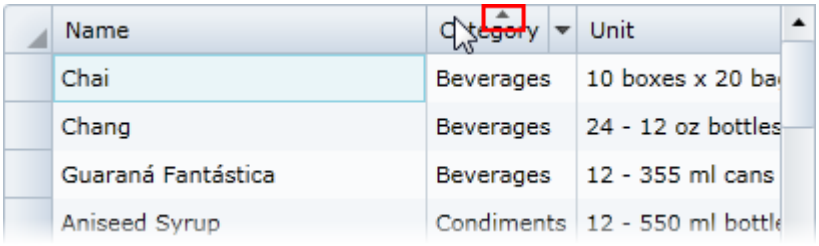
What You've Accomplished

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

Sorting

Sorting grid columns at run time is simple in **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.

Disabling Column Sorting

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

At Design Time

To disable column sorting, complete the following steps:

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

In XAML

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

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

In Code

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

Visual Basic

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

C#

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

What You've Accomplished

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

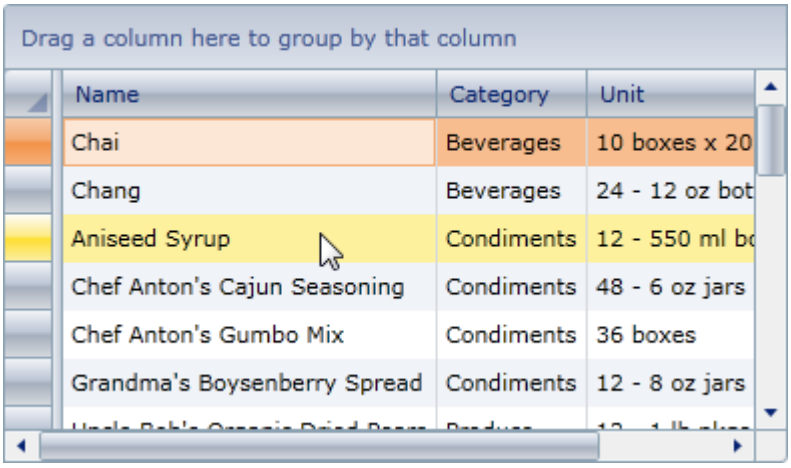
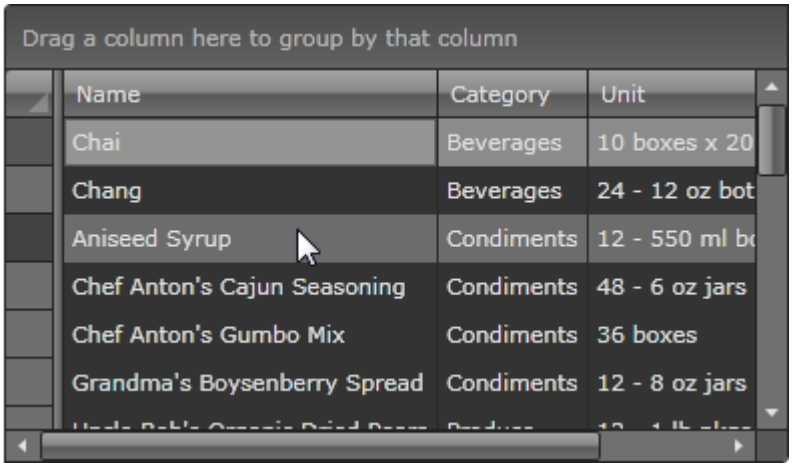
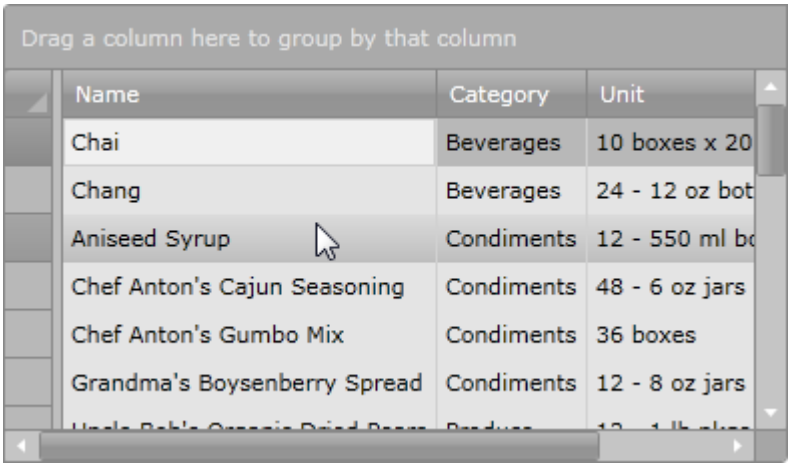
DataGrid Appearance

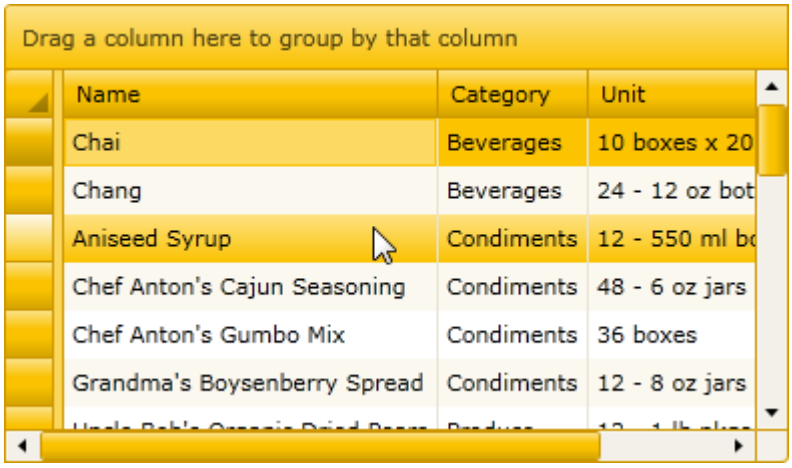
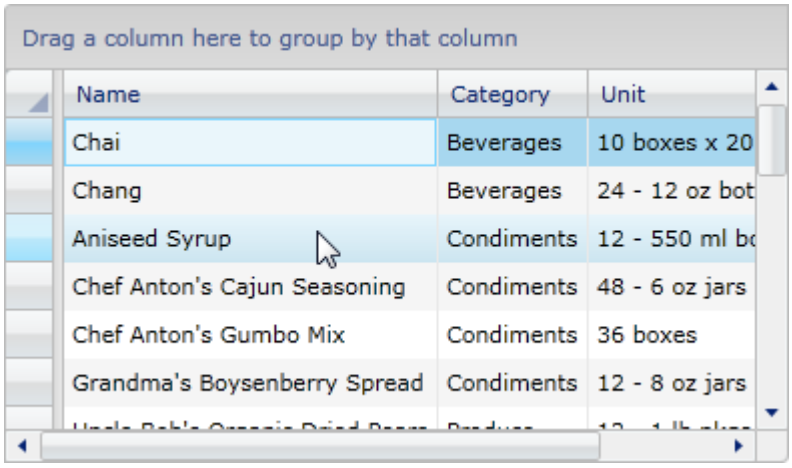
The following topics describe the C1DataGrid appearance for WPF and Silverlight.

C1DataGrid Themes

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:

Theme Name	Theme Preview
C1ThemeBureauBlack	
C1ThemeExpressionDark	
C1ThemeExpressionLight	

C1ThemeRainierOrange	
C1ThemeShinyBlue	
C1ThemeWhistlerBlue	

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 **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.

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

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

Table Formatting Options

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

Setting Row and Column Header Visibility

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

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

Setting Grid Line Visibility

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

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

Setting New Row Visibility

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

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

Setting Vertical and Horizontal Scrollbar Visibility

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

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

Setting Row Details Visibility

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

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

C1DataGrid Brushes

DataGrid for WPF 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:

Theme Name	Theme Preview
Background	Gets or sets the background brush that is used when rendering. (This brush will be applied to all the parts of the data grid)
Foreground	Gets or sets the foreground brush that is used when rendering. (This brush will be applied to all the parts of the data grid)
BorderBrush	Gets or sets the border brush that is

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

DataGrid for WPF uses ClearStyle technology for styling. For details, see C1DataGrid ClearStyle.

C1DataGrid Clear Style

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

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

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

The grid will appear similar to the following image:

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

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

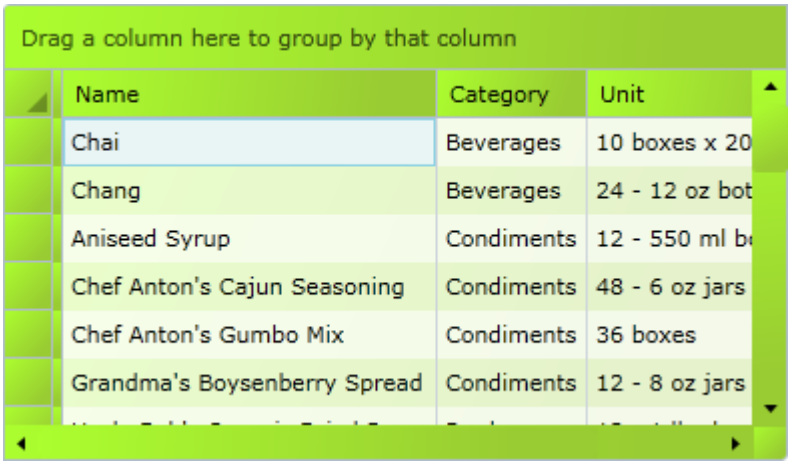
The grid will appear similar to the following image:

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

XAML

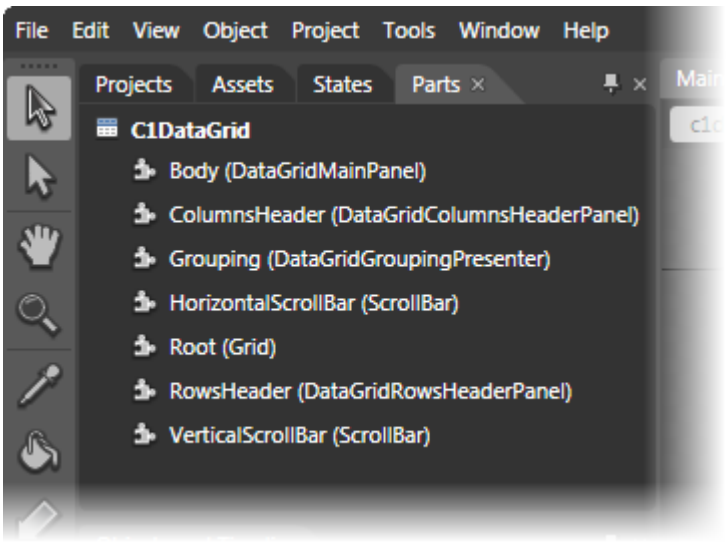
```
<c1:C1DataGrid x:Name="c1DataGrid1" HorizontalAlignment="Left" Margin="10,10,0,0"
VerticalAlignment="Top" CanUserFreezeColumns="Left" CanUserGroup="True">
   <c1:C1DataGrid.Background>
       <LinearGradientBrush StartPoint="0,0" EndPoint="1,1">
           <GradientStop Color="GreenYellow" Offset="0.0"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

The grid will appear similar to the following image:



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:

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

Name	Type	Description
Body	DataGridMainPanel	Panel that contains the body of the grid.
ColumnsHeader	DataGridColumnsHeaderPanel	Panel that contains a collection of DataGridColumnsHeaderPanel.
Grouping	DataGridGroupingPresenter	Presenter that displays the grouping panel or another element

		if there is no columns in the grouping panel.
HorizontalScrollBar	ScrollBar	Represents a control that provides a scroll bar that has a sliding Thumb whose position corresponds to a value.
Root	Grid	Definies a flexible grid area that consists of columns and rows.
RowsHeader	DataGridRowsHeaderPanel	Panel that contains DataGridRowsHeaderPanel.
VerticalScrollBar	ScrollBar	Represents a control that provides a scroll bar that has a sliding Thumb whose position corresponds to a value.

Customizing Grid Appearance

The following topics detail how you can customize **C1DataGrid** by changing the grid's appearance. **DataGrid for WPF and Silverlight** 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 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

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 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 to the <c1:C1DataGrid> tag so that it appears similar to the following:

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

In Code

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

Visual Basic

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

C#

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

```
SolidColorBrush(Colors.White);
```

What You've Accomplished

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

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 for more details.

Removing the Grid's Alternating Row Colors

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:

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

In Code

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

Visual Basic

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

C#

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

What You've Accomplished

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

Changing the Grid's Mouse Hover Style

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

At Design Time

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

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

In XAML

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

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

In Code

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

Visual Basic

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

C#

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

What You've Accomplished

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

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:

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

In Code

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

Visual Basic

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

C#

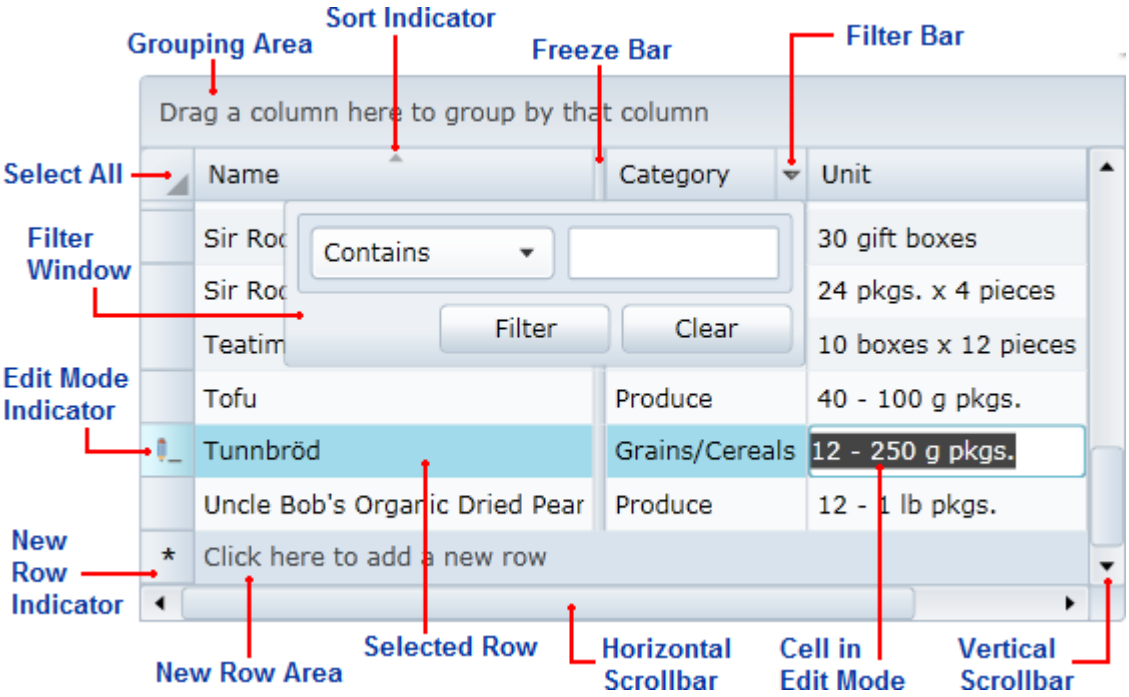
```
this.c1datagrid1.FontFamily = new FontFamily("Times New Roman");
this.c1datagrid1.FontSize = 10;
```

What You've Accomplished

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

Run Time Interation

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



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

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:

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

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

Customizing Automatically Generated Columns

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

Adding the AutoGeneratingColumn Event Handler

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

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

Visual Basic

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

C#

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

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

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

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

Canceling Column Generation

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

Visual Basic

```
<c1:C1DataGrid x:Name="c1DataGrid1" AutoGenerateColumns="True" AutoGeneratingColumn="
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;
}
```

DataGrid 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 Studio.

You can refer to pre-installed product samples through the following path

Documents | ComponentOne | WPF.

Several pages in the sample installed with **WPF Edition** details 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.

Tutorials

New Topic 5

Step 1 of 3: Creating the User Interface

In this step you'll begin in Visual Studio to create a Silverlight grid application. You'll then continue by creating and customizing the application's user interface (UI) and adding the **C1DataGrid** control to your project.

To set up your project, complete the following steps:

1. In Visual Studio, select **File | New | Project**.

2. In the **New Project** dialog box, select a language in the left pane and in the templates list select **Silverlight Application**. Enter a **Name** for your project "StealthPaging", and click **OK**. The **New Silverlight Application** dialog box will appear.

3. Click **OK** to accept the default settings, close the **New Silverlight Application** dialog box, and create your project.

4. Navigate to the Solution Explorer, right-click the **StealthPaging** project, and select **Add Reference** from the context menu.

5. In the **Add Reference** dialog box locate the **System.Runtime.Serialization** assembly and click the **OK** button to add a reference to your project. The dialog box will close and the reference will be added.

6. If the **MainPage.xaml** file is not currently open, navigate to the Solution Explorer and double-click on the **MainPage.xaml** item.

7. In the XAML view, place the cursor just after the <Grid x:Name="LayoutRoot" Background="White"> tag and add the following markup:

```
<!-- Grid Layout-->
<Grid.RowDefinitions>
   <RowDefinition Height="*" />
   <RowDefinition Height="Auto" />
</Grid.RowDefinitions>
```

This row definition will define the layout of your application.

8. In the XAML window of the project, place the cursor just above the </Grid> tag and click once.

9. Navigate to the Toolbox and double-click the **C1DataGrid** icon to add the grid control to **MainPage.xaml**. The XAML markup will now look similar to the following:

```
<UserControl x:Class="StealthPaging.MainPage"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
   xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
   mc:Ignorable="d"
   d:DesignHeight="300" d:DesignWidth="400"
xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml">
   <Grid x:Name="LayoutRoot" Background="White">
      <!-- Grid Layout-->
      <Grid.RowDefinitions>
         <RowDefinition Height="*" />
         <RowDefinition Height="Auto" />
      </Grid.RowDefinitions>
      <c1:C1DataGrid />
   </Grid>
</UserControl>
```

Note that the **C1.Silverlight.DataGrid** namespace and <c1:C1DataGrid /> tag has been added to the project.

10. If the <c1:C1DataGrid> tag includes existing content, delete it so it appears similar to the following:

```
<c1:C1DataGrid />
```

11. Give your grid a name by adding x:Name="peopleDataGrid" to the <c1:C1DataGrid> tag so that it appears similar to the following:

```
<c1:C1DataGrid x:Name="peopleDataGrid" />
```

By giving the control a unique identifier, you'll be able to access the **C1DataGrid** control in code.

12. Customize your grid by adding AutoGenerateColumns="True" CanUserAddRows="False" to the <c1:C1DataGrid> tag so that it appears similar to the following:

```
<c1:C1DataGrid x:Name="peopleDataGrid" AutoGenerateColumns="True" CanUserAddRows="False" />
```

This markup will set the grid to generate columns automatically and will disable adding new rows.

13. Add the following markup just after the </c1:C1DataGrid> tag:

```
<TextBlock x:Name="txtStatus" Grid.Row="1" Text="Ready."  Margin="0,5,0,0" />
```

This **TextBlock** will be used to display status information text.

What You've Accomplished

If you run your application you'll observe that your page includes a grid and text below the grid. You've successfully created a basic grid application, but the grid is blank and contains no data. In the next steps you'll bind the grid to a data source and add stealth paging in code.

Step 2 of 3: Adding a Web Service

In this step you'll add a data source to your project, and begin the process of binding the grid.

To set up your project, complete the following steps:

1. Navigate to the Solution Explorer, right-click the **StealthPaging.Web** project, and select **Add Reference** from the context menu.

1. In the **Add Reference** dialog box locate the **System.Runtime.Serialization** assembly and click the **OK** button to add a reference to your project. The dialog box will close and the reference will be added.

2. In the Solution Explorer right-click the **StealthPaging.Web** project, and select **Add | New Item**.

3. In the left pane of the **Add New Item** dialog box, select the **Web** item.

4. In the templates list, select **Web Service**, name the Web Service "DataWebService.asmx", and click the **Add** button. Note that the Web Service file will be added to your project and automatically opened.

5. In the **DataWebService.asmx** file, add the following using statements at the top of the file:

- Visual Basic

```
Imports System.Runtime.Serialization
```

- C#

```
using System.Runtime.Serialization;
```

6. In the **DataWebService.asmx** file, replace the code in the **StealthPaging.Web** namespace with the following::

- Visual Basic

```
' To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the following line.
' <System.Web.Script.Services.ScriptService()> _
<System.Web.Services.WebService(Namespace:="http://tempuri.org/")> _
<System.Web.Services.WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1)> _
<ToolboxItem(False)> _
```

```
Public Class DataWebService
    Inherits System.Web.Services.WebService
    <WebMethod()> _
        Public Function GetData(startRow As Integer, endRow As Integer) As List(Of ServerPerson)
            Dim personList As New List(Of ServerPerson)()
            For i As Integer = startRow To endRow - 1
                personList.Add(New ServerPerson() With { _
                    .FirstName = String.Format("First Name {0}", i), _
                    .LastName = String.Format("Last Name {0}", i), _
                    .Age = i, _
                    .City = String.Format("City {0}", i) _
                })
            Next
            Return personList
        End Function
    End Class
    <DataContract> _
    Public Class ServerPerson
        Private _firstName As String
        <DataMember> _
        Public Property FirstName() As String
            Get
                Return _firstName
            End Get
            Set
                _firstName = value
            End Set
        End Property
        Private _lastName As String
        <DataMember> _
        Public Property LastName() As String
            Get
                Return _lastName
            End Get
            Set
                _lastName = value
            End Set
        End Property
        Private _age As Integer
```

```
        <DataMember> _
        Public Property Age() As Integer
            Get
                Return _age
            End Get
            Set
                _age = value
            End Set
        End Property
        Private _city As String
        <DataMember> _
        Public Property City() As String
            Get
                Return _city
            End Get
            Set
                _city = value
            End Set
        End Property
    End Class
```

- C#

```
namespace StealthPaging.Web
{
    /// <summary>
    /// Summary description for DataWebService
    /// </summary>
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [System.ComponentModel.ToolboxItem(false)]
    // To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the following line.
    // [System.Web.Script.Services.ScriptService]
    public class DataWebService : System.Web.Services.WebService
    {
        [WebMethod]
        public List<ServerPerson> GetData(int startRow, int endRow)
        {
            List<ServerPerson> personList = new List<ServerPerson>();
            for (int i = startRow; i < endRow; i++)
```

```
        {
            personList.Add(new ServerPerson()
            {
                FirstName = string.Format("First Name {0}", i),
                LastName = string.Format("Last Name {0}", i),
                Age = i,
                City = string.Format("City {0}", i)
            });
        }
        return personList;
    }
}
[DataContract]
public class ServerPerson
{
    private string _firstName;
    [DataMember]
    public string FirstName
    {
        get { return _firstName; }
        set { _firstName = value; }
    }
    private string _lastName;
    [DataMember]
    public string LastName
    {
        get { return _lastName; }
        set { _lastName = value; }
    }
    private int _age;
    [DataMember]
    public int Age
    {
        get { return _age; }
        set { _age = value; }
    }
    private string _city;
    [DataMember]
    public string City
```

```
        {
            get { return _city; }
            set { _city = value; }
        }
    }
}
```

This code will create a new list that will be used to populate the C1DataGrid control.

7. Save your application, right-click the **StealthPaging.Web** project, and select **Build** from the context menu. Note that you'll now be done with the **StealthPaging.Web** project and will return to working with the **StealthPaging** project.

What You've Accomplished

In this step you've added a data source to your project and created a Web Service. In the next step you'll finish connecting the Web Service to your project and you'll run your application.

Step 3 of 3: Connecting the Web Service

In the previous step you created a Web Service and added a database to your project. In this step you'll continue by linking the Web Service to your application. Note that this step requires **ComponentOne Data for Silverlight**.

To set up your project, complete the following steps:

1. In the Solution Explorer, expand the project's node, right-click the project name (for example ComponentOneDataGrid) and select **Add Reference** from the context menu.

1. In the **Add Reference** dialog box, add a reference to the **C1.Silverlight.Data** assembly and click **OK**.

2. In the Solution Explorer, right-click the project name and select **Add Service Reference** from the context menu.

3. In the **Add Service Reference** dialog box click the **Discover** button. The DataService.asmx file will appear in the list of Services.

4. In the **Namespace** text box, change the default value to "DataService" and click the **OK** button to save your settings and close the dialog box.

5. In the Solution Explorer, expand the **MainPage.xaml** node and double-click the **MainPage.xaml.cs** or **MainPage.xaml.vb** file to open it in the Code Editor.

6. Add the following import statements at the top of the file:

- Visual Basic

```
Imports System.IO
Imports C1.Silverlight.Data
Imports ComponentOneDataGrid.DataService ' ComponentOneDataGrid is the project's namespace, change this if the name of your project is different.
```

- C#

```
using System.IO;
using C1.Silverlight.Data;
using ComponentOneDataGrid.DataService; // ComponentOneDataGrid is the project's namespace, change this if the name of your project is different.
```

7. Add LoadData(); to the **MainPage** constructor so it appears like the following:

- Visual Basic

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

- C#

```
public MainPage()
{
    InitializeComponent();
    LoadData();
}
```

8. Add the **LoadData** and **svc_GetDataCompleted** methods to retrieve data from the Web Service:

- Visual Basic

```
Private _ds As DataSet = Nothing
Private Sub LoadData()
  ' Invoke Web Service
  Dim svc = GetDataService()
  AddHandler svc.GetDataCompleted, AddressOf svc_GetDataCompleted
  'svc.GetDataAsync("Categories,Products,Employees");
  svc.GetDataAsync("Employees")
End Sub
Private Sub svc_GetDataCompleted(sender As Object, e As GetDataCompletedEventArgs)
  ' Handle errors
  If e.[Error] IsNot Nothing Then
      _tbStatus.Text = "Error downloading data..."
      Return
  End If
  ' Parse data stream from server (DataSet as XML)
  _tbStatus.Text = String.Format("Got data, {0:n0} kBytes", e.Result.Length / 1024)
  Dim ms = New MemoryStream(e.Result)
  _ds = New DataSet()
  _ds.ReadXml(ms)
  ' Bind control to the data
  BindData()
End Sub
```

- C#

```
DataSet _ds = null;
void LoadData()
{
  // Invoke Web Service
```

```
    var svc = GetDataService();
    svc.GetDataCompleted += svc_GetDataCompleted;
    //svc.GetDataAsync("Categories,Products,Employees");
    svc.GetDataAsync("Employees");
}
void svc_GetDataCompleted(object sender, GetDataCompletedEventArgs e)
{
    // Handle errors
    if (e.Error != null)
    {
        _tbStatus.Text = "Error downloading data...";
        return;
    }
    // Parse data stream from server (DataSet as XML)
    _tbStatus.Text = string.Format("Got data, {0:n0} kBytes", e.Result.Length / 1024);
    var ms = new MemoryStream(e.Result);
    _ds = new DataSet();
    _ds.ReadXml(ms);
    // Bind control to the data
    BindData();
}
```

9. Implement the **GetDataService()** method by adding the following code:

- Visual Basic

```
' Get data service relative to current host/domain
Private Function GetDataService() As DataServiceSoapClient
  ' Increase buffer size
  Dim binding = New System.ServiceModel.BasicHttpBinding()
  binding.MaxReceivedMessageSize = 2147483647
  ' int.MaxValue
  binding.MaxBufferSize = 2147483647
  ' int.MaxValue
  ' Get absolute service address
  Dim uri As Uri = C1.Silverlight.Extensions.GetAbsoluteUri("DataService.asmx")
  Dim address = New System.ServiceModel.EndpointAddress(uri)
  ' Return new service client
  Return New DataServiceSoapClient(binding, address)
End Function
```

- C#

```
// Get data service relative to current host/domain
DataServiceSoapClient GetDataService()
{
   // Increase buffer size
   var binding = new System.ServiceModel.BasicHttpBinding();
   binding.MaxReceivedMessageSize = 2147483647;
   // int.MaxValue
   binding.MaxBufferSize = 2147483647;
   // int.MaxValue
   // Get absolute service address
   Uri uri = C1.Silverlight.Extensions.GetAbsoluteUri("DataService.asmx");
   var address = new System.ServiceModel.EndpointAddress(uri);
   // Return new service client
   return new DataServiceSoapClient(binding, address);
}
```

10. Implement the **BindData()** method by adding the following code:

- Visual Basic

```
Private Sub BindData()
  ' Get the tables
  Dim dtEmployees As DataTable = _ds.Tables("Employees")
  ' Populate categories grid
  _c1DataGrid.ItemsSource = dtEmployees.DefaultView
End Sub
```

- C#

```
void BindData()
{
   // Get the tables
   DataTable dtEmployees = _ds.Tables["Employees"];
   // Populate categories grid
   _c1DataGrid.ItemsSource = dtEmployees.DefaultView;
}
```

11. Run your application and observe that the grid appears bound to the *Employees* table of the Northwind database:

What You've Accomplished

Congratulations, you've completed this tutorial! In this tutorial you created a new Silverlight project, added an Access database, and created a Web Service to bind the **C1DataGrid** control.

New Topic 6

Creating a Master Detail View

The following tutorial will walk you through using the **C1DataGrid** control to present data in a master/detail view using the row details feature.

The following example shows a set of product categories loaded from a XML file using LINQ to XML. For each row in the main grid (categories), a list of products is loaded and shown in the detail view using a second **C1DataGrid** control. The detail data is loaded when the detail view of a category row changes.

For more information, see the **C1DataGrid_Demo** sample installed with **Silverlight Edition**.

Step 1 of 3: Setting up the Master/Detail Grid

In this step you'll begin in Visual Studio to create a Silverlight grid application using **DataGrid for Silverlight**. You'll create a new Silverlight project and add the **C1DataGrid** control to your application.

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

1. In Visual Studio, select **File | New | Project**.

2. In the **New Project** dialog box, select a language in the left pane (in this example, **C#** is used), and in the templates list select **Silverlight Application**. Enter "MasterDetail" in the **Name** text box, and click **OK**. The **New Silverlight Application** dialog box will appear.

3. Click **OK** to accept the default settings, close the **New Silverlight Application** dialog box, and create your project.

4. In the XAML window of the project, place the cursor between the <Grid> and </Grid> tags and click once.

5. Navigate to the Toolbox and double-click the **C1DataGrid** icon to add the grid control to **MainPage.xaml**. The XAML markup will now look similar to the following:

```
<UserControl xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
x:Class="MasterDetail.MainPage"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006" mc:Ignorable="d"
d:DesignWidth="640" d:DesignHeight="480">
  <Grid x:Name="LayoutRoot">
    <c1:C1DataGrid></c1:C1DataGrid>
  </Grid>
</UserControl>
```

6. If the <c1:C1DataGrid> tag includes existing content, delete it so it appears similar to the following:

```
<c1:C1DataGrid></c1:C1DataGrid>
```

7. Give your grid a name by adding x:Name="c1dg" to the <c1:C1DataGrid> tag so that it appears similar to the following:

```
<c1:C1DataGrid x:Name="c1dg">
```

By giving the control a unique identifier, you'll be able to access the **C1DataGrid** control in code.

8. Add CanUserAddRows="False" to the <c1:C1DataGrid> tag so that it appears similar to the following:

```
<c1:C1DataGrid x:Name="c1dg" CanUserAddRows="False">
```

Users will not be able to add new rows to the grid.

9. Add Margin="5" to the <c1:C1DataGrid> tag so that it appears similar to the following:

```
<c1:C1DataGrid x:Name="c1dg" CanUserAddRows="False" Margin="5">
```

This will add a margin around the grid.

What You've Accomplished

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

Step 2 of 3: Adding a Data Source to the Project

In this step you'll add a data source to your application and add external files to set up the data source. Note that to simplify the tutorial, this step uses files included with the **C1DataGrid_Demo** sample included with **Silverlight Edition.** You can refer to pre-installed product samples through the following path:

Documents | ComponentOne Samples | Silverlight

To add a data source, complete the following steps:

1. In the Solution Explorer window, right-click the **MasterDetail** project and select **Add | New Folder**. Rename the folder "Resources".

2. In the Solution Explorer window, right-click the **Resources** folder and select **Add | Existing Item**.

3. In the **Add Existing Item** dialog box, navigate to the **C1DataGrid_Demo\Resources** sample folder, select the **products.xml** file, and click **Add**. This file provides that data you'll use in the project.

4. Select the **products.xml** file in the Solution Explorer, and in the Properties window set its **Build Action** property to **Embedded Resource**.

5. In the Solution Explorer window, right-click the **MasterDetail** project and select **Add | Existing Item**.

6. In the **Add Existing Item** dialog box, navigate to the **C1DataGrid_Demo** sample folder, select the **Data.cs** file, and click **Add**. This file contains code to set up the data source.

What You've Accomplished

In this step you added an XML data source. In the next step, you'll set up the row details section and finalize the application.

Step 3 of 3: Setting up Row Details

In this step you'll finish setting up the row details section of the grid. You'll add a **RowDetailsTemplate** to set the appearance of the details row, and you'll add code to set up the details row behavior.

To set up row details, complete the following steps:

1. Add the following <c1:C1DataGrid.RowDetailsTemplate> between the <c1:C1DataGrid></c1:C1DataGrid> tags so that it appears similar to the following:

```
<c1:C1DataGrid x:Name="c1dg" CanUserAddRows="False" Margin="5">
  <c1:C1DataGrid.RowDetailsTemplate>
    <DataTemplate>
      <c1:C1DataGrid HeadersVisibility="Column" Margin="5" CanUserAddRows="False"/>
    </DataTemplate>
  </c1:C1DataGrid.RowDetailsTemplate>
</c1:C1DataGrid>
```

This template will customize the row details section display.

2. Add LoadedRowDetailsPresenter="c1dg_LoadedRowDetailsPresenter" LoadingRow="c1dg_LoadingRow" to the <c1:C1DataGrid> tag so that it appears similar to the following:

```
<c1:C1DataGrid x:Name="c1dg" CanUserAddRows="False"
LoadedRowDetailsPresenter="c1dg_LoadedRowDetailsPresenter" LoadingRow="c1dg_LoadingRow">
```

Later you'll add handlers for these events in code.

3. In the Solution Explorer, right-click the project and select **Add Reference**. In the **Add Reference** dialog box, locate **System.Xml.Linq** and **System.ComponentModel.DataAnnotations** and click **OK** to add the reference.

4. Right-click the page and select **View Code** in the context menu to open the Code Editor.

5. In the Code Editor, import the following namespaces:

- C#

```
using System.Xml.Linq;
using C1.Silverlight.DataGrid;
using C1DataGrid_Demo;
```

6. Add code to the **Page** constructor to set the **ItemsSource** property:

- C#

```
public MainPage()
{
   InitializeComponent();
   c1dg.ItemsSource = Data.GetSubCategories(null).Take(10);
}
```

7. Add code for the **c1dg_LoadedRowDetailsPresenter** event to the **MainPage** class:

- C#

```
private void c1dg_LoadedRowDetailsPresenter(object sender,
C1.Silverlight.DataGrid.DataGridRowDetailsEventArgs e)
{
   if (e.Row.DetailsVisibility == Visibility.Visible)
   {
      C1.Silverlight.DataGrid.C1DataGrid detailGrid = e.DetailsElement as
C1.Silverlight.DataGrid.C1DataGrid;
      if (detailGrid.ItemsSource == null)
      {
         int subcategory = (e.Row.DataItem as Subcategory).ProductSubcategoryID;
         detailGrid.ItemsSource = Data.GetProducts((product) =>
product.Element("ProductSubcategoryID") != null && product.Element("ProductSubcategoryID").Value !=
"" && int.Parse(product.Element("ProductSubcategoryID").Value) == subcategory).Take(10);
      }
   }
}
```

8. Add code for the **c1dg_LoadingRow** event to the **MainPage** class to set the row details visibility for the first row:

- C#

```
private void c1dg_LoadingRow(object sender, DataGridRowEventArgs e)
```

```
{
    if (e.Row.Index == 0)
    {
        e.Row.DetailsVisibility = Visibility.Visible;
    }
}
```

What You've Accomplished

If you save and run your application you'll observe that the grid is now populated with data from the products.xml file, and that the first row's details section is visible:

To collapse the row details section or expand another's row detail section, click the arrow icon in the row header of a row:

You've completed this tutorial and learned how to set up row details in the grid to display a master/detail grid view.

Localizing the Grid

Localizing **DataGrid for Silverlight** for various audiences is a fairly simple process as localization in Silverlight is based on the standard .NET localization. For more information about localization, see Localizing the Application. In this tutorial, you'll localize the visible grid strings in an existing application.

Step 1 of 3: Setting up the Localized Grid

In this step you'll create a Silverlight grid application using **DataGrid for Silverlight**. You'll create a new Silverlight project, add the **C1DataGrid** control to your application, and bind the grid.

Complete the following steps:

1. In Visual Studio, select **File | New | Project**.
2. In the **New Project** dialog box, select a language in the left pane, and in the templates list select **Silverlight Application**. Enter "C1DataGridLocalization" in the **Name** text box, and click **OK**. The **New Silverlight Application** dialog box will appear.
3. Click **OK** to close the **New Silverlight Application** dialog box and create your project.
4. In the <UserControl> tag, replace Width="400" (or d:DesignWidth="400") with Width="450" to increase its size.
5. In the XAML window of the project, place the cursor between the <Grid> and </Grid> tags and click once.
6. Navigate to the Toolbox and double-click the **C1DataGrid** icon to add the grid control to **MainPage.xaml**. The XAML markup will now look similar to the following:

```
<UserControl xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
x:Class="C1DataGridLocalization.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Width="450" Height="300">
    <Grid x:Name="LayoutRoot">
        <c1:C1DataGrid></c1:C1DataGrid>
    </Grid>
</UserControl>
```

7. If the <c1:C1DataGrid> tag includes existing content, delete it so it appears similar to the following:

```
<c1:C1DataGrid></c1:C1DataGrid>
```

8. Give your grid a name by adding x:Name="c1dg" to the <c1:C1DataGrid> tag so that it appears similar to the following:

```
<c1:C1DataGrid x:Name="c1dg">
```

By giving the control a unique identifier, you'll be able to access the **C1DataGrid** control in code.

9. Add CanUserGroup="True" to the <c1:C1DataGrid> tag so that it appears similar to the following:

```
<c1:C1DataGrid x:Name="c1dg" CanUserGroup="True">
```

10. In the Solution Explorer, right-click the **C1DataGridLocalization** project and select **Build**.

11. In the Solution Explorer, right-click the **MainPage.xaml** file and click **View Code** in the context menu to open the Code Editor.

12. Add the following code to the project to create the **Data** class:

- Visual Basic

```
Public Class Data
    Private _ProductName As String
    Public Property ProductName() As String
        Get
            Return _ProductName
        End Get
        Set(ByVal value As String)
            _ProductName = value
        End Set
    End Property
    Private _Description As String
    Public Property Description() As String
        Get
            Return _Description
        End Get
        Set(ByVal value As String)
            _Description = value
        End Set
    End Property
    Private _Quantity As Integer
    Public Property Quantity() As Integer
        Get
            Return _Quantity
        End Get
        Set(ByVal value As Integer)
            _Quantity = value
        End Set
    End Property
```

```
    Private _InStock As Boolean
    Public Property InStock() As Boolean
        Get
            Return _InStock
        End Get
        Set(ByVal value As Boolean)
            _InStock = value
        End Set
    End Property
End Class
```

- C#

```
public class Data
{
    public string ProductName { get; set; }
    public string Description { get; set; }
    public int Quantity { get; set; }
    public bool InStock { get; set; }
}
```

13. Add the following code to the **MainPage** constructor to populate the grid:

- Visual Basic

```
Public Sub New()
    InitializeComponent()
    ' Add data to a data source.
    Dim source As New List(Of Data)()
    Dim itemsCount As Integer = 25
    For i As Integer = 0 To itemsCount - 1
        source.Add(New Data With
            {
                .ProductName = "Name",
                .Description = "Description",
                .Quantity = i,
                .InStock = (i Mod 2 = 0)
            })
    Next
    ' Set the grid's ItemsSource property.
    c1dg.ItemsSource = source
End Sub
```

- C#

```
public MainPage()
```

```
{
    InitializeComponent();
    // Add data to a data source.
    List<Data> source = new List<Data>();
    int itemsCount = 25;
    for (int i = 0; i < itemsCount; i++)
    {
        source.Add(new Data()
        {
            ProductName = "Name",
            Description = "Description",
            Quantity = i,
            InStock = (i % 2 == 0)
        });
    }
    // Set the grid's ItemsSource property.
    c1dg.ItemsSource = source;
}
```

What You've Accomplished

In this step you created a new Silverlight application, added a **C1DataGrid** control, and bound the control to a data source. In the next step, you'll add a resource file to localize the grid.

Step 2 of 3: Adding a Resource File

In this step, you'll begin by adding a resource file to your application. Note that if you choose, you can add multiple resources files to your project.

1. In the Solution Explorer, right-click the **C1DataGridLocalization** project and choose **Add | New Folder**.
2. Name the folder you just created "Resources".
3. Right-click the **Resources** folder, and in the context menu select **Add | New Item**.
4. In the **Add New Item** dialog box, select **Resources File** in the templates pane, name the file "C1.Silverlight.DataGrid.resx", and click **Add** to add the file to your project.
5. If the resource file did not automatically open, double-click the file name in the Solution Explorer.
6. In the C1.Silverlight.DataGrid.es.resx file, add the following Names and Values:

Name	Value
AddNewRow	Click here to add a new row
CheckBoxFilter_Checked	Checked:
ComboBoxFilter_SelectAll	Select All
DateTimeFilter_End	End

DateTimeFilter_Start	Start
EmptyGroupPanel	Drag a column here to group by that column
Filter_Clear	Clear
Filter_Filter	Filter
NumericFilter_And	And
NumericFilter_GreaterOrEquals	Greater/Equals
NumericFilter_Greater	Greater
NumericFilter_Less	Less
NumericFilter_LessOrEquals	Less/Equals
NumericFilter_NotEquals	Not Equals
NumericFilter_Or	Or
TextFilter_Contains	Contains
TextFilter_StartsWith	Starts With
TextFilter_Equals	Equals
TextFilter_NotEquals	Not Equals

1. Save and close the resource file.
2. Right-click the **Resources** folder, and in the context menu select **Add | New Item**.
3. In the **Add New Item** dialog box, select **Resources File** in the templates pane, name the file "C1.Silverlight.DataGrid.es.resx", and click **Add** to add the file to your project.

This file will localize the application to Spanish. For information on file naming, see Adding Resource Files.

4. If the resource file did not automatically open, double-click the file name in the Solution Explorer.
5. In the C1.Silverlight.DataGrid.es.resx file, add the following Names and Values to add Spanish localization:

Name	Value
AddNewRow	Cliquee aquí para agregar un nuevo renglón
CheckBoxFilter_Checked	Seleccionado:
ComboBoxFilter_SelectAll	Seleccionar todo
DateTimeFilter_End	Fin
DateTimeFilter_Start	Inicio
EmptyGroupPanel	Arrastre una columna aquí para agrupar
Filter_Clear	Borrar
Filter_Filter	Filtrar
NumericFilter_And	Y

NumericFilter_Equals	Igual
NumericFilter_GreaterOrEquals	Mayor o igual
NumericFilter_Greater	Mayor
NumericFilter_Less	Menor
NumericFilter_LessOrEquals	Menor o igual
NumericFilter_NotEquals	Diferente
NumericFilter_Or	O
TextFilter_Contains	Contiene
TextFilter_StartsWith	Empieza con
TextFilter_Equals	Igual
TextFilter_NotEquals	Diferente

1. Save and close the resource file.

What You've Accomplished

In this step you added a new resource file to your application. In the next step you'll add the file's culture to the project's supported cultures, and then set that culture to be the current culture.

Step 3 of 3: Setting the Culture

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

1. In the Solution Explorer, right-click the **C1DataGridLocalization** project and select **Unload Project**. Click **Yes** if Visual Studio asks you to save the project.

The project will appear grayed out and unavailable.

2. Right-click the project again, and select the **Edit C1DataGridLocalization.csproj** option.

In the .csproj file, locate the <SupportedCultures></SupportedCultures> tags. Add "es" in between the tags, so they appear similar to the following:

<SupportedCultures>es</SupportedCultures>

3. Save and close the .csproj file.

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

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

5. In the Solution Explorer, right-click the **MainPage.xaml** file and click **View Code** in the context menu to open the Code Editor.

6. Add the following using statements to the top of the file:

- Visual Basic

```
Imports System.Globalization
Imports System.Threading
```

- C#

```
using System.Globalization;
using System.Threading;
```

7. Add the following code to the **MainPage** constructor above the **InitializeComponent()** call to set the **CurrentUICulture** property:

- Visual Basic

```
Thread.CurrentThread.CurrentUICulture = New CultureInfo("es")
```

- C#

```
Thread.CurrentThread.CurrentUICulture = new CultureInfo("es");
```

It should now look similar to the following:

- Visual Basic

```
Public Sub New()
    ' Set the culture.
    Thread.CurrentThread.CurrentUICulture = New CultureInfo("es")
    InitializeComponent()
    ' Add data to a data source.
    Dim source As New List(Of Data)()
    Dim itemsCount As Integer = 25
    For i As Integer = 0 To itemsCount - 1
        source.Add(New Data())
    Next
    ' Set the grid's ItemsSource property.
    c1dg.ItemsSource = source
End Sub
```

- C#

```
public MainPage()
{
    // Set the culture.
    Thread.CurrentThread.CurrentUICulture = new CultureInfo("es");
    InitializeComponent();
    // Add data to a data source.
    List<Data> source = new List<Data>();
    int itemsCount = 25;
    for (int i = 0; i < itemsCount; i++)
    {
        source.Add(new Data()
        {
            ProductName = "Name",
            Description = "Description",
            Quantity = i,
            InStock = (i % 2 == 0)
        });
```

```
            }
            // Set the grid's ItemsSource property.
            c1dg.ItemsSource = source;
        }
```

8. Save and run your application.
9. To observe some of the localized language strings, select the drop-down filter icon in the grid:
10. Click the drop-down arrow in the filter box to view additional strings:

What You've Accomplished

In this step you added the file's culture to the project's supported cultures and set that culture to be the current culture. In this tutorial you've learned how to localize an application. You created a resource file, set the project's supported culture, and explicitly set the current culture in code.

Binding the Grid to a WCF RIA Services Data Source

The following tutorial will walk you through the process of binding the **C1DataGrid** control to a WCF RIA Services data source. For more information, see the WCF RIA Services Data Binding topic. Note that this example will use files included in the **C1DataGrid_Ria2010** sample installed with **Silverlight Edition**.

Step 1 of 3: Creating the Application and Adding the Data Source

In this step you'll create a new Silverlight project with WCF RIA services enabled, add a data source, and set up the client-side project. Complete the following steps:

1. In Visual Studio 2010, select **File | New | Project**.
2. In the **New Project** dialog box, choose **Visual C#** in the left pane, and in the templates list select **Silverlight Application**. Enter "C1DataGridRIA" in the **Name** text box, and click **OK**. The **New Silverlight Application** dialog box will appear.
3. In the **New Silverlight Application** dialog box, check the **Enable WCF RIA Services** check box and click **OK** to close the **New Silverlight Application** dialog box and create your project.

1. In the Solution Explorer, right-click the **C1DataGridRIA.Web** project and choose **Add | New Folder**. Rename the folder "App_Data".
2. Right-click the **App_Data** folder and select **Add | Existing Item**.
3. In the **Add Existing Item** dialog box, navigate to where the ComponentOne samples are installed, by default in the **Documents** or **My Documents** folder, and navigate to the **ComponentOne Samples\Studio for Silverlight 4.0\C1.Silverlight.DataGrid\C1DataGrid_Ria\C1DataGrid_Ria2010Web\App_Data** folder. Choose the **NORTHWND.MDF** file and click the **Add** button.

The database will be added to your project. Note that this is the standard Microsoft Northwind database.

4. In the Solution Explorer, right-click the **C1DataGridRIA.Web** project and choose **Add | New Item**.
5. In the Add New Item dialog box choose **Data** in the left list and choose **ADO.NET Entity Data Model** from the list of data templates. Name the file "NorthwindModel" and click **Add** to add the file to your project.
6. The **Entity Data Model Wizard** should appear. Choose the **Generate from database** option and click **Next**.
7. In the **Choose Your Data Connection** screen, confirm that the **NORTHWND.MDF** file is selected. If it is not selected, choose **New Connection** and locate the file. Save the connection string with the default name, "NORTHWNDEntities", and click **Next**.

8. In the **Choose Your Database Objects** screen, select the **Tables** check box to choose the **Products** table. Click **Finish**.

9. Choose **Build | Rebuild Solution** to build the entire solution and make sure the autogenerated RIA Services files get created.

10. In the Solution Explorer, right-click the **C1DataGridRIA.Web** project and choose **Add | New Item**.

11. In the **Add New Item** dialog box choose **Web** in the left list and choose **Domain Service Class** from the list of code templates. Name the file "NorthwindService" and click **Add** to add the file to your project. The **Add New Domain Service Class** dialog box will appear.

12. Iin the **Add New Domain Service Class** dialog box, select **NorthwindEntities**as DataContext item and select the **Enable client access** check box. Check the **Product** entity and **Enable editing** check boxes and click **OK**.

13. Save the project and choose **Build | Rebuild Solution** to ensure everything is working correctly.

What You've Accomplished

In this step you added a new RIA data source to your application. In the next step you'll add the **C1DataGrid** control to the application.

Step 2 of 3: Adding the C1DataGrid Control

In the previous step you created a new Silverlight application with WCF RIA services enabled and added a new data source. In this step you'll set up your application and add the **C1DataGrid** control to the application. Complete the following steps:

1. In the Solution Explorer, right click the **C1DataGridRIA** project and choose **Add Reference**. The **Add Reference** dialog box will appear.

2. In the **Add Reference** dialog box, select the following assemblies and then click **OK**:

- System.Windows.Controls.Data
- System.Windows.Controls.DomainServices
- C1.Silverlight
- C1.Silverlight.DataGrid
- C1.Silverlight.DataGrid.Ria

This will add references to the project for the selected assemblies.

3. In the Solution Explorer, double-click the **MainPage.xaml** file to open it.

4. In the XAML window of the project, update the **UserControl** tag so it appears similar to the following:

- XAML

```
<UserControl x:Class="C1DataGridRIA.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:data="clr-namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data"
    xmlns:ria="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.DomainServices"
    xmlns:c1data="clr-namespace:C1.Silverlight.DataGrid;assembly=C1.Silverlight.DataGrid"
    xmlns:adapter="clr-namespace:C1.Silverlight.DataGrid.Ria;assembly=C1.Silverlight.DataGrid.Ria"
```

```
xmlns:local="clr-namespace:C1DataGridRIA.Web"
mc:Ignorable="d" d:DesignWidth="640" d:DesignHeight="480">
```

This markup will add references to the assemblies you added, and resize the **UserControl**.

5. Add the following markup just after the **Grid** tag to create a row definition:

- XAML

```
<Grid.RowDefinitions>
  <RowDefinition Height="Auto"/>
  <RowDefinition Height="*"/>
</Grid.RowDefinitions>
```

This markup will set the layout of the page.

6. Add the following markup within the **Grid** tag and just under the row definitions to create a **C1RiaAdaptor**:

- XAML

```
<!-- RIA Data Source -->
<adapter:C1RiaAdapter x:Name="_adapter" DataGrid="{Binding ElementName=_dataGrid}">
  <ria:DomainDataSource x:Name="_myDataSource" QueryName="GetProducts" PageSize="8">
    <ria:DomainDataSource.DomainContext>
      <local:NorthwindContext/>
    </ria:DomainDataSource.DomainContext>
    <ria:DomainDataSource.GroupDescriptors>
      <ria:GroupDescriptor PropertyPath="CategoryID"/>
      <ria:GroupDescriptor PropertyPath="Discontinued"/>
    </ria:DomainDataSource.GroupDescriptors>
    <ria:DomainDataSource.SortDescriptors>
      <ria:SortDescriptor PropertyPath="ProductName" Direction="Descending"/>
    </ria:DomainDataSource.SortDescriptors>
    <ria:DomainDataSource.FilterDescriptors>
      <ria:FilterDescriptor PropertyPath="UnitPrice" Operator="IsGreaterThanOrEqualTo" Value="18"/>
      <ria:FilterDescriptor PropertyPath="ProductName" Operator="Contains" Value="C"/>
    </ria:DomainDataSource.FilterDescriptors>
  </ria:DomainDataSource>
</adapter:C1RiaAdapter>
```

This markup will add the RIA data source.

7. Add the following markup within the **Grid** tag and under the **C1RiaAdaptor** tag to add a header to the page:

- XAML

```
<!-- Header -->
<Border Grid.Row="0" Height="40" Background="LightBlue">
  <TextBlock Text="CollectionView adapter for C1DataGrid: RIA Services"
    Margin="10 0 0 0" FontSize="15" FontWeight="Bold" VerticalAlignment="Center"/>
</Border>
```

8. Add the following markup within the **Grid** tag and under the Header to add a layout **Grid** to the page:

- XAML

```
<!-- Content -->
<Grid Grid.Row="1" Margin="20">
  <Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="*"/>
  </Grid.RowDefinitions>
</Grid>
```

You will add the **C1DataGrid** control within this layout grid.

9. Add the following markup within within the content layout **Grid** you just added (just above the </Grid> tag) to add a standard **DataPager** control to the page:

- XAML

```
<!-- DataPager -->
  <data:DataPager x:Name="_dataPager" Source="{Binding Data, ElementName=_myDataSource}"
BorderThickness="0" Background="White"/>
```

10. Add the following markup within within the content layout **Grid** and just after the **DataPager** to add a **C1DataGrid** control to the page:

- XAML

```
<!-- C1DataGrid -->
<c1data:C1DataGrid x:Name="_dataGrid" CanUserGroup="True" AutoGenerateColumns="False"
Grid.Row="1"
    CanUserAddRows="True" CanUserEditRows="True" CanUserRemoveRows="True"
    ItemsSource="{Binding Data, ElementName=_adapter}"
    BeginningRowEdit="_dataGrid_BeginningRowEdit"
CommittingRowEdit="_dataGrid_CommittingRowEdit"
    CancelingRowEdit="_dataGrid_CancelingRowEdit" RowsDeleted="_dataGrid_RowsDeleted" >
  <c1data:C1DataGrid.Columns>
    <c1data:DataGridNumericColumn Binding="{Binding CategoryID, Mode=TwoWay}"
SortMemberPath="CategoryID" FilterMemberPath="CategoryID" Header="CategoryID"/>
    <c1data:DataGridCheckBoxColumn Binding="{Binding Discontinued, Mode=TwoWay}"
SortMemberPath="Discontinued" FilterMemberPath="Discontinued" Header="Discontinued"/>
    <c1data:DataGridTextColumn Binding="{Binding ProductName, Mode=TwoWay}"
SortMemberPath="ProductName" FilterMemberPath="ProductName" Header="ProductName"/>
    <c1data:DataGridTextColumn Binding="{Binding QuantityPerUnit, Mode=TwoWay}"
SortMemberPath="QuantityPerUnit" FilterMemberPath="QuantityPerUnit" Header="QtyPerUnit"/>
    <c1data:DataGridNumericColumn Binding="{Binding UnitPrice, Mode=TwoWay}"
SortMemberPath="UnitPrice" FilterMemberPath="UnitPrice" Header="UnitPrice"/>
```

```
    </c1data:C1DataGrid.Columns>
</c1data:C1DataGrid>
```

This **C1DataGrid** control is bound to the database added earlier and includes defined and bound columns.

11. Add the following markup within within the content layout **Grid** and just after the **C1DataGrid** to add a text box and two buttons to the page:

- XAML

```
<!-- Change Text -->
<TextBox x:Name="_changeText" Margin="0 4 0 0" Grid.Row="2"/>
<!-- Reject Button -->
<Button x:Name="_rejectButton" Content="Reject Changes" IsEnabled="False"
Click="_rejectButton_Click" Width="120" HorizontalAlignment="Right" Margin="0 4 130 0"
Grid.Row="3"/>
<!-- Submit Button -->
<Button x:Name="_submitButton" Content="Submit Changes" IsEnabled="False"
Click="_submitButton_Click" Width="120" HorizontalAlignment="Right" Margin="0 4 0 0"
Grid.Row="3"/>
```

At run time, the text box will display the location of any changes made to the grid and the buttons will allow you to reject or apply any changes made to the grid at run time. In the next step you'll add code to implement the XAML you added to the application.

12. Right-click the **MainPage.xaml** page and choose **View Code** to open the **MainPage.xaml.cs** (or **MainPage.xaml.vb**) page in the Code Editor.

13. Add the imports statement to the top of the page:

- Visual Basic

```
Imports C1.Silverlight.DataGrid
Imports System.ServiceModel.DomainServices.Client
```

- C#

```
using C1.Silverlight.DataGrid;
using System.ServiceModel.DomainServices.Client;
```

14. Add the following code within the **MainPage** class to implement the controls that were added in XAML:

- Visual Basic

```
Private Sub _submitButton_Click(sender As Object, e As RoutedEventArgs)
  ' Submit changes to the server
  _dataGrid.IsLoading = True
  _myDataSource.DomainContext.SubmitChanges(AddressOf OnSubmitCompleted, Nothing)
End Sub
Private Sub _rejectButton_Click(sender As Object, e As RoutedEventArgs)
  ' Reject changes
  _myDataSource.DomainContext.RejectChanges()
  CheckChanges()
  _dataGrid.Reload(False)
End Sub
```

```
' Disable submit/reject buttons when there are pending changes in the row
Private Sub _dataGrid_BeginningRowEdit(sender As Object, e As DataGridEditingRowEventArgs)
  _submitButton.IsEnabled = False
  _rejectButton.IsEnabled = False
End Sub
' Enable/disable submit/reject buttons after pending changes are committed
Private Sub _dataGrid_CommittingRowEdit(sender As Object, e As DataGridEditingRowEventArgs)
  CheckChanges()
End Sub
' Enable/disable submit/reject buttons after pending changes are canceled
Private Sub _dataGrid_CancelingRowEdit(sender As Object, e As DataGridEditingRowEventArgs)
  CheckChanges()
End Sub
' Enable/disable submit/reject buttons after rows deleted
Private Sub _dataGrid_RowsDeleted(sender As Object, e As DataGridRowsDeletedEventArgs)
  CheckChanges()
End Sub
' Check the pending changes to submit/reject and enable/disable buttons according to this.
Private Sub CheckChanges()
  Dim changeSet As EntityChangeSet = _myDataSource.DomainContext.EntityContainer.GetChanges()
  _changeText.Text = changeSet.ToString()
  Dim hasChanges As Boolean = _myDataSource.HasChanges
  _submitButton.IsEnabled = hasChanges
  _rejectButton.IsEnabled = hasChanges
End Sub
' Check for errors when submitting changes to the server
Private Sub OnSubmitCompleted(so As SubmitOperation)
  _dataGrid.IsLoading = False
  If so.HasError Then
      MessageBox.Show(String.Format("Submit Failed: {0}", so.[Error].Message))
      so.MarkErrorAsHandled()
  End If
  CheckChanges()
End Sub
```

- C#

```
private void _submitButton_Click(object sender, RoutedEventArgs e)
{
   // Submit changes to the server
   _dataGrid.IsLoading = true;
```

```
    _myDataSource.DomainContext.SubmitChanges(OnSubmitCompleted, null);
}
private void _rejectButton_Click(object sender, RoutedEventArgs e)
{
    // Reject changes
    _myDataSource.DomainContext.RejectChanges();
    CheckChanges();
    _dataGrid.Reload(false);
}
// Disable submit/reject buttons when there are pending changes in the row
private void _dataGrid_BeginningRowEdit(object sender, DataGridEditingRowEventArgs e)
{
    _submitButton.IsEnabled = false;
    _rejectButton.IsEnabled = false;
}
// Enable/disable submit/reject buttons after pending changes are committed
private void _dataGrid_CommittingRowEdit(object sender, DataGridEditingRowEventArgs e)
{
    CheckChanges();
}
// Enable/disable submit/reject buttons after pending changes are canceled
private void _dataGrid_CancelingRowEdit(object sender, DataGridEditingRowEventArgs e)
{
    CheckChanges();
}
// Enable/disable submit/reject buttons after rows deleted
private void _dataGrid_RowsDeleted(object sender, DataGridRowsDeletedEventArgs e)
{
    CheckChanges();
}
// Check the pending changes to submit/reject and enable/disable buttons according to this.
private void CheckChanges()
{
    EntityChangeSet changeSet = _myDataSource.DomainContext.EntityContainer.GetChanges();
    _changeText.Text = changeSet.ToString();
    bool hasChanges = _myDataSource.HasChanges;
    _submitButton.IsEnabled = hasChanges;
    _rejectButton.IsEnabled = hasChanges;
```

```
}
// Check for errors when submitting changes to the server
private void OnSubmitCompleted(SubmitOperation so)
{
    _dataGrid.IsLoading = false;
    if (so.HasError)
    {
        MessageBox.Show(string.Format("Submit Failed: {0}", so.Error.Message));
        so.MarkErrorAsHandled();
    }
    CheckChanges();
}
```

What You've Accomplished

You learned how to bind the **C1DataGrid** control to an RIA Services data source. You created a Silverlight application, added the data source, and added and implemented the **C1DataGrid** control. In the next step you'll run the application, to view its run time interactions.

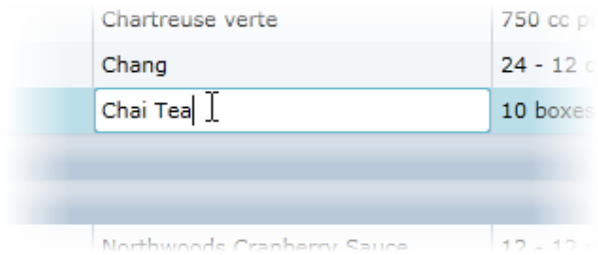
手順 3:アプリケーションの実行

前の手順では、WCF RIA Services を有効にして新しい Silverlight アプリケーションを作成し、新しいデータソースを追加し、アプリケーションに C1DataGrid コントロールを追加しました。この手順では、このアプリケーションを実行して実行時の操作を確認します。次の手順に従います。

1. プロジェクトを保存し、[デバッグ]→[デバッグ開始]を選択してアプリケーションを実行します。次の画像のように表示されます。

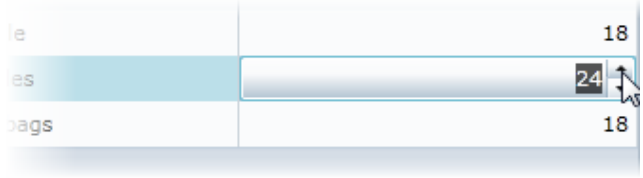
2. 実行時に、製品名列の1つのセルをクリックし、セルからテキストを削除します。確認テキストが表示されます。

3. 削除した 製品名列のセルにテキストを入力します。



編集したセルの外をクリックし、グリッドの下のボックスに注目してください。グリッド内の1つのセルが変更され、グリッドの下のボタンがアクティブになっていることがわかります。

4. [変更の拒否]ボタンをクリックして、行った変更を破棄します。
5. 単価列内の項目をクリックし、上向き/下向き矢印ボタンを使用して、セルの値を変更します。



6. セルの外をクリックし、[変更の送信]ボタンをクリックして、変更をデータに保存します。

ここまでの成果

このチュートリアルでは、C1DataGrid コントロールを RIA サービスのデータソースに連結する方法について学習しました。Silverlight アプリケーションを作成してデータソースを追加し、C1DataGrid コントロールを追加して実装しました。

Implementing Stealth Paging

With paging you can only load the necessary data to fit one page. See Paging Grid Data for details. Stealth paging is a little different; you can achieve paging functionality with a scrollbar. As the user scrolls down the grid, more data is fetched as needed, just like with paging. **C1DataGrid** supports server-side sorting and filtering so you can still achieve these functionalities without sacrificing performance. In this tutorial you'll create a Silverlight application with a **C1DataGrid** control that implements stealth paging functionality.

Step 1 of 3: Creating the User Interface

In this step you'll begin in Visual Studio to create a Silverlight grid application. You'll then continue by creating and customizing the application's user interface (UI) and adding the **C1DataGrid** control to your project.

To set up your project, complete the following steps:

1. In Visual Studio, select **File | New | Project**.

2. In the **New Project** dialog box, select a language in the left pane and in the templates list select **Silverlight Application**. Enter a **Name** for your project, for example "ComponentOneDataGrid", and click **OK**. The **New Silverlight Application** dialog box will appear.

3. Click **OK** to accept the default settings, close the **New Silverlight Application** dialog box, and create your project.

4. If the **MainPage.xaml** file is not currently open, navigate to the Solution Explorer and double-click on the **MainPage.xaml** item.

5. In the XAML view, locate the <UserControl> tag.

6. In the <UserControl> tag, replace Width="400" Height="300" (or d:DesignWidth="400" d:DesignHeight="300") with Width="600" Height="400".

This will increase the size of your Silverlight application.

7. Place the cursor just after the <Grid x:Name="LayoutRoot" Background="White"> tag and add the following markup:

```
<!-- Grid Layout-->
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="*" />
<RowDefinition Height="Auto" />
</Grid.RowDefinitions>
```

This row definition will define the layout of your grid.

8. Add a title to your application by adding the following **TextBlock** just under the </Grid.RowDefinitions> tag:

```
<!-- Title -->
<TextBlock Text="DataGrid for Silverlight" Margin="5" FontSize="16"/>
```

9. In the XAML window of the project, place the cursor just above the </Grid> tag and click once.

10. Navigate to the Toolbox and double-click the **C1DataGrid** icon to add the grid control to **MainPage.xaml**. The XAML markup will now look similar to the following:

```
<UserControl xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
x:Class="C1DataGrid.MainPage" xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Width="600" Height="400">
  <Grid x:Name="LayoutRoot" Background="White">
    <!-- Grid Layout-->
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto" />
      <RowDefinition Height="*" />
      <RowDefinition Height="Auto" />
    </Grid.RowDefinitions>
    <!-- Title -->
    <TextBlock Text="DataGrid for Silverlight" Margin="5" FontSize="16"/>
     <c1:C1DataGrid></c1:C1DataGrid>
  </Grid>
</UserControl>
```

Note that the **C1.Silverlight.DataGrid** namespace and <c1:C1DataGrid></c1:C1DataGrid> tags have been added to the project.

11. If the <c1:C1DataGrid> tag includes existing content, delete it so it appears similar to the following:

```
<c1:C1DataGrid>
```

12. Give your grid a name by adding x:Name="_c1DataGrid" to the <c1:C1DataGrid> tag so that it appears similar to

the following:

```
<c1:C1DataGrid x:Name="_c1DataGrid">
```

By giving the control a unique identifier, you'll be able to access the **C1DataGrid** control in code.

13. Define the location of your grid by adding Grid.Row="1" to the <c1:C1DataGrid> tag so that it appears similar to the following:

```
<c1:C1DataGrid x:Name="_c1DataGrid" Grid.Row="1">
```

14. Add the following markup just after the </c1:C1DataGrid> tag:

```
<TextBlock x:Name="_tbStatus" Text="Ready"
  VerticalAlignment="Center" FontSize="12" Foreground="Gray" Margin="5" Grid.Row="2" />
```

This **TextBlock** will be used to display status information text.

What You've Accomplished

Run your application, and observe that your page includes a title, a grid, and text below the grid. You've successfully created a basic grid application, but the grid is blank and contains no data. In the next steps you'll add a database to your project and bind the grid to a data source.

Step 2 of 3: Adding a Web Service

In this step you'll add a data source to your project, and begin the process of binding the grid.

To set up your project, complete the following steps:

1. Navigate to the Solution Explorer, right-click the **StealthPaging.Web** project, and select **Add Reference** from the context menu.

1. In the **Add Reference** dialog box locate the **System.Runtime.Serialization** assembly and click the **OK** button to add a reference to your project. The dialog box will close and the reference will be added.

2. In the Solution Explorer right-click the **StealthPaging.Web** project, and select **Add | New Item**.

3. In the left pane of the **Add New Item** dialog box, select the **Web** item.

4. In the templates list, select **Web Service**, name the Web Service "DataWebService.asmx", and click the **Add** button. Note that the Web Service file will be added to your project and automatically opened.

5. In the **DataWebService.asmx** file, add the following using statements at the top of the file:

- Visual Basic

```
Imports System.Runtime.Serialization
```

- C#

```
using System.Runtime.Serialization;
```

6. In the **DataWebService.asmx** file, replace the code in the **StealthPaging.Web** namespace with the following::

- Visual Basic

```
' To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the following line.
' <System.Web.Script.Services.ScriptService()> _
<System.Web.Services.WebService(Namespace:="http://tempuri.org/")> _
<System.Web.Services.WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1)> _
<ToolboxItem(False)> _
Public Class DataWebService
    Inherits System.Web.Services.WebService
```

```
    <WebMethod()> _
    Public Function GetData(startRow As Integer, endRow As Integer) As List(Of ServerPerson)
        Dim personList As New List(Of ServerPerson)()
        For i As Integer = startRow To endRow - 1
            personList.Add(New ServerPerson() With { _
                .FirstName = String.Format("First Name {0}", i), _
                .LastName = String.Format("Last Name {0}", i), _
                .Age = i, _
                .City = String.Format("City {0}", i) _
            })
        Next
        Return personList
    End Function
End Class
<DataContract> _
Public Class ServerPerson
    Private _firstName As String
    <DataMember> _
    Public Property FirstName() As String
        Get
            Return _firstName
        End Get
        Set
            _firstName = value
        End Set
    End Property
    Private _lastName As String
    <DataMember> _
    Public Property LastName() As String
        Get
            Return _lastName
        End Get
        Set
            _lastName = value
        End Set
    End Property
    Private _age As Integer
    <DataMember> _
    Public Property Age() As Integer
```

```
            Get
                Return _age
            End Get
            Set
                _age = value
            End Set
        End Property
        Private _city As String
        <DataMember> _
        Public Property City() As String
            Get
                Return _city
            End Get
            Set
                _city = value
            End Set
        End Property
    End Class
```

- C#

```
namespace StealthPaging.Web
{
    /// <summary>
    /// Summary description for DataWebService
    /// </summary>
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [System.ComponentModel.ToolboxItem(false)]
    // To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the following line.
    // [System.Web.Script.Services.ScriptService]
    public class DataWebService : System.Web.Services.WebService
    {
        [WebMethod]
        public List<ServerPerson> GetData(int startRow, int endRow)
        {
            List<ServerPerson> personList = new List<ServerPerson>();
            for (int i = startRow; i < endRow; i++)
            {
                personList.Add(new ServerPerson()
```

```
            {
                FirstName = string.Format("First Name {0}", i),
                LastName = string.Format("Last Name {0}", i),
                Age = i,
                City = string.Format("City {0}", i)
            });
        }
        return personList;
    }
}
[DataContract]
public class ServerPerson
{
    private string _firstName;
    [DataMember]
    public string FirstName
    {
        get { return _firstName; }
        set { _firstName = value; }
    }
    private string _lastName;
    [DataMember]
    public string LastName
    {
        get { return _lastName; }
        set { _lastName = value; }
    }
    private int _age;
    [DataMember]
    public int Age
    {
        get { return _age; }
        set { _age = value; }
    }
    private string _city;
    [DataMember]
    public string City
    {
        get { return _city; }
```

```
            set { _city = value; }
        }
    }
}
```

This code will create a new list that will be used to populate the C1DataGrid control.

7. Save your application, right-click the **StealthPaging.Web** project, and select **Build** from the context menu. Note that you'll now be done with the **StealthPaging.Web** project and will return to working with the **StealthPaging** project.

What You've Accomplished

In this step you've added a data source to your project and created a Web Service. In the next step you'll finish connecting the Web Service to your project and you'll run your application.

Step 3 of 3: Connecting the Web Service and Adding Stealth Paging

In the previous step you created a Web Service and added a data source to your project. In this step you'll continue by linking the Web Service to your application.

To set up your project, complete the following steps:

1. Return to the **MainPage.xaml** file.

1. In the Solution Explorer, right-click the project name and select **Add Service Reference** from the context menu.

2. In the **Add Service Reference** dialog box click the **Discover** button. The DataWebService.asmx file will appear in the list of Services.

3. In the **Namespace** text box, change the default value to "DataService" and click the **OK** button to save your settings and close the dialog box.

4. Customize your grid by adding LoadedRowPresenter="peopleDataGrid_LoadedRowPresenter" to the <c1:C1DataGrid> tag so that it appears similar to the following:

```
<c1:C1DataGrid x:Name="peopleDataGrid" AutoGenerateColumns="True" CanUserAddRows="False"
LoadedRowPresenter="peopleDataGrid_LoadedRowPresenter">
```

This markup adds an event handler – you'll add code for the event handler in the next steps.

5. In the Solution Explorer, expand the **MainPage.xaml** node and double-click the **MainPage.xaml.cs** or **MainPage.xaml.vb** file to open it in the Code Editor.

6. Add the following import statements at the top of the file:

- Visual Basic

```
Imports System.Runtime.Serialization
Imports System.Collections.ObjectModel
Imports System.ServiceModel
Imports C1.Silverlight
Imports C1.Silverlight.DataGrid
Imports StealthPaging.DataService ' Change this if the name of your project is different.
```

- C#

```
using System.Runtime.Serialization;
```

```
using System.Collections.ObjectModel;
using System.ServiceModel;
using C1.Silverlight;
using C1.Silverlight.DataGrid;
using StealthPaging.DataService; // Change this if the name of your project is different.
```

7. Add the following variables to the **MainPage** class:

- Visual Basic

```
Dim _startRow As Integer = 0
Dim _pageSize As Integer = 20
Dim _people As New ObservableCollection(Of ServerPerson)()
Dim _loading As Boolean
```

- C#

```
int _startRow = 0;
int _pageSize = 20;
ObservableCollection<ServerPerson> _people = new ObservableCollection<ServerPerson>();
bool _loading;
```

8. Add code to the **MainPage** constructor so it appears like the following:

- Visual Basic

```
Public Sub New()
  InitializeComponent()
  AddHandler peopleDataGrid.LoadedRowPresenter, AddressOf peopleDataGrid_LoadedRowPresenter
  peopleDataGrid.ItemsSource = _people
  GetData(_startRow, _pageSize)
End Sub
```

- C#

```
public MainPage()
{
   InitializeComponent();
   peopleDataGrid.LoadedRowPresenter += new EventHandler<DataGridRowEventArgs>
(peopleDataGrid_LoadedRowPresenter);
   peopleDataGrid.ItemsSource = _people;
   GetData(_startRow, _pageSize);
}
```

9. Add the **LoadedRowPresenter** event handler to your code under the **MainPage** constructor:

- Visual Basic

```
Private Sub peopleDataGrid_LoadedRowPresenter(ByVal sender As System.Object, ByVal e As
C1.Silverlight.DataGrid.DataGridRowEventArgs)
  If _loading OrElse _people.Count < _pageSize Then
      Return
```

```
    End If
    If _people.Count - 5 < e.Row.Index Then
        GetData(_startRow, _startRow + _pageSize)
    End If
End Sub
```

- C#

```
private void peopleDataGrid_LoadedRowPresenter(object sender,
C1.Silverlight.DataGrid.DataGridRowEventArgs e)
{
    if (_loading || _people.Count < _pageSize)
    {
        return;
    }
    if (_people.Count - 5 < e.Row.Index)
    {
        GetData(_startRow, _startRow + _pageSize);
    }
}
```

10. Add the following code to retrieve data from the server:

- Visual Basic

```
#Region "retrieve data from the server"
Private Sub GetData(startRow As Integer, endRow As Integer)
    UpdateState(True, startRow, endRow)
    ' Call web service
    Dim proxy = New DataWebServiceSoapClient(New BasicHttpBinding(), New
EndpointAddress(Extensions.GetAbsoluteUri("DataWebService.asmx")))
    AddHandler proxy.GetDataCompleted, AddressOf proxy_GetDataCompleted
    proxy.GetDataAsync(startRow, endRow)
End Sub
Private Sub proxy_GetDataCompleted(sender As Object, e As GetDataCompletedEventArgs)
    If e.[Error] IsNot Nothing Then
        MessageBox.Show(e.[Error].Message, "Error Getting Data", MessageBoxButton.OK)
        Return
    End If
    ' Data retrieved OK, add to observable collection
    _startRow += _pageSize
    For Each person As ServerPerson In e.Result
        _people.Add(person)
    Next
```

```
    UpdateState(False, 0, 0)
End Sub
' Sets loading status
' You could use a VisualState here too
Private Sub UpdateState(loading As Boolean, startRow As Integer, endRow As Integer)
    If loading Then
        txtStatus.Text = String.Format("Retrieving rows {0} to {1}...", startRow, endRow)
        Cursor = Cursors.Wait
        _loading = True
    Else
        _loading = False
        txtStatus.Text = "Ready"
        Cursor = Cursors.Arrow
    End If
End Sub
#End Region
```

- C#

```
#region retrieve data from the server
private void GetData(int startRow, int endRow)
{
    UpdateState(true, startRow, endRow);
    // Call Web service
    var proxy = new DataWebServiceSoapClient(new BasicHttpBinding(), new
EndpointAddress(Extensions.GetAbsoluteUri("DataWebService.asmx")));
    proxy.GetDataCompleted += new EventHandler<GetDataCompletedEventArgs>
(proxy_GetDataCompleted);
    proxy.GetDataAsync(startRow, endRow);
}
void proxy_GetDataCompleted(object sender, GetDataCompletedEventArgs e)
{
    if (null != e.Error)
    {
        MessageBox.Show(e.Error.Message, "Error Getting Data", MessageBoxButton.OK);
        return;
    }
    // Data retrieved OK, add to observable collection
    _startRow += _pageSize;
    foreach (ServerPerson person in e.Result)
    {
```

```
            _people.Add(person);
        }
        UpdateState(false, 0, 0);
    }
    // Sets loading status
    // You could use a VisualState here too
    private void UpdateState(bool loading, int startRow, int endRow)
    {
        if (loading)
        {
            txtStatus.Text = string.Format("Retrieving rows {0} to {1}...", startRow, endRow);
            Cursor = Cursors.Wait;
            _loading = true;
        }
        else
        {
            _loading = false;
            txtStatus.Text = "Ready";
            Cursor = Cursors.Arrow;
        }
    }
    #endregion
```

11. Run your application and observe that the grid appears bound to a data source:
12. Run your application and observe that as you scroll through the grid more rows appear in the grid:

Also note that the text below the grid indicates the rows being added as you scroll.

What You've Accomplished

Congratulations, you've completed this tutorial! In this tutorial you created a new Silverlight project, added a data source, and created a Web Service to bind the **C1DataGrid** control. You implemented stealth paging, so that when the grid is scrolled at run time, the grid pages through the grid instead, improving performance.

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 **DataGrid for WPF and Silverlight**, please see the Quick Start first.

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

Creating a DataGrid

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

At Design Time in Blend

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

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

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

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

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

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

In XAML

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

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

   XAML

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

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

   XAML

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

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

In Code

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

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

Visual Basic

```
Imports C1.WPF|variable=WPF.DataGrid
```

C#

```
using C1.WPF|variable=WPF.DataGrid;
```

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

Visual Basic

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

C#

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

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

What You've Accomplished

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

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