ComponentOne

GridView for ASP.NET Web Forms

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

Website: http://www.componentone.com
Sales: sales@componentone.com
Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)

Trademarks

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

Warranty

ComponentOne warrants that the 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 $2 5 (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

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

Table of Contents

Overview

Display items from a data source in an interactive, fully customizable table with **GridView for ASP.NET Web Forms**. Interactively select, edit, delete, sort, and group data. Show data across multiple pages making it easy for end-users to page through data.

Help with ASP.NET Web Forms Edition

For information on installing **ComponentOne Studio ASP.NET Web Forms Edition**, licensing, technical support, namespaces and creating a project with the control, please visit Getting Started with ASP.NET Web Forms Edition.

Key Features

GridView for ASP.NET Web Forms allows you to interactively select, edit, delete, sort, and group data. **GridView for ASP.NET Web Forms** includes advanced features that enable developers to build intuitive, professional-looking Web applications quickly and easily:

- **Microsoft GridView Compatibility**
  GridView is fully compatible with Microsoft's GridView control, so you can upgrade existing ASP.NET applications easily and be productive immediately.
- **Filtering**
  GridView supports a built-in data-entry row below the column headers for custom end-user operations such as searching and filtering records. You can easily filter data by enabling the showFilter option.
- **Custom Grid Formatting**
  Intuitive formatting lets you customize your grids to get the look and functionally you want. Automatically create columns, display borders and gridlines, allow editing, sort data, and much more.
- **Editing**
  Make edits to your data in the grid and save or cancel changes.
- **Row Merging**
  Configure your grid to automatically merge together rows containing the same value. Your end result is a clean, well-organized grid.
- **Embed Input Controls**
  Enhance data entry and interactivity by incorporating Input for ASP.NET Web Forms controls in your grid. Embed the masked editor, date time editor, numeric editor, percent editor, and currency editor into any grid column.
- **Embed Controls and Images Into Any Grid Column**
  Display check boxes, list boxes, buttons, images, and more.
- **Code-free Grid Customization**
  Manage column collection, paging behavior, and edit the grid’s design without writing any UI code. With GridView's extensive design-time support, adding customized grid functionality to your Web site has never been easier.
- **Automatically Resize Columns and Rows**
  By setting a single property, GridView will automatically size column width and row height when the grid is scrolled to adjust to fit the longest text in a column. You can also enable or disable automatic sizing for specific columns and rows.
- **Bands**
  Column headers can span over multiple columns enabling you to create a hierarchical structure. For example, organize several column subheadings under another, broader column header.
- **Grouping and Aggregates**
  Configure your grid in outline mode to enable the end-user to collapse and expand the groups by clicking on the group headers. You can also determine whether groups will be initially collapsed or expanded. The grid will display expanded/collapsed icons next to each group header row. Grouped rows can display aggregate data such as counts or sums.
- **Reusable Templates**
  Save and load grid templates to create multiple grids with the same look and feel throughout a project and across projects.
- **Theming**
  With just a click of the SmartTag, change the GridView's look by selecting one of the 6 premium themes (Arctic, Midnight, Aristo, Rocket, Cobalt, and Sterling). Optionally, use ThemeRoller from jQuery UI to create a customized theme!
- **CSS Support**
  Use a cascading style sheet (CSS) style to define custom skins. CSS support allows you to match the grid to your organization's standards.

Quick Start

In this quick start you'll explore the functionality of **GridView for ASP.NET Web Forms**. In the following steps you'll create a simple bound grid application using the C1GridView control. You'll add the C1GridViewcontrol to a project, bind the control to a datasource, customize the grid's appearance and behavior, and explore some of the run-time interactions possible with the control.

Step 1 of 3: Binding C1GridView to a DataSource

In this step you'll begin the quick start by creating a new project and adding the C1GridView control to your project. You'll then bind the C1GridView control to a datasource. Note that in this example, you'll be using the Northwind database, **C1Nwind.mdb**, installed by default in the **ComponentOne Samples\Common** folder installed in your **Documents** folder.

Complete the following steps to begin:

1. From the Visual Studio **File** menu select **New** | **Project**. The **New Project** dialog box will appear.
2. In the **New Project** dialog box expand a language in the left-hand pane and select **Web**. In the right pane, choose **ASP.NET Empty Web Application**, enter a **Name** for your application, and select **OK**. A new application will be created.
3. In the Solution Explorer, right-click the project and choose **Add Reference**.
4. In the **Add Reference** dialog box, locate and select the **C1.Web.Wijmo.Controls** and **C1.Web.Wijmo.Controls.Design** assemblies and click **OK**. The references will be added.
5. Right-click the project in the Solution Explorer and from the context menu choose **Add** | **New Item**.
6. In the **Add New Item** dialog box choose **Web Form** from the list of templates, name the item "Default.aspx", and click **Add**. The new page should open.
7. In the Solution Explorer, right click the project name and choose **Add | New Folder**. Name the new folder "App_Data".
8. Navigate to the Visual Studio Toolbox and double-click the **C1GridView** icon to add the control to your project. Note that in source view, tag for C1GridView control is added within <form> tags.
9. In the Solution Explorer window, right-click the **App_Data** folder and select **Add Existing Item** in the context menu.
10. In the **Add Existing Item** dialog box, navigate to where the Northwind database is located, by default in the samples directory, select **C1Nwind.mdb**, and click **Add** to close the dialog box and add the file to your project.
11. Click the C1GridView control's smart tag to open the **C1GridView Tasks** menu.
12. Click the **Choose Data Source** drop-down arrow, and select **<New data source>**: The **Data Source Configuration Wizard** will open.
13. Configure your data source by completing the following steps:

    In Visual Studio 2012:

    a. On the **Choose a Data Source Type** screen, select **Database,** leave the default ID entered, and click **OK**.
    b. On the **Choose Your Data Connection** screen, click the drop-down arrow and select the **C1NWind.mdb** database.
    c. Click **Next** to continue.
    d. Click **Next** on the next screen to use the default connection string.

    In previous versions of Visual Studio:

    a. On the **Choose a Data Source Type** screen, select **Access Database**. Leave the default ID entered, and click **OK**.
    b. On the **Choose a Database** screen, click the **Browse** button to locate a database.
    c. In the **Select Microsoft Access Database** dialog box, click the **App_Data** folder in the **Project folders** list, select the **C1Nwind.mdb** file in the **Contents of folder** pane, and click **OK**. The **Nwind.mdb**

database should now be listed on the **Choose a Database** screen.
   d. Click **Next** to continue.
14. On the **Configure the Select Statement** screen, confirm that the **Specify columns from a table or view** radio button is selected, under **Name** choose **Products** from the drop-down list, and in the **Columns** pane choose the asterisk (*) check box to select all the columns. Click **Next** to continue.
15. You can test the query on the **Test Query** page, and select **Finish** to close the wizard and complete binding the grid. Note that the grid columns now reflect the datasource.

That's all you need to do to create a simple grid application! The grid is now bound Northwind database. If you run your project now, you'll have a fully-functional grid application where you can interact with data in a tabular form and you'll be able to access and interact with the data from the **Products** table of the database. In the next steps of this quick start you'll customize the grid's appearance and behavior and explore the grid's run-time interactions.

Step 2 of 3: Customizing the C1GridView Control

In the previous step of the quick start you created a simple grid application and bound the grid to a datasource. In this step you'll customize the initial grid application further by changing the grid's appearance and behavior settings.

Complete the following steps to continue:

1. Click once on the C1GridView control to select it and navigate to the Properties window.
2. Click the drop-down arrow next to the **Theme** property and select **arctic** in the list of themes. For more information, see the Themes topic.
3. Click the C1GridView control's smart tag, and in the **C1GridView Tasks** menu, select **Property builder**.
4. The **C1GridView Properties** dialog box will open.
5. On the **General** tab, select the **Allow sorting** check box. Under **Client-side** select **the Allow column moving** check box.
6. Click the **Paging** tab, and click the **Allow paging** check box. Note that, by default, **Bottom** is selected for the position of the navigation, and the **Numeric** mode is selected.
7. Click the **Columns** tab in the **C1GridView Properties** dialog box. Here you can add and remove columns and change column settings.
8. In the **Selected columns** list, choose any columns you wish to remove and press the  button. For example, remove the **ProductID**, **SupplierID**, **CategoryID**, **UnitsOnOrder**, and **ReorderLevel** columns.
9. Click **Apply** to apply your changes, and click **OK** to close the **C1GridView Properties** dialog box.

You've now completed binding the C1GridView control and customizing the grid's appearance and behavior settings. In the next, and final, step you'll run your application to observe the changes you made.

Step 3 of 3: Running the Application

In the previous steps of the quick start you created a simple grid application, bound the grid to a datasource, and customized the grid's appearance and behavior. In this step you'll run the grid application and explore some of the run-time interactions possible with the C1GridView control.

Complete the following steps to continue:

1. In Visual Studio, select **Debug | Start Debugging** to run your application.
   The grid will look similar to the following at run time:

ProductName	QuantityPerUnit	UnitPrice	UnitsInStock	Discontinued
Chai	10 boxes x 20 bags	18.00	39	☐
Chang	24 - 12 oz bottles	19.00	17	☐
Aniseed Syrup	12 - 550 ml bottles	10.00	13	☐
Chef Anton's Cajun Seasoning	48 - 6 oz jars	22.00	53	☐
Chef Anton's Gumbo Mix	36 boxes	21.35	0	☑
Grandma's Boysenberry Spread	12 - 8 oz jars	25.00	120	☐
Uncle Bob's Organic Dried Pears	12 - 1 lb pkgs.	30.00	15	☐
Northwoods Cranberry Sauce	12 - 12 oz jars	40.00	6	☐
Mishi Kobe Niku	18 - 500 g pkgs.	97.00	29	☑
Ikura	12 - 200 ml jars	31.00	31	☐
1 2 3 4 5 6 7 8				

2. Sort a column, by clicking in a column's header. In the following image.
3. Move a column by clicking on a column header once and dragging it to another location in the grid's header.
4. Page forward in the grid, by clicking once on a number page indicator at the bottom of the grid.

Congratulations! You've completed the **C1GridView** quick start. If you'd like to continue exploring **GridView for ASP.NET Web Forms**, see the Samples and take a look at the Task-Based Help topics.

GridView Top Tips

Grid Performance

The initialization process of the grid is slower if a large amount of data is displayed. You can reduce the page length to speed up the process by using the following options:

- **Setting paging and sorting:** Use Ajax calls instead of full post backs to implement paging and sorting.
- **Virtualization**: Grid supports virtual scrolling for both rows and columns. You can set the VirtualizationSettings property to row, column or both. This results in faster grid rendering and better performance. You can also refer to Row Virtual Scrolling and Column Virtual Scrolling samples.
- **Modify Scroll Settings**: When your grid is in the scroll mode, remove fixed columns and rows to enter a "light" scrolling mode, which works much faster.
- **Use EnableConditionalDependencies:** Set the EnableConditionalDependencies property to **True**. This property reduces the number of resource files required to be registered at the time of page load. By default, the property is set to **False**.

  EnableConditionalDependencies adds only those resource files (.js and .css) to the page that are required for a specific operation. A property, on which registration of a resource depends, is called **controlling** property.

  These controlling properties, including AllowPaging, ScrollingSettings, AllowC1InputEditors and ShowFilter, depend on different resources. For example, if you don't want to use paging in your grid, the resource files related to AllowPaging do not get registered on pageload, resulting in optimized browser payload and faster rendering.


This property works only when the EnableCombinedJavaScripts property is set to **True**.

Add GridView Manually

If you are not able to see the ASP.NET controls in the toolbox after installing ASP.NET Web Forms Edition, you can add them manually. To manually add the C1GridView control to the Visual Studio Toolbox:

1. Open Visual Studio.
2. From the **View** menu, select **Toolbox** and right-click to open the context menu.
3. Right-click the tab where the controls are added and select **Choose Items** from the context menu. The **Choose Toolbox Items** dialog box opens.
4. In the dialog box, select the **.NET Framework Components** tab. Sort the list by Namespace (click the Namespace column header) and check the check boxes for all components belonging to the namespace **C1.Web.Wijmo.Controls.C1GridView**.
5. From the Visual Studio Toolbox, double-click the control or drag it onto your form. This adds the C1GridView control to your web form.

Design-Time Support

GridView for ASP.NET Web Forms provides visual editing to make it easier to create a grid application. The following sections describe how to use **C1GridView's** design-time environment to configure the C1GridView control:

Tasks Menu

A smart tag represents a shortcut tasks menu that provides the most commonly used properties in each control. You can invoke each control's tasks menu by clicking on the smart tag (▶) in the upper-right corner of the control. For more information on how to use the smart tag in **C1GridView**, see C1GridView Smart Tag.

Context Menu

The context menu represents a shortcut menu that provides common Visual Studio commands, as well as commands specific to the C1GridView control. You can display the C1GridView context menu by right-clicking anywhere on the grid. For more information on how to use the context menu in C1GridView, see C1GridView Context Menu.

Property Builder

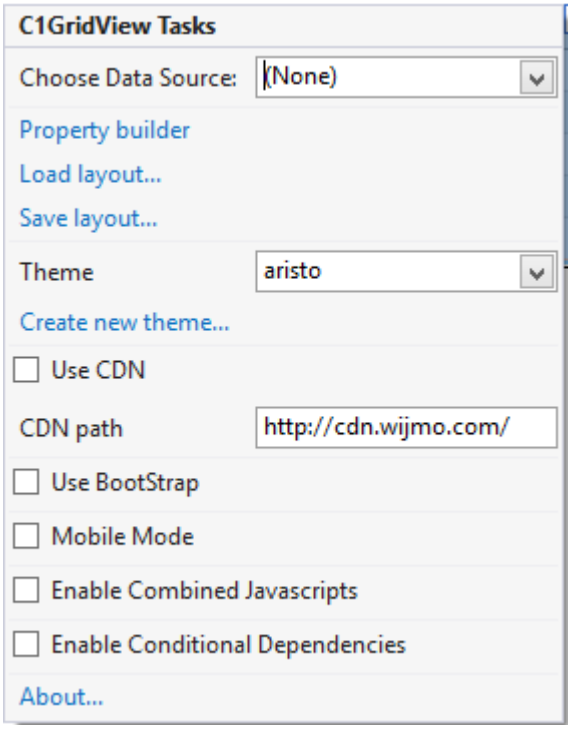
Quickly customize the appearance and behavior of the C1GridView control using the **C1GridView Properties** dialog box. For more information on how to access and use the **Property builder** in C1GridView, see Property Builder.

Properties Window

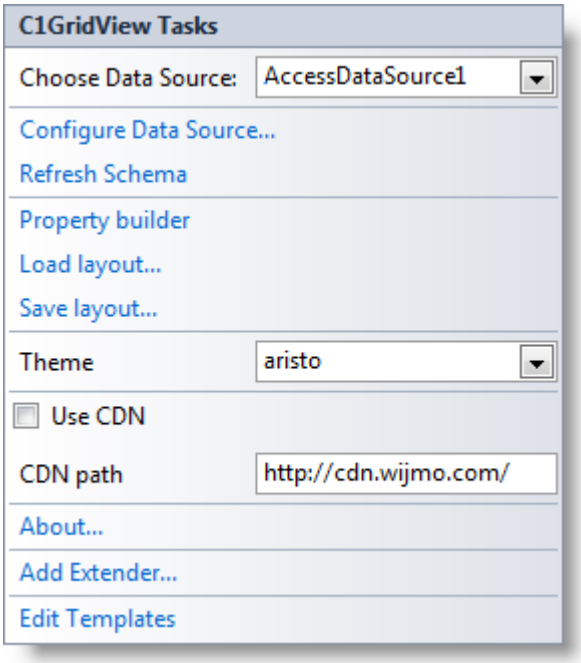
You can also easily configure C1GridView at design time using the Properties window in Visual Studio. You can access the Properties window by right-clicking the control and selecting **Properties**.

Smart Tag

A **smart tag** represents a short-cut **Tasks** menu that provides the most commonly used properties of a component. To access the **C1GridView Tasks** menu, click the **smart tag** in the upper-right corner of the C1GridView control. The **C1GridView Tasks** menu appears.



Note that when the grid is bound to a data source, the **Tasks** menu lists additional options and appears similar to the following:



The **C1GridView Tasks** menu operates as follows:

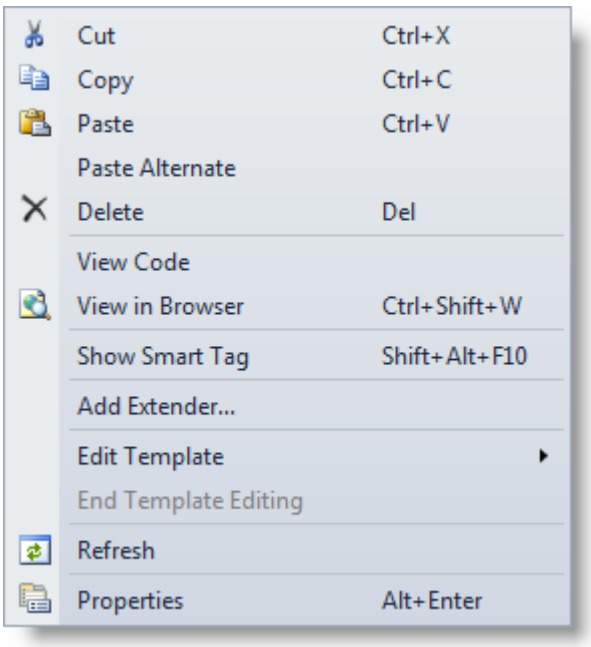
- **Choose Data Source**
  Choose an existing data source or create a new connection through the **Data Source Configuration Wizard**. See Binding the Grid to a Data Source for more information.
- **Configure Data Source**
  This option invokes the **Data Source Configuration Wizard** to configure the data source. This option only appears if the C1GridView control is bound to a data source.
- **Refresh Schema**
  This option refreshes the data source's schema. This option only appears if the C1GridView control is bound to a data source.
- **Property builder**
  Opens the **Property builder** where you can set properties and customize the grid. See Property Builder for more information.
- **Load Layout**
  Allows you to load an XML layout file. When you click this option the **Open** dialog box opens allowing you to select a file to load.
- **Save Layout**
  Allows you to save the layout file as an XML file. When you click this option the **Save** dialog box opens allowing you to select a file to load.
- **Theme**
  Clicking the **Theme** drop-down box allows you to select from various visual schemes. For more information about available visual styles, see Themes.
- **Create new theme...**
  The **Create new theme...** option opens **ThemeRoller for Visual Studio**. This allows you to customize a theme without leaving your development environment. To find more information on using **ThemeRoller** in your application, see ThemeRoller for Visual Studio.
- **Use CDN**
  Selecting the **Use CDN** check box will indicate that the widget extender must load client resources from a content delivery network. By default this box is not checked.
- **CDN Path**
  Indicates the path for the content delivery network. Enter a URL here to change the path.
- **Use Bootstrap**
  Selecting the **Use Bootstrap** option applies Bootstrap theming to your control. To find more information on

using **Bootstrap** theming in your application, see Bootstrap Theming.

- **Mobile Mode**
  Enables/Disables the mobile mode.
- **About**
  Clicking the **About** item displays a dialog box, which is helpful in finding the version number of **ASP.NET Web Forms Edition** and online resources.
- **Add Extender**
  Clicking the **Add Extender** item opens the **Extender Wizard**, allowing you to add an extender to the control.
- **Edit Templates**
  Clicking this option invokes **Template Editing Mode**.
- **Enable Combined Javascripts**
  Checking on this property combines all the required javascript files into one file.
- **Enable Conditional Dependencies**
  This property registers dependency resources according to the selected control's settings. It adds only those javascripts files on the page that are required to carry out a specific operation. This results in faster page load and optimized grid performance.

Context Menu

Right-click anywhere on the grid to display the C1GridView context menu, which is a context menu that Visual Studio provides for all .NET controls, although the C1GridView context menu has a few extra features.

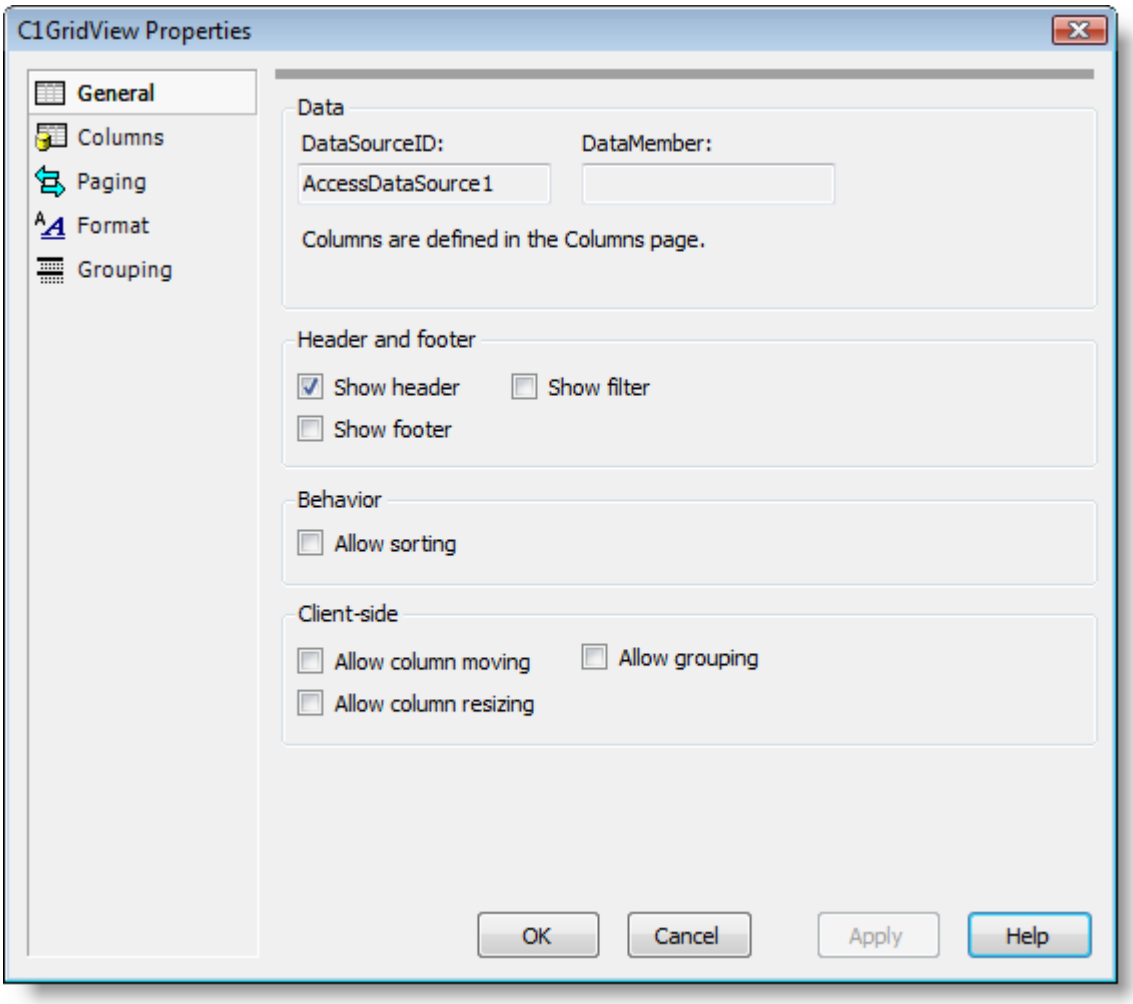


The context menu commands operate as follows:

- **Show Smart Tag**
  This option shows the smart tag for the C1GridView control. For more information on how to use the smart tag and available features, see Smart Tag.
- **Edit Template**
  Select **Edit Template** to edit the **EmptyData Template** or **Pager Template**. Choosing one of these options will invoke **Template Editing Mode**.
- **End Template Editing**
  This option ends **Template Editing Mode** if you're currently editing a template. If you're not in **Template Editing Mode**, this option will not be available.

Property Builder

The **Property builder** allows you to easily customize the appearance and behavior of the C1GridView control. To access the **Property builder**, select **Property builder** from the **C1GridView Tasks** menu (see C1GridView Smart Tag for details). The **C1GridView Properties** dialog box appears:

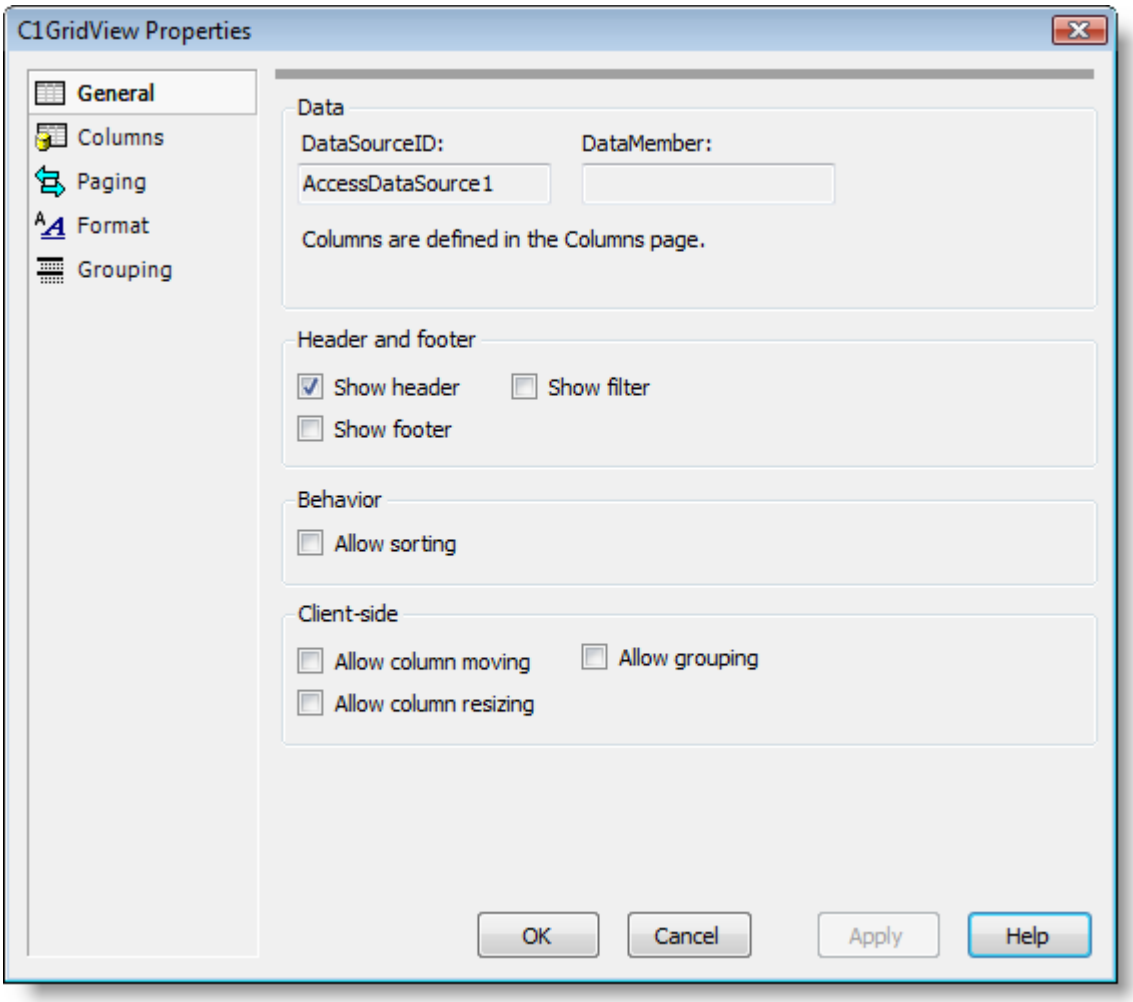


The **C1GridView Properties** window consists of five tabs:

Tab	Description
General	Displays the data source and allows you to set headers, footers, and sorting properties.
Columns	Allows you to specify the types of columns that appear and set the properties for each.
Paging	Lets you determine whether paging is used, and allows you to customize how and where it appears.
Format	Allows you to set the font, color and alignment of the grid and everything within it, including the headers, footers, the pager, specific items, and columns.
Grouping	Lets you set column grouping properties and determine how the group header and footer rows should be formatted and displayed.

General Tab

The **General** tab of the **Property builder** displays the data source and allows you to set headers, footers, and sorting properties. The **General** tab consists of **Data**, **Header and Footer**, and **Behavior** settings:



The **General** tab includes the following options:

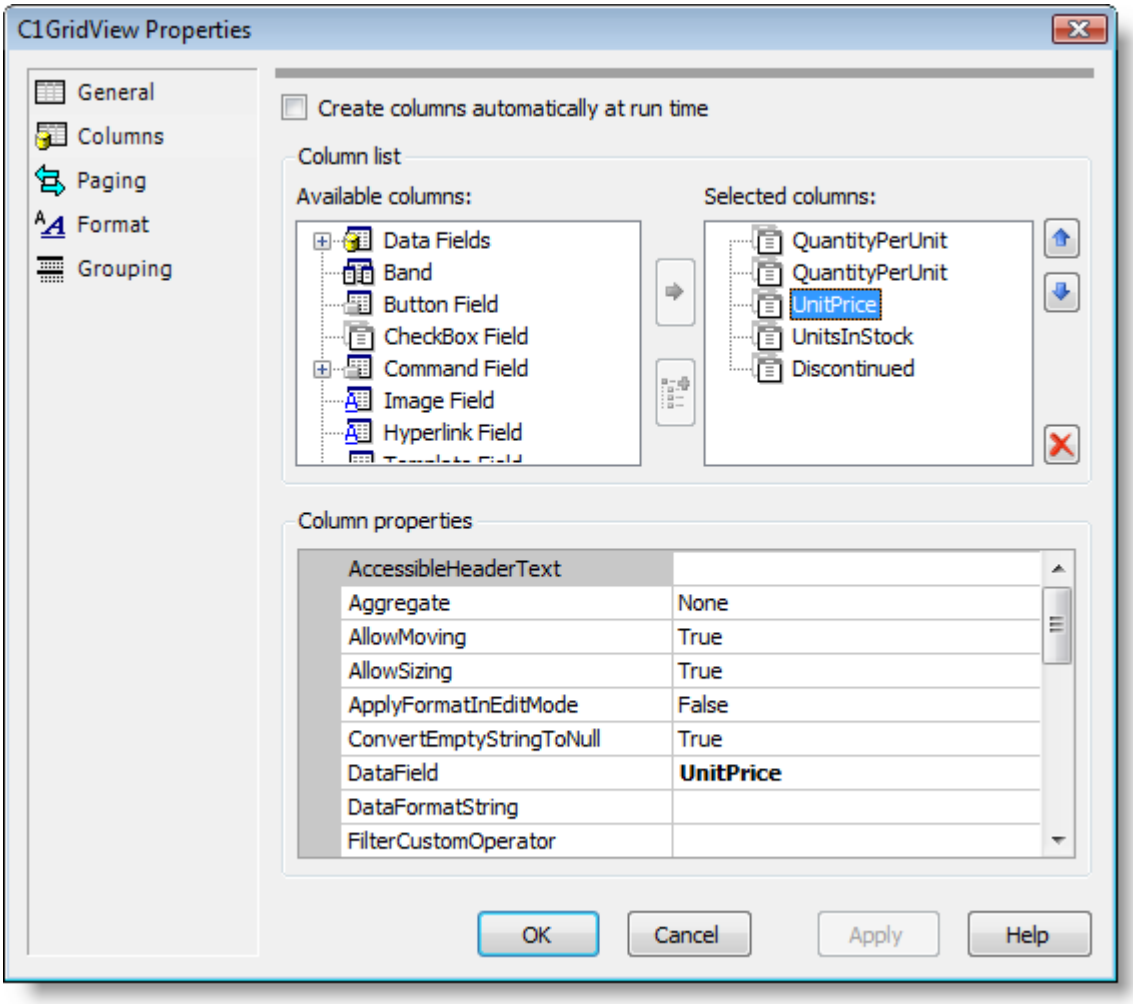
- **DataSourceID**:
  Displays the current data source. For information about data binding, see Binding the Grid to a Data Source.
- **DataMember**:
  Displays the current data member.
- **Show header**:
  When this check box is selected (default) the ShowHeader property is set to **True** and the grid's header is visible.
- **Show footer**:
  When this check box is selected the ShowFooter property is set to **True** and the grid's header is visible. For an example, see Setting the Footer Text. By default, this check box is not selected and ShowFooter is **False**.
- **Show filter**:
  Shows a filter row at the top of the grid. By default, this check box is not selected and ShowFilter is **False**.
- **Allow sorting**:
  When this check box is selected the AllowSorting property is set to **True** and users can sort the grid at run time. By default, this check box is not selected and AllowSorting is **False**. For more information about sorting, see the Sorting topic.
- **Allow column moving**:

When this check box is selected the AllowColMoving property is set to **True** and columns can be moved and reordered in the grid at run time. By default this check box is unchecked and AllowPaging is **False**.

- **Allow column resizing**:
  When this check box is selected the AllowColSizing property is set to **True** and columns can be resized in the grid at run time. See the Creating a Pageable Grid topic for an example. By default this check box is unchecked and AllowColSizing is **False**.
- **Allow grouping**:
  When this check box is selected the grouping area appears at the top of the grid. By default this check box is unchecked and grouping is not included.

Columns Tab

The **Columns** tab of the **Property builder** allows you to specify the types of columns that appear and set the properties for each. The **Columns** tab consists of **Column generation**, **Column list**, and **Column properties** sections:



The **Columns** tab includes the following options and sections:

- **Create columns automatically at run time**: When this check box is selected (default) the AutoGenerateColumns property is set to **True** and column objects are generated and displayed automatically. Note that when the grid is bound, this option may be automatically deselected.
- **Available columns**: This window lists the available column types. Available column types include:
  - **C1Band:** displays a banding column used to create multilevel column headers.
  - **C1BoundField:** displays a column bound to a field in a data source.
  - **C1ButtonField:** displays a command button for each item.

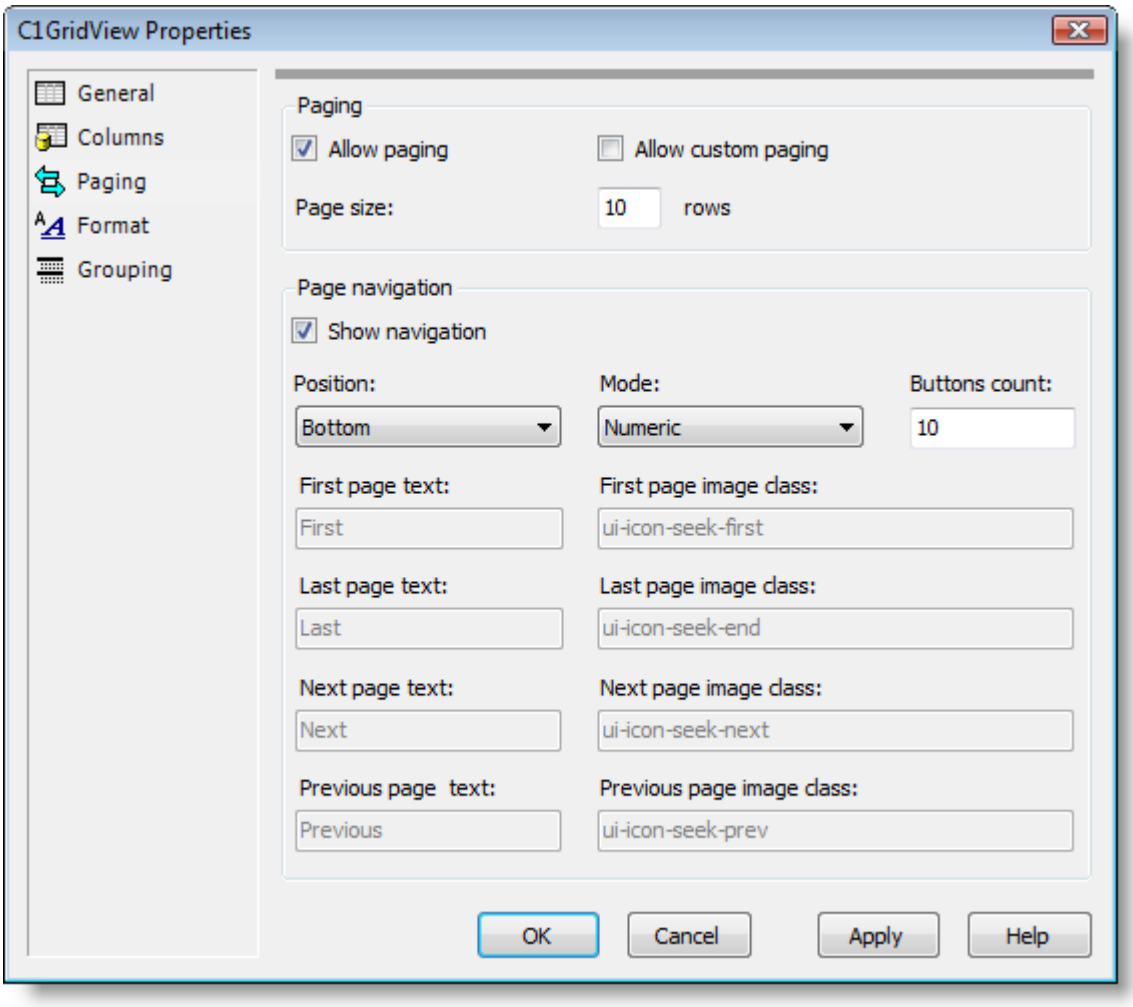
- **C1CheckBoxField:** displays a Boolean check box for each item.
- **C1CommandField:** displays a column containing editing commands for each item.
- **C1HyperLinkField:** displays each item as a hyperlink.
- **C1TemplateField:** displays each item in the column following a specified template.

When the grid is bound to a data source, the columns available from that data source will be listed under the **Data Fields** option in the **Available columns** window.

- **Selected columns**: This window lists the current selected columns. When the grid is bound to a data source, the bound columns will appear in this window.
  To add a columns to the **Selected columns** list, choose a column in the **Available columns** list and click the **Move** arrow button to move the item to the **Selected columns** list.
  To move the order of columns, select the **Up** and **Down** arrows to the right of the window. To remove a column from the list, click the **Delete** button to the right of the window.
- **Column properties**: This list displays the available properties for a column. To change a column's properties, click on that column in the **Selected columns** list and click the text box or drop-down arrow next to a property in the **Column properties** window to modify its value.

Paging Tab

The **Paging** tab of the Property builder allows you to determine whether paging is used and lets you customize how and where it appears. For more information about paging, see the Paging topic. The **Paging** tab includes the **Paging** and **Page Navigation** sections:

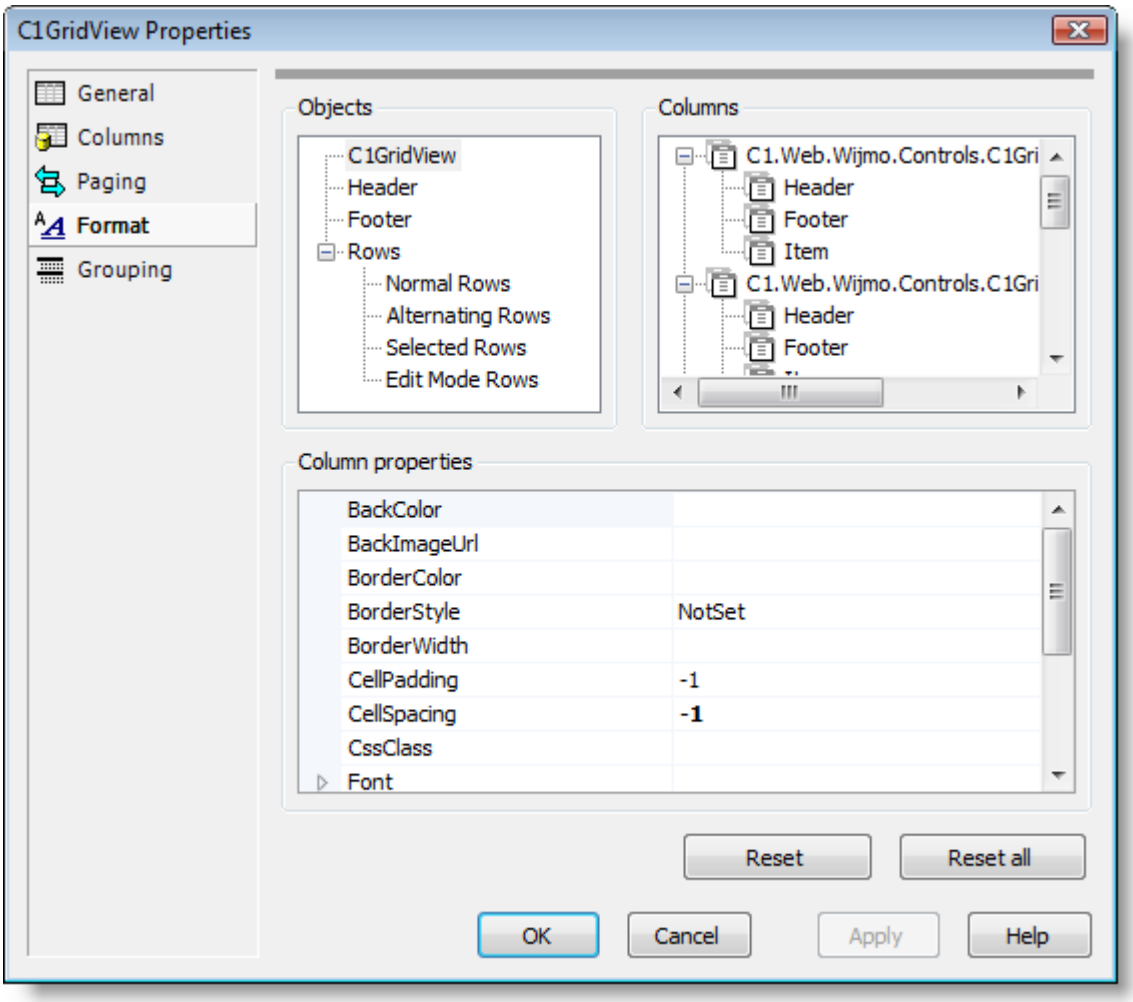


The **Paging** tab includes the following options:

- **Allow paging**: When this check box is selected the AllowPaging property is set to **True** and paging is included in the grid. See the Creating a Pageable Grid topic for an example. By default this check box is unchecked and AllowPagingis **False**.
- **Allow custom paging**: When the **Allow paging** check box is selected, this option is available. When this check box is selected the AllowCustomPaging property is set to **True** and custom paging is allowed. By default this check box is unchecked and AllowCustomPaging is **False**.
- **Page size**: When the **Allow paging** check box is selected, this option is available. The **Page size** text box sets the PageSize property and allows you to choose how many items are visible on the grid on each page. By default, PageSize is **10**.
- **Show navigation**: This option is available when the **Allow paging** check box is selected. When this check box is selected (default) the Visible property is set to **True** and paging controls appear on the grid.
- **Position**: This option is available when the **Allow paging** and **Show navigation** check boxes are selected. This option sets the Position property and lets you determine where the pager controls will appear in the grid. Options include:
  - **Bottom**: Paging controls will appear at the bottom of the grid. This is the default setting.
  - **Top**: Paging controls will appear at the top of the grid.
  - **TopAndBottom**: Paging controls will appear both at the top and bottom of the grid.
- **Mode**: This option is available when the **Allow paging** and **Show navigation** check boxes are selected. This option sets the Mode property and lets you determine what kind of pager controls should appear on the grid. Options include:
  - **NextPrevious**: **Previous** and **Next** button pager controls will used.
  - **Numeric**: Numbered link button pager controls that allow users to access pages directly will be used . This is the default setting.
  - **NextPreviousFirstLast**: **Previous**, **Next**, **First**, and **Last** button pager controls will be used.
  - **NumericFirstLast**: **Numbered** and **First**, and **Last** button pager controls will be used.
- **Buttons count**: This option is only available when the **Numeric** or **NumericFirstLast** Mode is selected and it sets PageButtonCount property and determines the number of page buttons to display in the pager control. By default, PageButtonCount is **10**.
- **First page text**, **First page image class**, **Last page text**, and **Last page image image class**: These options are only available when the **NextPreviousFirstLast** or **NumericFirstLast** Modeis selected:
  - The **First page text** option sets the FirstPageText property and determines the text to display for the **First** page button.
  - The **First page image class** option sets the **C1GridViewPagerSettings.FirstPageImageUrl** property and determines the image to display for the **First** page button. Click the button next to the **First page image class** text box to select an image.
  - The **Last page text** option sets the LastPageText property and determines the text to display for the **Last** page button.
  - The **Last page image class** option sets the **C1GridViewPagerSettings.LastPageImageUrl** property and determines the image to display for the **Last** page button. Click the button next to the **Last page image class** text box to select an image.
- **Next page text**, **Next page image class**, **Previous page text**, and **Previous page image class**: These options are only available when the **NextPrevious** or **NextPreviousFirstLast** Modeis selected:
  - The **Next page text** option sets the NextPageText property and determines the text to display for the **Next** page button.
  - The **Next page image class** option sets the **C1GridViewPagerSettings.NextPageImageUrl** property and determines the image to display for the **Next** page button. Click the button next to the **Next page image class** text box to select an image.
  - The **Previous page text** option sets the PreviousPageText property and determines the text to display for the **Previous** page button.
  - The **Previous page image class** option sets the **C1GridViewPagerSettings.PreviousPageImageUrl** property and determines the image to display for the **Previous** page button. Click the button next to the **Previous page image class** text box to select an image.

Format Tab

The **Format** tab of the **Property builder** allows you to set the font, color and alignment of the grid and everything within it, including the headers, footers, the pager, specific items, and columns. The **Format** tab includes the **Objects**, **Columns**, and **Column properties** sections:

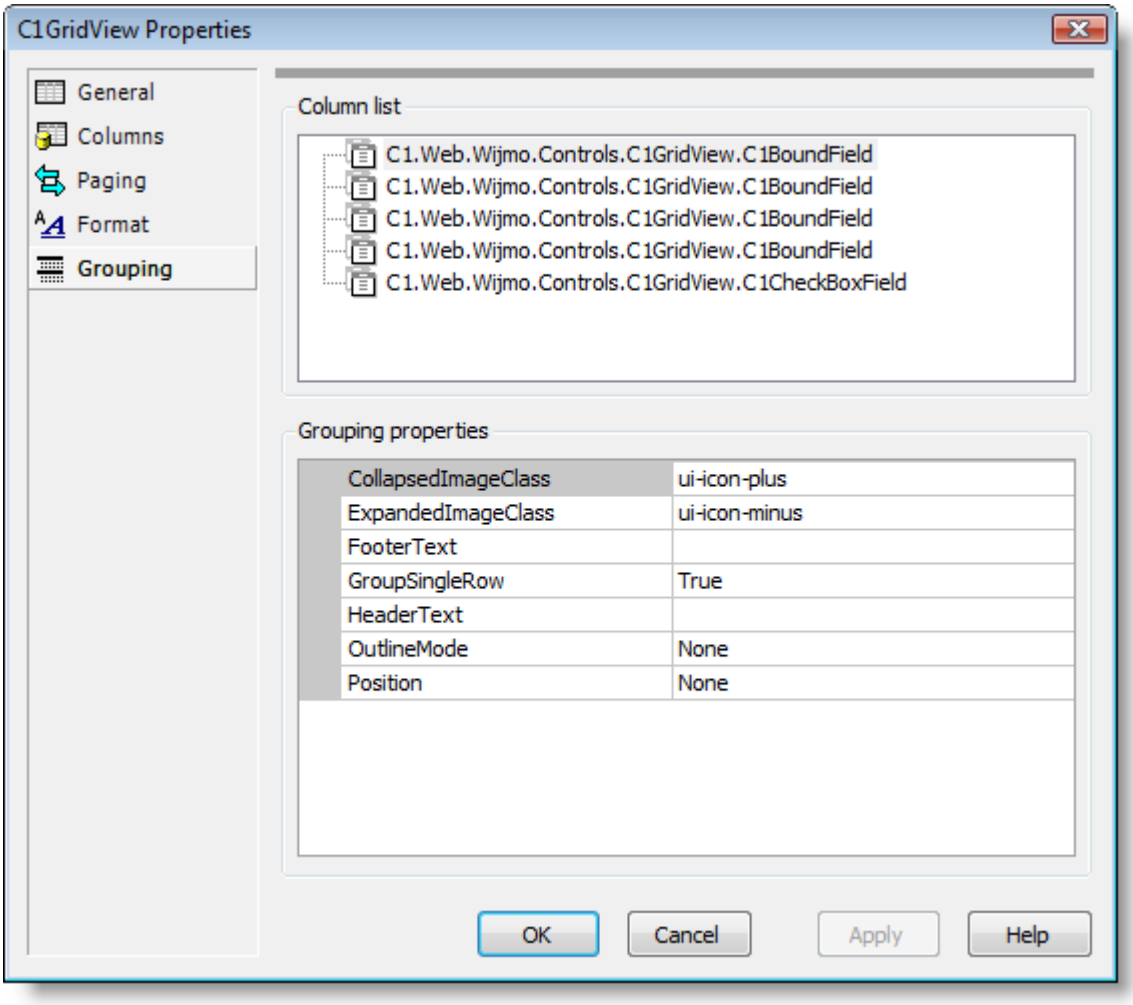


The **Format** tab includes the following sections:

- **Objects:** Displays the available objects. Choose a object to set it's formatting properties in the **Column properties** window.
- **Columns:** Displays the list of available columns. If the grid is bound, the bound columns will appear in this list. Choose a column to set it's formatting properties in the **Column properties** window.
- **Column properties:** When an object or column is selected from the appropriate pane, the **Column properties** window displays the available formatting properties that can be set for that object or column.

Grouping Tab

The **Grouping** tab of the **Property builder** allows you to set column grouping properties and determine how the group header and footer rows should be formatted and displayed. For more information about grouping, see the Grouping topic. The **Grouping** tab includes the **Column list** and **Grouping properties** sections:



The **Grouping** tab includes the following sections:

- **Column list**: Displays the list of available columns. If the grid is bound, the bound columns will appear in this list. Choose a column to set it's grouping properties in the **Grouping properties** window.
- **Grouping properties**: When an column is selected from **Column list**, the **Grouping properties** window displays the available grouping properties that can be set for that column. You can customize grouping behavior and set how the group header and footer rows should be formatted and displayed.

Working with GridView for ASP.NET Web Forms

GridView for ASP.NET Web Forms allows you to select, edit, delete, filter, and sort the items displayed in the table generated by the C1GridView component. **GridView for ASP.NET Web Forms** also supports paging, so data can be displayed across multiple pages, which are accessed by default or customized navigation buttons.

The columns of a table created using the C1GridView component correspond to the fields in a data source. You can control which columns are displayed, the types of columns to display, and the appearance of the whole table.

Using the AutoGenerateColumns property, you can generate columns automatically, manually, or both. Setting this property to **True** (default) creates a C1BoundField for each field in the data source. Setting this property to **False** allows you to specify the columns to display, which are added to the **Columns** collection.

> **Note:** Explicitly declared columns are rendered first, followed by automatically generated columns. Automatically generated columns are not added to the **Columns** collection.

Class Hierarchy

The following list summarizes the class relationships between the more important classes included in the **GridView for ASP.NET Web Forms**:

- C1.Web.Wijmo.Controls.C1GridView.C1GridView : System.Web.UI.WebControls.WebControl
  Encapsulates most of the grid functionality. This component is shown in the Visual Studio's Toolbox.
- C1.Web.Wijmo.Controls.C1GridView.C1Field : System.Object
  Base class for columns.
- C1.Web.Wijmo.Controls.C1GridView.C1GridViewRow : System.Web.UI.WebControls.TableRow
  Grid items.
- C1.Web.Wijmo.Controls.C1GridView.C1GridViewRowCollection : System.Collections.CollectionBase
  Collection of items.
- C1.Web.Wijmo.Controls.C1GridView.PagerStyle : System.Web.UI.WebControls.TableItemStyle
  Style of the Pager. Determines if and how the pager is displayed.
- C1.Web.Wijmo.Controls.C1GridView.GroupInfo : System.Object
  Used to determine how and where grouped header and footer rows are displayed.

Themes

C1GridView includes themes allowing you to easily change the control's appearance. The control includes several built-in themes allowing you to customize the control's appearance to your application. You can easily change themes from the **C1GridView Tasks** menu, from the Properties window, and in code. For more information on changing themes see the Theming.

Changing Themes

You can change the theme of a **C1GridView** at design time using the Properties window:

1. Click the C1GridView control once to select it, and navigate to the Properties window.
2. In the Properties window, click the **Theme** drop-down arrow and select a style, for example **rocket**.

The theme you selected is applied to your grid.

Included Themes

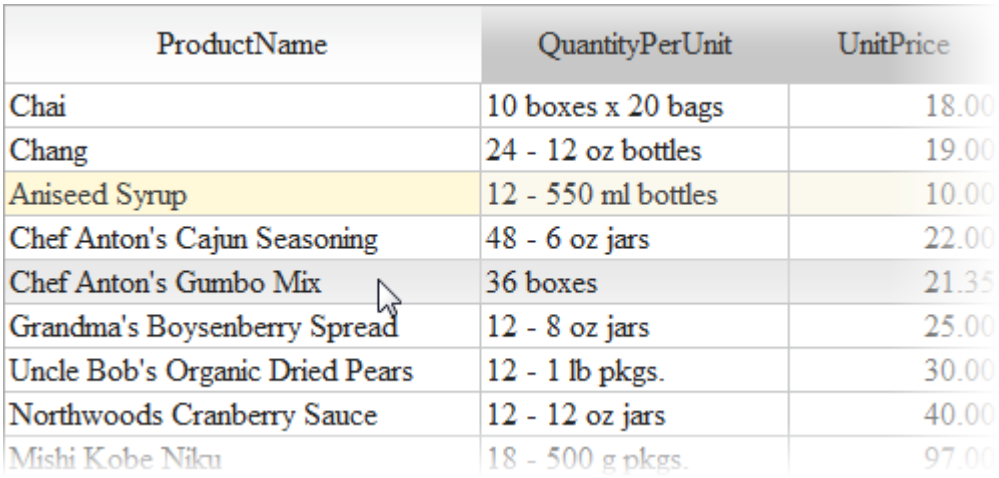
The following themes are included in **GridView for ASP.NET Web Forms**:

- arctic
- aristo
- cobalt

- midnight
- rocket
- sterling

arctic

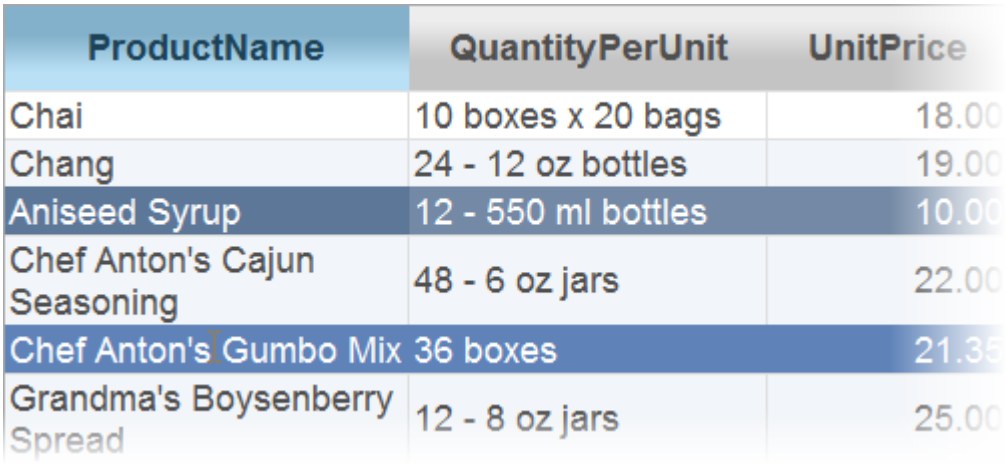
The following image displays the **arctic** theme:



ProductName	QuantityPerUnit	UnitPrice
Chai	10 boxes x 20 bags	18.00
Chang	24 - 12 oz bottles	19.00
Aniseed Syrup	12 - 550 ml bottles	10.00
Chef Anton's Cajun Seasoning	48 - 6 oz jars	22.00
Chef Anton's Gumbo Mix	36 boxes	21.35
Grandma's Boysenberry Spread	12 - 8 oz jars	25.00
Uncle Bob's Organic Dried Pears	12 - 1 lb pkgs.	30.00
Northwoods Cranberry Sauce	12 - 12 oz jars	40.00
Mishi Kobe Niku	18 - 500 g pkgs.	97.00

aristo

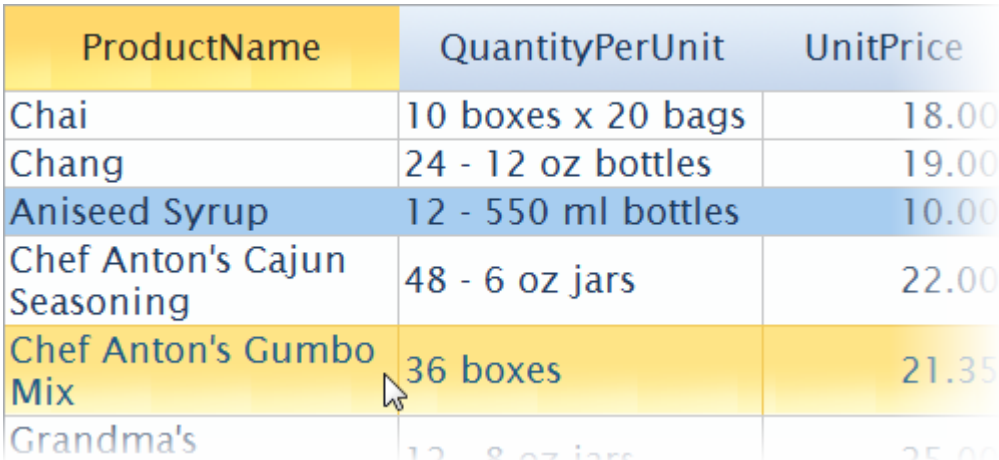
The following image displays the **aristo** theme. This is the default appearance of C1GridView.



ProductName	QuantityPerUnit	UnitPrice
Chai	10 boxes x 20 bags	18.00
Chang	24 - 12 oz bottles	19.00
Aniseed Syrup	12 - 550 ml bottles	10.00
Chef Anton's Cajun Seasoning	48 - 6 oz jars	22.00
Chef Anton's Gumbo Mix	36 boxes	21.35
Grandma's Boysenberry Spread	12 - 8 oz jars	25.00

cobalt

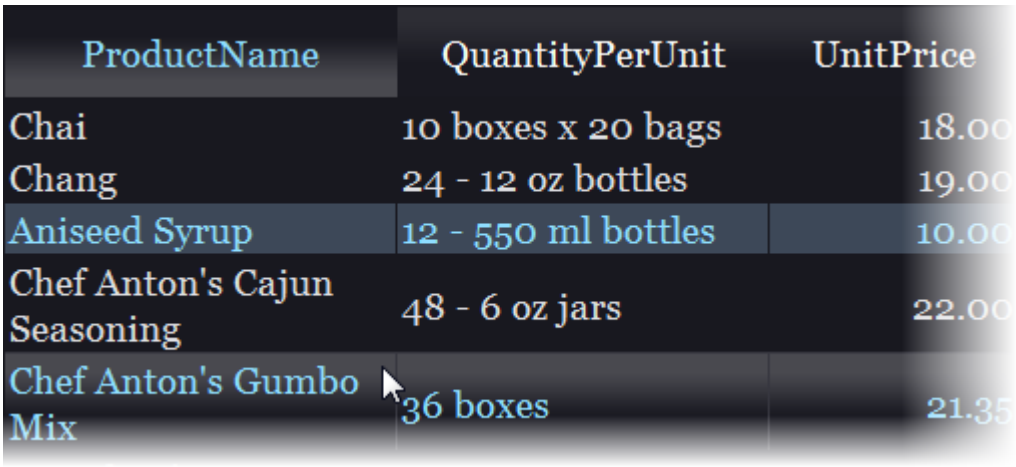
The following image displays the **cobalt** theme:



ProductName	QuantityPerUnit	UnitPrice
Chai	10 boxes x 20 bags	18.00
Chang	24 - 12 oz bottles	19.00
Aniseed Syrup	12 - 550 ml bottles	10.00
Chef Anton's Cajun Seasoning	48 - 6 oz jars	22.00
Chef Anton's Gumbo Mix	36 boxes	21.35
Grandma's	12 - 8 oz jars	25.00

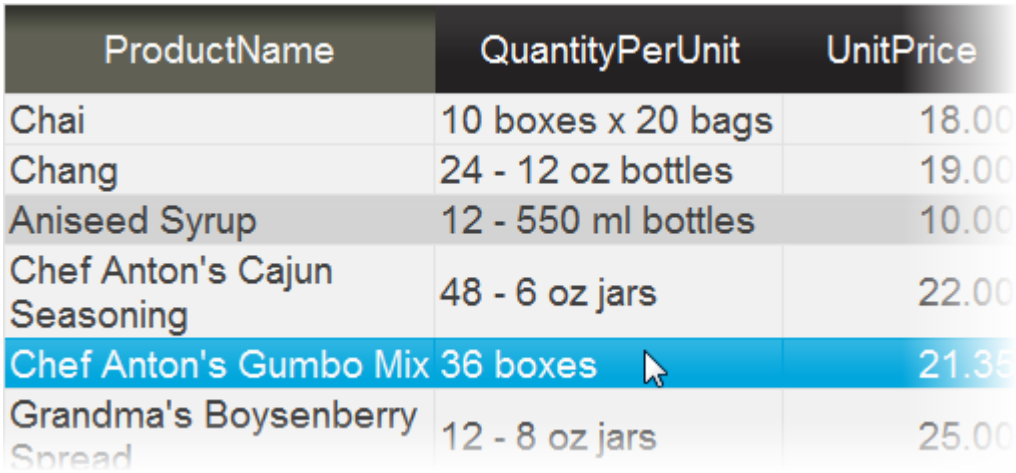
midnight

The following image displays the **midnight** theme:



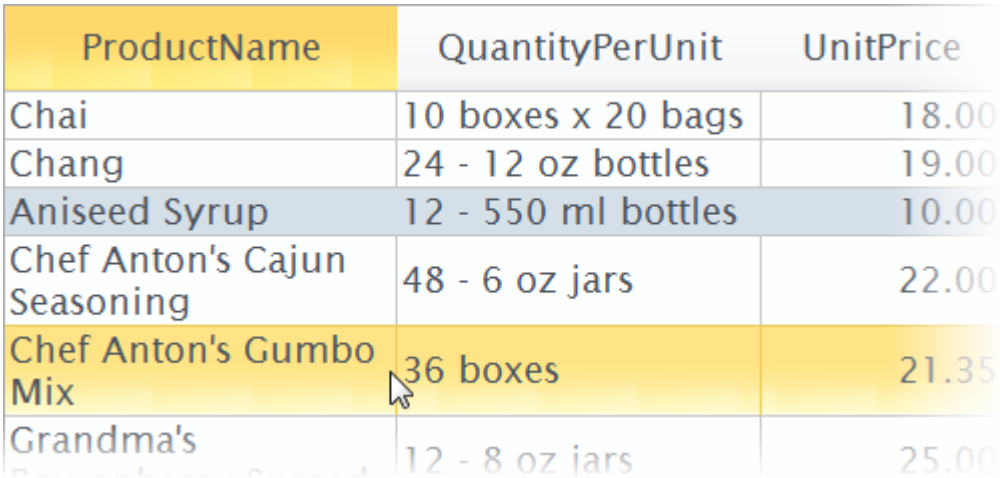
rocket

The following image displays the **rocket** theme:



sterling

The following image displays the **sterling** theme:



Keyboard Navigation

At run time you can navigate through the grid using keyboard navigation. Available keyboard navigation is similar to products like Microsoft Excel, to make it easier for users to intuitively navigate the grid. Keyboard navigation also allows greater accessibility in navigating and manipulating the grid.

The following table describes the possible interactions:

Option	Description
Up Arrow	Moves the focus to one cell up.
Down Arrow	Moves the focus to one cell down.
Left Arrow	Moves the focus by one cell to the left.
Right Arrow	Moves the focus by one cell to the right.
Home	Moves the focus to the first cell of the current row.
End	Moves the focus to the last cell of the current row.
Page Up	Moves the focus to the first cell of the current column.
Page Down	Moves the focus to the last cell of the current column.
Tab	Moves the focus to the next cell of the current row. When last cell of the grid is in focus, Tab key press moves the focus to the first cell of the GridView. When KeyActionTab property of **C1GridView** is set to **MoveAcrossOut**, the focus shifts from last cell to the next control on pressing the Tab key.

Editing Rows

Each row in the C1GridView component represents a record in the data source. You can select, edit, delete, and apply different styles to your rows.

If you are creating data manually, by adding a C1ButtonField and code for the **C1GridView.RowEditing**, **C1GridView**.**UpdateCommand** and **C1GridView.RowCancelingEdit** events, you can select each row to be edited or deleted. If you connect to the data source control, this is all taken care for you automatically.

The EditRowStyle property allows you to determine the appearance of the rows. For additional information on editing rows, see GridView for ASP.NET Web Forms Task-Based Help.

The RowMerge property allows you to merge rows containing identical text. You can use the **Property builder** to specify which rows to merge and how they are merged. For additional information, see Merging Rows.

You can use AJAX to update the grid when the user edits a record at run time by setting the Action property. For more information, see Editing a Record.

Grouping

The C1GridView data grouping features allow you to automatically calculate data aggregates and create trees with nodes that can be collapsed and expanded.

If ShowGroupArea is set to **True**, a grouping area will appear at the top of the grid:

To group at run time, simply drag a column header into the grouping area. Note that you may need to set the AllowColMoving property to True to allow run-time column manipulation.

The C1GridView data grouping features are controlled by two properties of the C1Field class:

- The GroupInfo property determines whether a group should be created for the column, as well as the

appearance, position, and content of the group header and group footer rows.
- The Aggregate property determines the type of aggregate that should be calculated for this column and included in the group header and footer rows.

Typically, you will set the GroupInfo property for columns that contain categories that you would like to group on (for example, *OrderYear*, *Country*, and so on). You will set the Aggregateproperty for columns that contain the values that you want to aggregate on (for example, *Amount*, *Sales*, *Expenses*, and so on).

You can customize the text that appears in the grouping area by setting the GroupAreaCaption property.

Setting Grouping Properties at Design Time

You can set up the **Grouping** properties at design time using the **Property builder**. You will use three tabs in the **C1GridView Properties** dialog box to set up each aspect of data grouping:

- **Grouping tab:** This tab contains a list of columns. For each column that you want to group on, set the Position property to determine where the group header and footer rows should be displayed (set Position to None to remove grouping for a column). Also set the HeaderText (and/or FooterText) property to determine what will be displayed in the group header/footer rows. Optionally, set up the group header and footer styles.
  See Grouping Tab for more information.
- **Columns tab:** Set the Aggregate property on each column that you want to aggregate on. Usually, these will be columns that contain numeric values, and will not be grouped on. See Columns Tab for more information.
- **Format tab:** Select each column that you are grouping on and set the styles for the group header and group footer rows. Typically, you will set the background color so the group headers and footers stand out from the regular rows. See Format Tab for more information.

For additional information on the **Property builder**, see Property Builder.

Collapsing and Expanding Groups

GridView for ASP.NET Web Forms generates client-side script that allows users to collapse and expand the groups without round trips to the server.

If you set the OutlineMode property to **StartCollapsed** or **StartExpanded**, the grid will display expanded/collapsed icons next to each group header row. The users can click these icons to collapse and expand the groups to see the overall structure or the details of the data.

The behavior for the outline nodes is as follows:

- **Click:**
  - If the node is currently expanded, it becomes collapsed. All subordinate group headers and data are hidden.
  - If the node is currently collapsed, it becomes expanded. If there are any subordinate group headers, they are displayed in the collapsed state. If there are no subordinate group headers, the data is displayed.
- **SHIFT-Click:**
  - All nodes and data below the node that was clicked are displayed.

You can select different icons for the group headers using the CollapsedImageClass and ExpandedImageClass properties.

For additional information on grouping, see the GridView for ASP.NET Web Forms Task-Based Help.

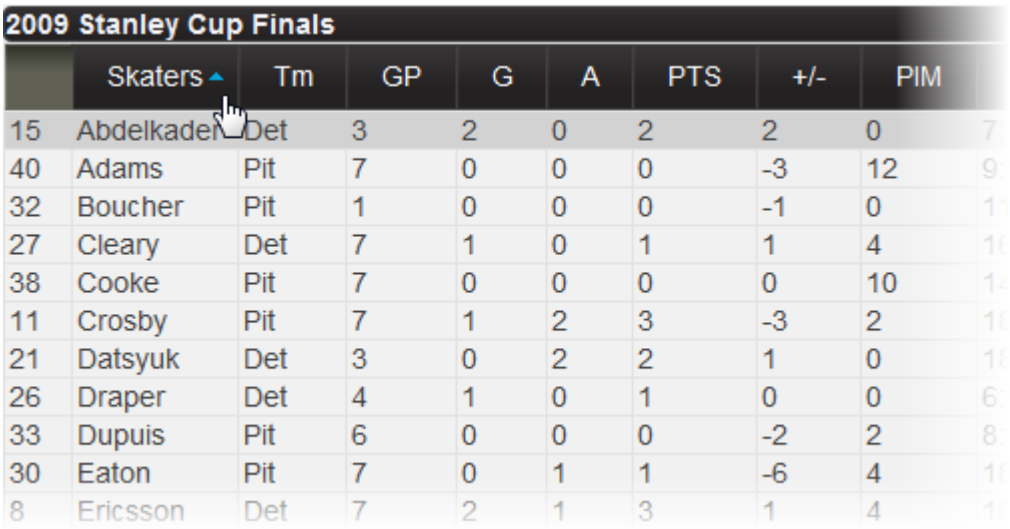
Sorting

The C1GridView control provides built-in sorting functionality without requiring any coding. You can further customize the sort functionality of the C1GridView control by using custom SortExpression property values for columns as well as by using the Sorting and Sorted events.

You can enable the default sorting behavior in the C1GridView control by setting its AllowSorting property to **True**. When AllowSorting is **True**, the C1GridView control renders a **LinkButton** control in the column headers and implicitly sets the SortExpression property of each column to the name of the data field to which it is bound. For example, if the grid contains a column that displays the *City* column of the **Employees** table in the Northwind sample database, the SortExpression of that column will be set to **City**. You can determine or set the sort direction with the SortDirection property (**Ascending**, **Descending**, or **None**).

At run time, users can click the **LinkButton** control in a column heading to sort by that column. Clicking the link causes the page to perform a postback and raises the C1GridView control's Sorting event. After the query has executed, the grid's Sorted event is raised. Finally, the data source control rebinds the C1GridView control to the results of the resorted query. When a column is sorted, the column header will display a sort indicator at run time (an arrow icon) indicating the direction of the sort.

In the following image, the *Skaters* column was sorted:



2009 Stanley Cup Finals

	Skaters ▲	Tm	GP	G	A	PTS	+/-	PIM
15	Abdelkader	Det	3	2	0	2	2	0
40	Adams	Pit	7	0	0	0	-3	12
32	Boucher	Pit	1	0	0	0	-1	0
27	Cleary	Det	7	1	0	1	1	4
38	Cooke	Pit	7	0	0	0	0	10
11	Crosby	Pit	7	1	2	3	-3	2
21	Datsyuk	Det	3	0	2	2	1	0
26	Draper	Det	4	1	0	1	0	0
33	Dupuis	Pit	6	0	0	0	-2	2
30	Eaton	Pit	7	0	1	1	-6	4
8	Ericsson	Det	7	2	1	3	1	4

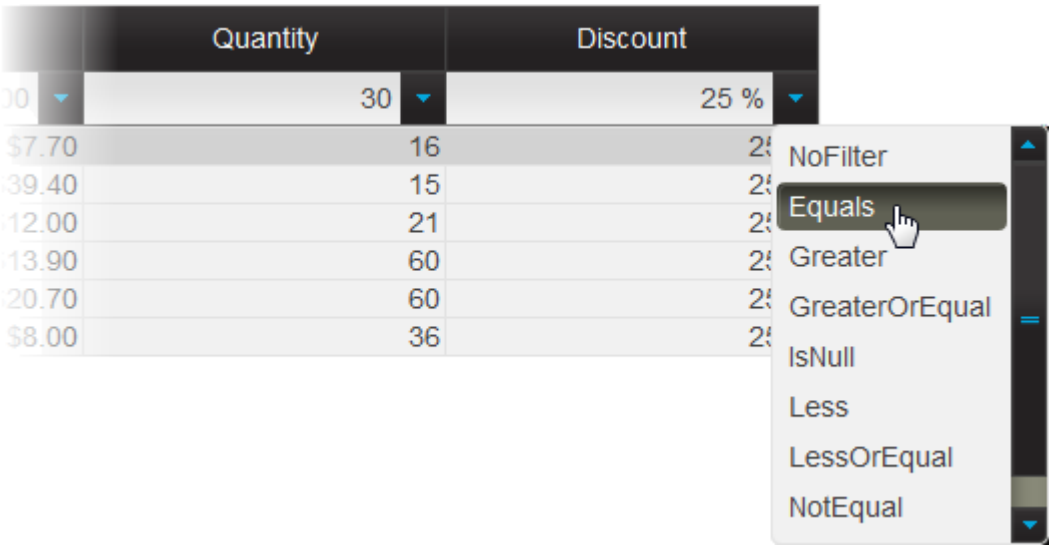
Note that if the data source control does not support sorting and a sorting operation is performed in the C1GridView control, the C1GridView control throws the **NotSupportedException** exception. You can catch this exception in a handler for the Sorting event and check the data source to determine whether sorting is supported, or by using your own sorting logic.

You can use AJAX to update the grid when the user sorts a column at run time by setting the Action property. For an example, see Sorting Columns.

Filtering

The C1GridView control provides built-in data filtering functionality without requiring any coding. To be able to filter data at run time, the ShowFilter property must be set to **True** (by default this is **False**). When ShowFilter is **True** a filter bar appears at the top of the grid just under the grid's column headings.

To filter data in the grid at run time, simply type text in the filter bar, click the drop-down arrow in the filter row in the column you are filtering, and select a filter type from the menu. To remove the filter, click the drop-down arrow in the filter row in the column you are filtering and select **NoFilter**. In the following image, the *Discount* column has been filtered so entries equal to "25%" appear:



You can choose what the filter condition should be by setting the FilterOperator property (note that you can set this property in the **Columns** tab of the **Property builder**). By default FilterOperator is set to **Contains**, but you can set the FilterOperator to the following options:

Option	Description
NoFilter	No filter is applied.
Contains (default)	The filter term is contained in the column.
NotContain	The filter term is not contained in the column.
BeginsWith	Items in the column begin with the filter term.
EndsWith	Items in the column end with the filter term.
Equals	Items in the column equal the filter term exactly.
NotEqual	Items in the column do not equal the filter term exactly.
Greater	Items in the column are less than the (numeric) filter term.
Less	Items in the column are greater than the (numeric) filter term.
GreaterOrEqual	Items in the column are greater than or equal to the (numeric) filter term.
LessOrEqual	Items in the column are less than or equal to the (numeric) filter term.
IsEmpty	Items in the column are empty.
NotIsEmpty	Items in the column are not empty.
IsNull	Items in the column are null.
NotIsNull	Items in the column are not null.
Custom	Custom filter is applied.

You can customize the appearance of the filter bar by setting the FilterStyle property. To show a filter button, you should either set AutoGenerateFilterButton to **True** or manually create a C1CommandField and set its ShowFilterButton property to **True**. If no filter button is visible, the user can trigger the filter by pressing the ENTER key.

You can use AJAX to update the grid when the user filters columns at run time by setting the Action property. For

more information, see Filtering Columns.

Filtering in C1BoundField and C1TemplateField Columns

The C1GridView control supports filtering in both C1BoundField and C1TemplateField columns through the following available properties:

- DataField

  It is used to specify the data field to which the filter setting is applied. Its type is string.

- FilterOperator

  It is used to specify the filter operator. Its type is the enum of **FilterOperator**. It only works when **DataField** is set.

- FilterValue

  It is used to specify the filter value. Its type is string. It only works when DataField is set.

- FilterCustomOperator

  It is used as filter operator when the **FilterOperator** property is equal to **FilterOperator.Custom**. Its type is string. It only works when **DataField** is set.

- FilterStyle

  It is used to provide a custom style for the filter item. Its type is TableItemStyle. It only works when **DataField** is set.

- ShowFilter

  It is used to indicate whether filter text box should be shown in the filter row. It allows end-user filtering if the **DataField** property is set. Its type is bool. It only works when **DataField** is set.

- FilterExpression

  It can contain the filter conditions that include the filter settings for multiple fields. The user can set the text of the filter conditions to this property to apply a complex filter. Its type is string. It works when **DataField** is not set.

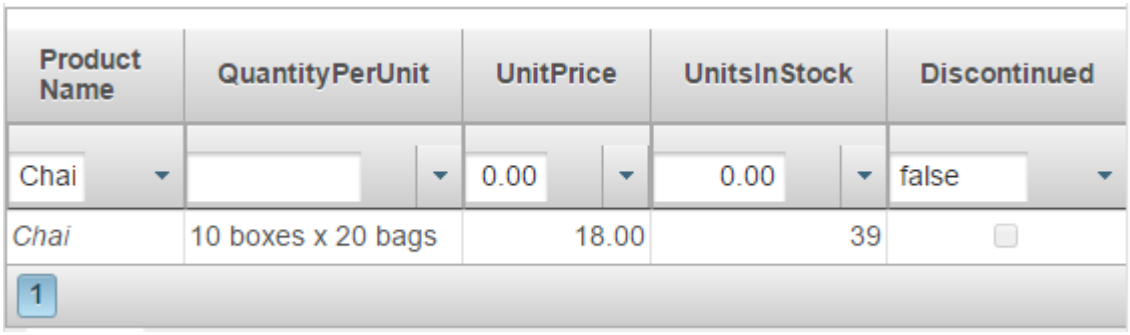
The following code illustrates filtering for **C1TemplateField** setting **ShowFilter** and **FilterValue** in the grid.

Source View

```
    <Columns>
           <cc1:C1TemplateField DataField="ProductName" HeaderText="Product Name"
FilterValue="Chai">
            <ItemTemplate>
                    <asp:Label runat="server" Font-Italic="true" ShowFilter=true
ID="lbl1" Text='<%# Bind("ProductName") %>'> </asp:Label>
            </ItemTemplate>
        </cc1:C1TemplateField>
        <cc1:C1BoundField DataField="QuantityPerUnit" HeaderText="QuantityPerUnit"
SortExpression="QuantityPerUnit">
        </cc1:C1BoundField>
        <cc1:C1BoundField DataField="UnitPrice" HeaderText="UnitPrice"
```

```
SortExpression="UnitPrice">
        </cc1:C1BoundField>
        <cc1:C1BoundField DataField="UnitsInStock" HeaderText="UnitsInStock"
SortExpression="UnitsInStock">
        </cc1:C1BoundField>
        <cc1:C1CheckBoxField DataField="Discontinued" HeaderText="Discontinued"
SortExpression="Discontinued">
        </cc1:C1CheckBoxField>
    </Columns>
```

The output for the above code is as shown:

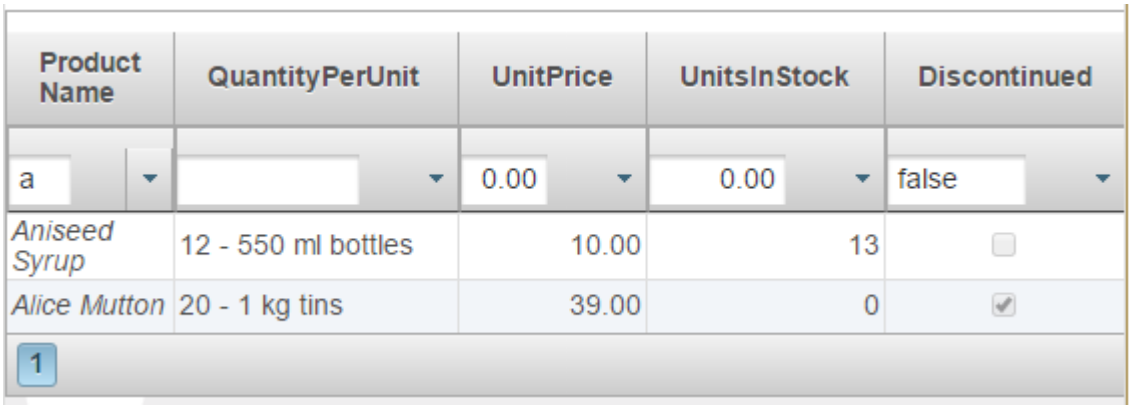


The following code illustrates filtering for **C1TemplateField** by setting **FilterOperator** and **FilterValue** properties.

Source View

```
<Columns>
        <cc1:C1TemplateField DataField="ProductName" HeaderText="Product Name"
FilterOperator="BeginsWith" FilterValue="a">
            <ItemTemplate>
                    <asp:Label runat="server" Font-Italic="true" ID="lbl1" Text='<%#
Bind("ProductName") %>'> </asp:Label>
            </ItemTemplate>
        </cc1:C1TemplateField>
</Columns>
```

The output for the above code is as shown:



Paging

The C1GridView control provides built-in data paging functionality without requiring any coding. By default the grid is displayed in one continuous window. If you choose, you can display the grid on multipage "pages". If paging is set, at run time users can navigate each page of the grid through buttons or links at the top or bottom of the grid. For an example of adding paging, see the Creating a Pageable Grid topic.

To be able to page data at run time, the AllowPaging property must be set to **True** (by default this is **False**). When PagerSettings.Visible is **True** paging navigation appears by default at the bottom of the grid in the grid's footer. You can change the position of the paging controls, if you choose, by setting the Position property. By default, 10 items appear on each page of the grid. You can change this number, by setting the PageSize property. For example, in the image below the PageSize has been set to **5**.

By default the paging controls appear as numeric links:

You can change the appearance of the paging controls by setting the Mode property. You can set the Mode property to the following options:

Mode	Description
NextPrevious	A set of pagination controls consisting of **Previous** and **Next** buttons.
Numeric(default)	A set of pagination controls consisting of numbered link buttons to access pages directly.
NextPreviousFirstLast	A set of pagination controls consisting of **Previous**, **Next**, **First**, and **Last** buttons.
NumericFirstLast	A set of pagination controls consisting of numbered and **First** and **Last** link buttons.

To customize the pager controls, you can set the PagerSettings. The paging properties are easily accessed at design time on the **Paging** tab of the **Property builder** (the **C1GridView Properties** dialog box).

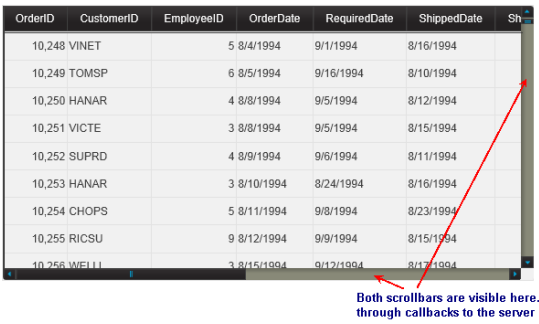
You can use AJAX to update the grid when the user pages the grid at run time by setting the Action property. For more information, see Paging the Grid.

Virtual Scrolling

Virtual scrolling can be used to speed up the rendering of a grid when a massive amount of data is displayed. It allows the grid to retrieve data through callbacks to the server, making the data load faster while keeping the scrolling smooth.

The following example shows how to implement virtual scrolling for rows and columns by setting the ScrollingSettings, VirtualizationSettings and CallbackSettings properties. This markup can be added to the *.aspx page's body content.

```
<asp:Content ID="BodyContent" runat="server" ContentPlaceHolderID="MainContent">
        <asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
   </p>

<%--Bind the grid to a datasource--%>
  <asp:SqlDataSource ID="SqlDataSource1" runat="server" ConnectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=|DataDirectory|\C1NWind.mdb;Persist Security Info=True"
        ProviderName="System.Data.OleDb" SelectCommand="SELECT * FROM ORDERS">
    </asp:SqlDataSource>

 <%--Set C1GridView's ScrollingSettings.VirtualizationSettings.Mode and CallbackSettings.Action--%>


<cc1:C1GridView ID="C1GridView1" runat="server" DataSourceID="SqlDataSource1" Height="400px">
        <CallbackSettings Action="Scrolling" />
                <ScrollingSettings Mode="Both">
                        <VirtualizationSettings Mode="Both" />
                </ScrollingSettings>
</cc1:C1GridView>
```



To implement virtual scrolling for rows and columns in GridView, follow these steps:

1. Bind the grid to a data source.
2. Set the ScrollingSettings Mode to *Both* to enable the virtual scrolling feature.
3. Set the **VirtualizationSettings** Mode value to *Rows*, *Columns*, or *Both* to allow scrolling for rows and columns accordingly.
4. Set the Action property to **Scrolling**. This tells the grid to use callbacks to the server during scrolling. There are two modes of virtual scrolling determined by the CallbackSettings property. The first one is static, where all data is available to the client, but the rows are rendered when scrolled. The second mode is dynamic and set by the **Scrolling** value. New portions of data are requested from the server when they need to be rendered.

Hierarchical Data Binding

C1GridView allows you to bind data in a hierarchical grid of summary rows and detail rows using your customized query. Hierarchical relationships are defined manually between multiple tables in C1DetailGridView. It enables you to display the hierarchical data with multilevel, multiple layouts and provide features such as sorting, filtering, grouping and editing.

The image below shows the hierarchical grid with one of its rows expanded.

Important Properties

- **DataSourceID**: Each grid must have its own data source to represent the data hierarchy in GridView. The data source for each detail grid view is set using the **DataSourceID** property.
- **DataKeyNames**: The DataKeyNames property is used to identify a primary key for the C1GridView control. This property takes the name of fields from the connected database, as an array of strings. The primary key columns for each table in the datasource are added to the **DataKeyNames** property of the respective parent table or child table.

Source View

```
<cc1:C1GridView runat="server" DataSourceID="SqlDataSource1"
DataKeyNames="CustomerID, CompanyName">
```

- **MasterDetailRelation:** The MasterDetailRelation property is used to specify how data is linked between the detail or child table and the parent table. It allows you to link two different tables from the data source by

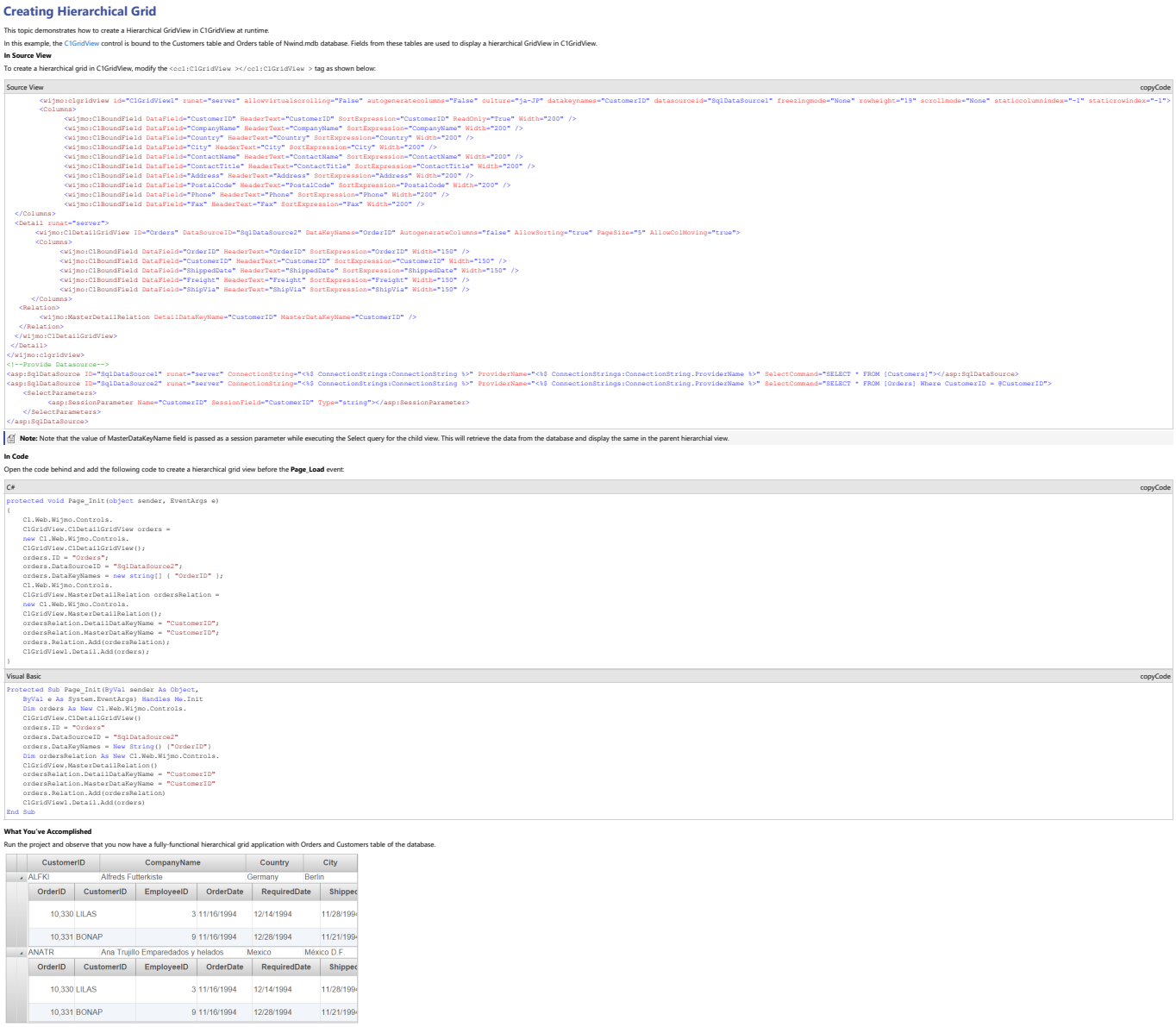
specifying MasterDataKeyName and DetailDataKeyName property.

- **MasterDataKeyName :** The MasterDataKeyName property is used to set the relation between the parent and the child table. This property should be set to match the field specified in DataKeyNames for the parent table.
- **DetailDataKeyName :** The DetailDataKeyName property is also used to set the relation between the child table and the parent table. It retrieves the data from the child table for the detail section. The field set as DetailDataKeyName should be the foreign key of the child table in the corresponding relation and it should also be a primary key in child table so that it matches with the field set as the MasterDataKeyName.

Source View

```
<Relation>
<cc1:MasterDetailRelation MasterDataKeyName="CustomerID"
DetailDataKeyName="CustomerID" />
</Relation>
```

- **PageSize**: The PageSize property can be set to define the number of rows to be displayed in the parent or child grid.



Creating Hierarchical Grid

This topic demonstrates how to create a Hierarchical GridView in C1GridView at runtime.

In this example, the C1GridView control is bound to the Customers table and Orders table of Nwind.mdb database. Fields from these tables are used to display a hierarchical GridView in C1GridView.

In Source View

To create a hierarchical grid in C1GridView, modify the <cc1:C1GridView ></cc1:C1GridView > tag as shown below:

Source View copyCode

```
        <wijmo:c1gridview id="C1GridView1" runat="server" allowvirtualscrolling="False" autogeneratecolumns="False" culture="ja-JP" datakeynames="CustomerID" datasourceid="SqlDataSource1" freezingmode="None" rowheight="19" scrollmode="None" staticcolumnindex="-1" staticrowindex="-1">
        <Columns>
                <wijmo:C1BoundField DataField="CustomerID" HeaderText="CustomerID" SortExpression="CustomerID" ReadOnly="True" Width="200" />
                <wijmo:C1BoundField DataField="CompanyName" HeaderText="CompanyName" SortExpression="CompanyName" Width="200" />
                <wijmo:C1BoundField DataField="Country" HeaderText="Country" SortExpression="Country" Width="200" />
                <wijmo:C1BoundField DataField="City" HeaderText="City" SortExpression="City" Width="200" />
                <wijmo:C1BoundField DataField="ContactName" HeaderText="ContactName" SortExpression="ContactName" Width="200" />
                <wijmo:C1BoundField DataField="ContactTitle" HeaderText="ContactTitle" SortExpression="ContactTitle" Width="200" />
                <wijmo:C1BoundField DataField="Address" HeaderText="Address" SortExpression="Address" Width="200" />
                <wijmo:C1BoundField DataField="PostalCode" HeaderText="PostalCode" SortExpression="PostalCode" Width="200" />
                <wijmo:C1BoundField DataField="Phone" HeaderText="Phone" SortExpression="Phone" Width="200" />
                <wijmo:C1BoundField DataField="Fax" HeaderText="Fax" SortExpression="Fax" Width="200" />
  </Columns>
  <Detail runat="server">
      <wijmo:C1DetailGridView ID="Orders" DataSourceID="SqlDataSource2" DataKeyNames="OrderID" AutogenerateColumns="false" AllowSorting="true" PageSize="5" AllowColMoving="true">
      <Columns>
            <wijmo:C1BoundField DataField="OrderID" HeaderText="OrderID" SortExpression="OrderID" Width="150" />
            <wijmo:C1BoundField DataField="CustomerID" HeaderText="CustomerID" SortExpression="CustomerID" Width="150" />
            <wijmo:C1BoundField DataField="ShippedDate" HeaderText="ShippedDate" SortExpression="ShippedDate" Width="150" />
            <wijmo:C1BoundField DataField="Freight" HeaderText="Freight" SortExpression="Freight" Width="150" />
            <wijmo:C1BoundField DataField="ShipVia" HeaderText="ShipVia" SortExpression="ShipVia" Width="150" />
      </Columns>
    <Relation>
        <wijmo:MasterDetailRelation DetailDataKeyName="CustomerID" MasterDataKeyName="CustomerID" />
    </Relation>
  </wijmo:C1DetailGridView>
 </Detail>
</wijmo:c1gridview>
<!--Provide Datasource-->
<asp:SqlDataSource ID="SqlDataSource1" runat="server" ConnectionString="<%$ ConnectionStrings:ConnectionString %>" ProviderName="<%$ ConnectionStrings:ConnectionString.ProviderName %>" SelectCommand="SELECT * FROM [Customers]"></asp:SqlDataSource>
<asp:SqlDataSource ID="SqlDataSource2" runat="server" ConnectionString="<%$ ConnectionStrings:ConnectionString %>" ProviderName="<%$ ConnectionStrings:ConnectionString.ProviderName %>" SelectCommand="SELECT * FROM [Orders] Where CustomerID = @CustomerID">
    <SelectParameters>
        <asp:SessionParameter Name="CustomerID" SessionField="CustomerID" Type="string"></asp:SessionParameter>
    </SelectParameters>
</asp:SqlDataSource>
```

Note: Note that the value of MasterDataKeyName field is passed as a session parameter while executing the Select query for the child view. This will retrieve the data from the database and display the same in the parent hierarchial view.

In Code

Open the code behind and add the following code to create a hierarchical grid view before the **Page_Load** event:

C# copyCode

```
protected void Page_Init(object sender, EventArgs e)
{
    C1.Web.Wijmo.Controls.
    C1GridView.C1DetailGridView orders =
    new C1.Web.Wijmo.Controls.
    C1GridView.C1DetailGridView();
    orders.ID = "Orders";
    orders.DataSourceID = "SqlDataSource2";
    orders.DataKeyNames = new string[] { "OrderID" };
    C1.Web.Wijmo.Controls.
    C1GridView.MasterDetailRelation ordersRelation =
    new C1.Web.Wijmo.Controls.
    C1GridView.MasterDetailRelation();
    ordersRelation.DetailDataKeyName = "CustomerID";
    ordersRelation.MasterDataKeyName = "CustomerID";
    orders.Relation.Add(ordersRelation);
    C1GridView1.Detail.Add(orders);
}
```

Visual Basic copyCode

```
Protected Sub Page_Init(ByVal sender As Object,
    ByVal e As System.EventArgs) Handles Me.Init
    Dim orders As New C1.Web.Wijmo.Controls.
    C1GridView.C1DetailGridView()
    orders.ID = "Orders"
    orders.DataSourceID = "SqlDataSource2"
    orders.DataKeyNames = New String() {"OrderID"}
    Dim ordersRelation As New C1.Web.Wijmo.Controls.
    C1GridView.MasterDetailRelation()
    ordersRelation.DetailDataKeyName = "CustomerID"
    ordersRelation.MasterDataKeyName = "CustomerID"
    orders.Relation.Add(ordersRelation)
    C1GridView1.Detail.Add(orders)
End Sub
```

What You've Accomplished

Run the project and observe that you now have a fully-functional hierarchical grid application with Orders and Customers table of the database.

	CustomerID	CompanyName	Country	City
▲	ALFKI	Alfreds Futterkiste	Germany	Berlin

OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	Shippe
10,330	LILAS	3	11/16/1994	12/14/1994	11/28/199
10,331	BONAP	9	11/16/1994	12/28/1994	11/21/199

	CustomerID	CompanyName	Country	City
▲	ANATR	Ana Trujillo Emparedados y helados	Mexico	México D.F.

OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	Shippe
10,330	LILAS	3	11/16/1994	12/14/1994	11/28/199
10,331	BONAP	9	11/16/1994	12/28/1994	11/21/199

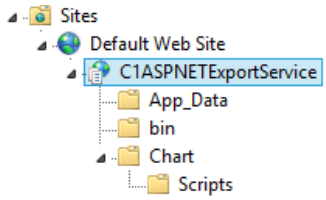
Export Service

Use the C1 ASP.NET Export Service, to export GridView to Excel, CSV and PDF, without creating a complex export application. This service resides on the application server.

The advantages of using C1 ASP.NET Export Service are:

- Export your grid, while preserving its formatting.
- Change exported file settings as per the requirement.

Export Service is a web-application that is deployed on Internet Information Services (IIS). Run the **C1ASPNETExportService** installer, placed in C:\Program Files\ComponentOne\ASP.NET Web Forms Edition folder. It installs the following files in IIS.



You can also locate these files at the following location:

C:\ProgramData\ComponentOne\C1ASPNET\C1APNETExportService

System Requirements

Following are the system requirements of the service host:

- Microsoft Windows 7 or above.
- IIS 7.0 or above with ASP.NET 4.0 or above (.NET framework 4.0).

> You may update .Net framework 4.0 in **Microsoft Windows 7** or **Microsoft Windows Server 2008 R2**. See http://support.microsoft.com/kb/2468871, for more information.

GridView Export Setting

Following are the settings, to download the grid in various formats:

Export to Excel

- **FileFormat**: GridView can be exported to Excel sheets in xls or xlsx format.
- **Author**: Specifies the name of the person or organization responsible for writing the data.
- **Auto Row Height**: Selects whether the rows height will be adjusted based on its content or not.
- **Server URL**: Sets the URL of the server, from where the grid is to be exported. Enter ServerURL**/exportapi/grid**.
- **File Name**: Sets the name to be used for the exported file.

Export to CSV:

- **Server URL**: Sets the URL or the server, from where the grid is to be exported. Enter ServerURL**/exportapi/grid**.
- **File Name**: Sets the name to be used for the exported file.

Export to PDF

- **Repeat Grid Header**: Repeats the grid header.
- **Auto Fit Width**: Enables auto-fit.
- **Landscape**: Enables landscape mode.
- **Margin Settings**:
  - Top: Sets the top margin size in points.
  - Bottom: Sets the bottom margin size in points.
  - Right: Sets the right margin size in points.
  - Left: Sets the left margin size in points.
  - Paper Kind: Sets the paper kind. eg: Custom, Letter, LetterSmall, etc
- **Page Size**:
  - Width: Sets the page width in points
  - Height: Sets the page height in points.
- **File Content**:
  - Image Quality: Sets the image quality to Low, Medium or High
  - Compression: Sets the compression level to Default, None, Best Speed or Best Compression.
  - Font Type: Sets the Font type to True Type or Embedded.
- **Document Info**:
  - Author: Sets the name of the person or organization that created the document.
  - Creator: Sets the name of the application that created the original document.
  - Subject: Sets the subject of the document.
  - Title: Sets document title in the title bar.
  - Producer: Sets the name of the application that created the PDF document.
  - Keywords: Sets the keywords associated with the PDF document that can be used to locate the document.

- **Document Security**:
  - Encryption Type: Sets the Encryption Type to NotPermit, Standard40, Standard128 or Aes128.
  - Owner Password: Sets the password required to edit permissions for the document.
  - User Password: Sets the password required to open the document.
  - Allow Copy Content: Enables or disables copy content.
  - Allow Edit Annotations: Enables or disables users from editing annotations.
  - Allow Edit Content: Enables or disables users from editing content in the document.
  - Allow Print: Enables or disables printing for the document.
- **Configuration Setting**:
  - Server URL: Sets the server URL. Enter ServerURL**/exportapi/grid**.
  - File Name: Sets the file name to be used for the exported PDF.

Usage

The `exportGrid` method will be called to export the GridView content to Excel, CSVS or PDF. These steps assume that you have added a button to the form, on whose click event you would call the export function. Add the following code within the `<head></head>`tags, to export the GridView to Excel.

```
            <script src="http://code.jquery.com/jquery-1.9.1.min.js" type="text/javascript"></script>
 <asp:PlaceHolder runat="server">
            <script type="text/javascript">

                $(function () {
                    $("#Button1").click(exportExcel);
                });

                // Export function
                function exportExcel() {
                    var fileName = "ExportedGrid";
                    var type = "Xls";
                    var excelSetting = {
                        showGridLines: true,
                        autoRowHeight: true,
                        author: "ComponentOne"
                    };
                    var url = "http://demos.componentone.com/ASPNET/ExportService" + "/exportapi/grid";
                    $("#<%=C1GridView1.ClientID%>").c1gridview("exportGrid", fileName, type, excelSetting, url);
    }
            </script>
    </asp:PlaceHolder>
```

Add the following code within the `<head></head>` tags, to export the GridView to a CSV file:

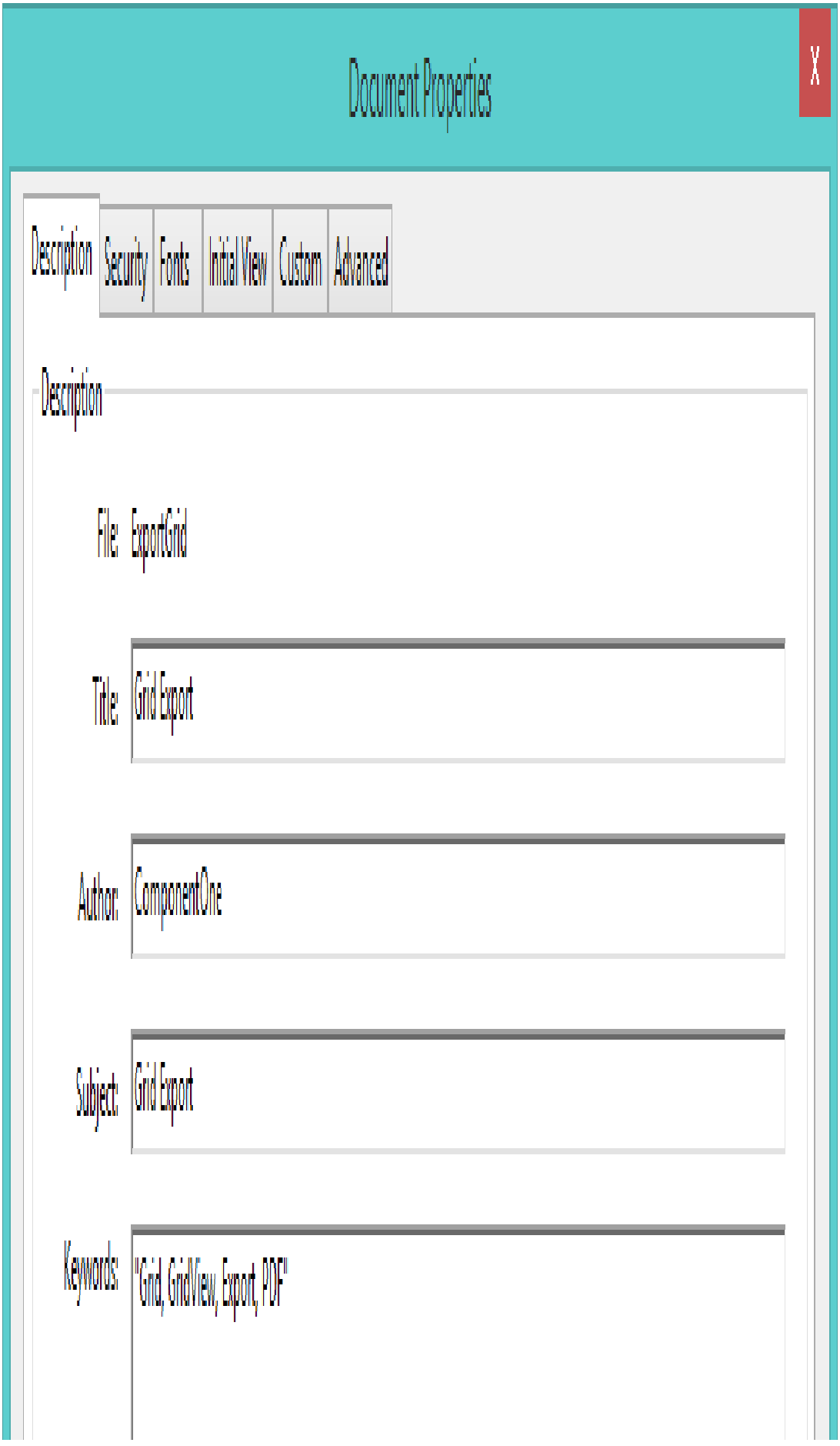
```
        <script src="http://code.jquery.com/jquery-1.9.1.min.js" type="text/javascript"></script>
 <asp:PlaceHolder ID="PlaceHolder1" runat="server">
            <script type="text/javascript">

                $(function () {
                    $("#Button1").click(exportCsv);
                });

                // Export function
                function exportCsv() {
                    var fileName = "ExportedGrid";
                    var url = "http://demos.componentone.com/ASPNET/ExportService" + "/exportapi/grid";
                    $("#<%=C1GridView1.ClientID%>").c1gridview("exportGrid", fileName, "csv", url);
    }
            </script>
        </asp:PlaceHolder>
```

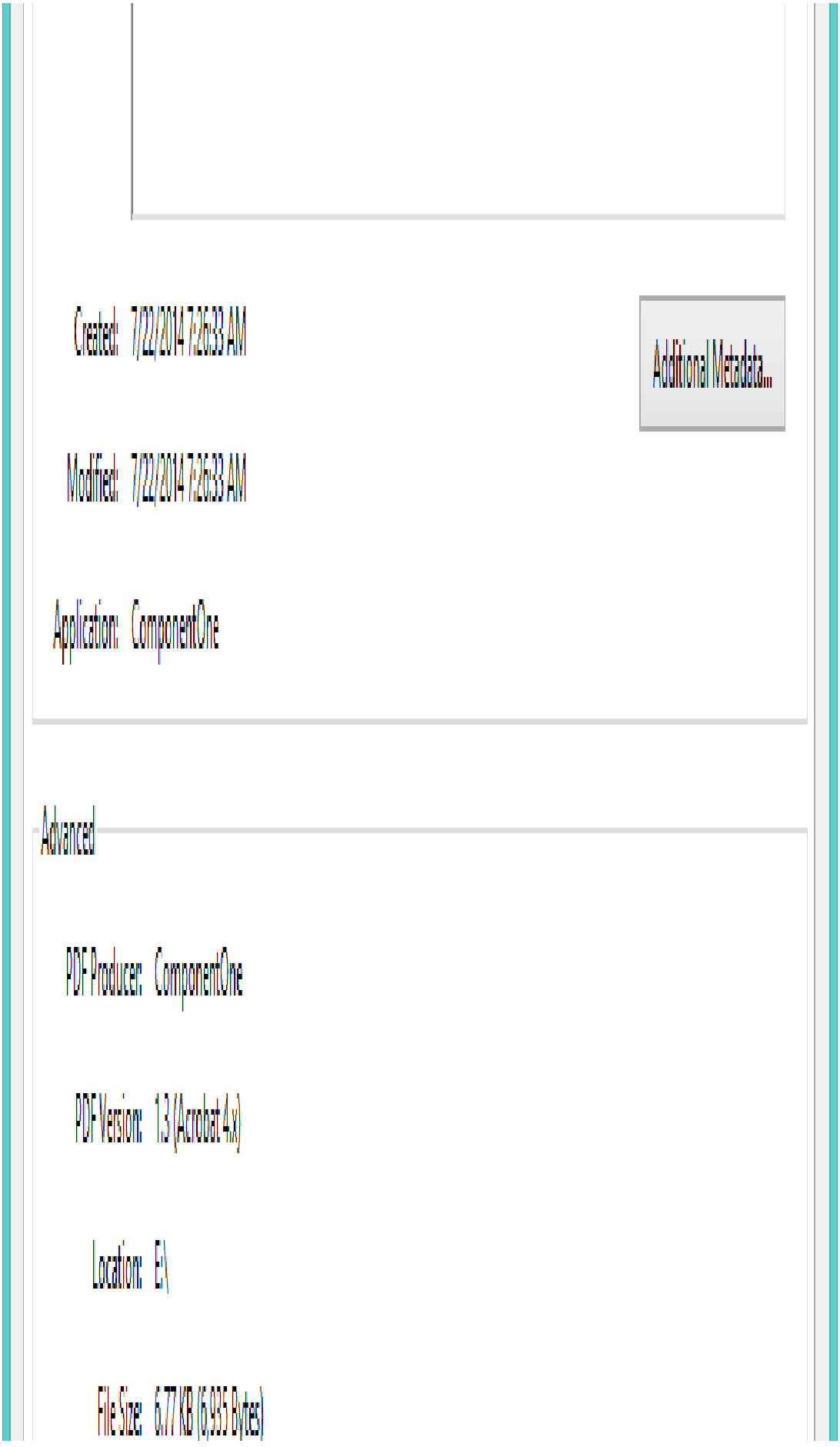
Add the following code within the `<head></head>` tags, to export the GridView to a PDF:

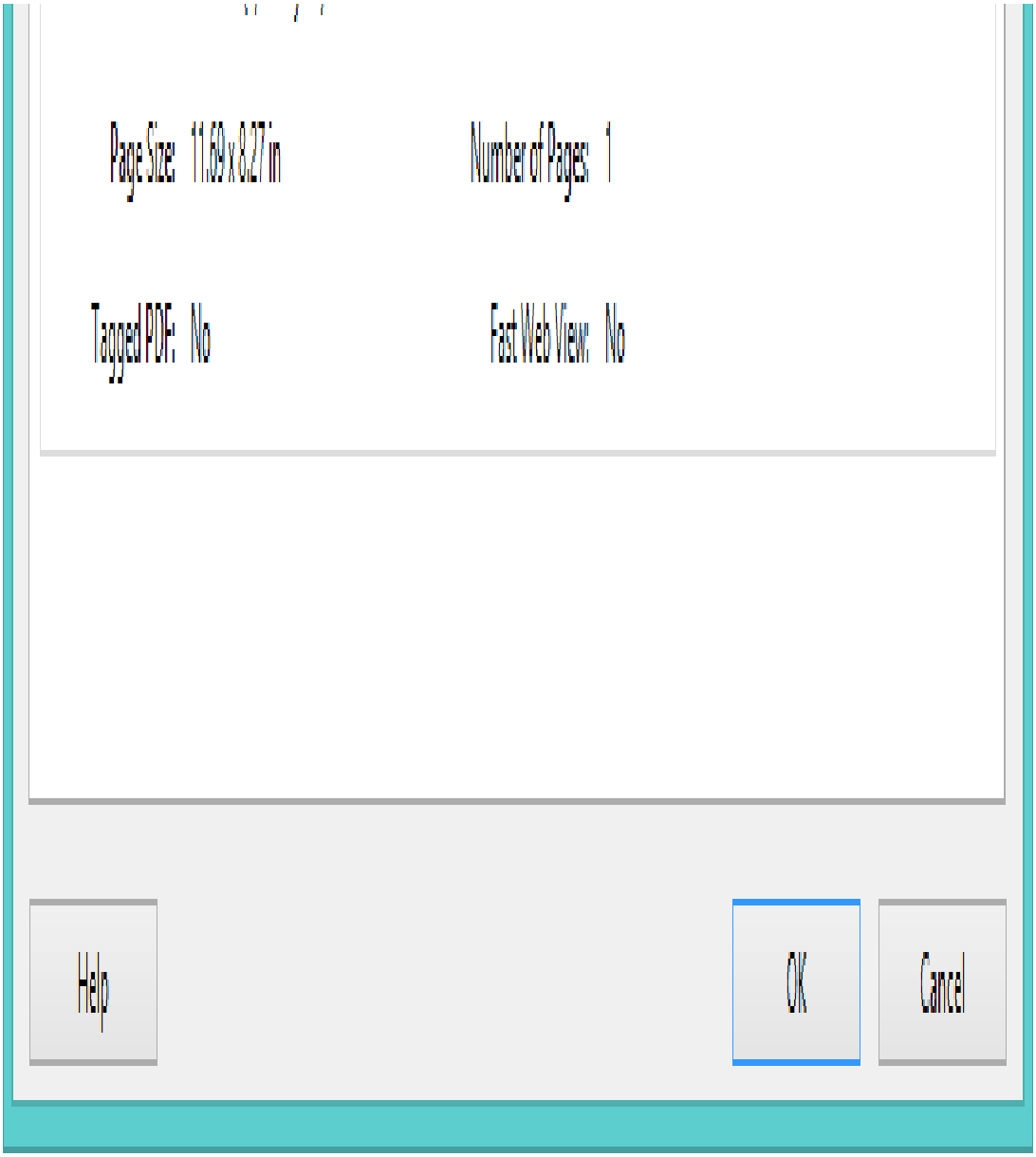
```
<script src="http://code.jquery.com/jquery-1.9.1.min.js" type="text/javascript"></script>
    <asp:PlaceHolder ID="PlaceHolder1" runat="server">
            <script type="text/javascript">

                $(function () {
                    $("#Button1").click(exportPdf);

                });

                // pdf settings
                function getPdfSetting() {
                        return {
                            repeatHeader: true,
                            landscape: true,
                            autoFitWidth: true,
                            pageSize: {
                                width: 300,
                                height: 400
                            },
```

```
                        paperKind: 'A4',
                        margins: {
                            top: 50,
                            right: 50,
                            bottom: 50,
                            left: 50
                        },
                        imageQuality:'Low',
                        compression: 'BestCompression',
                        fontType: 'TrueType',
                        author: 'ComponentOne',
                        creator: 'ComponentOne',
                        subject: 'Grid Export',
                        title: 'Grid Export',
                        producer: 'ComponentOne',
                        keywords: 'Grid, GridView, Export, PDF',
                        encryption: 'NotPermit',
                        ownerPassword: '0000',
                        userPassword: '00000',
                        allowCopyContent: true,
                        allowEditAnnotations: true,
                        allowEditContent: true,
                        allowPrint: true,
                    }
                }

                // Export function
                    function exportPdf() {
                    var fileName = 'ExportGrid';
                    var pdfSetting = getPdfSetting();
                    var url = "http://demos.componentone.com/ASPNET/ExportService" + "/exportapi/grid";
                    $("#<%=C1GridView1.ClientID%>").c1gridview("exportGrid", fileName, "pdf", pdfSetting, url);
                    }

            </script>
</asp:PlaceHolder>
```

The following image displays the properties of the PDF generated:







In case the file is not downloaded on Internet Explorer, turn off Internet Explorer protected mode to export file or run Internet Explorer as administrator. To turn off the protected mode:

- Open Internet Explorer **Settings** and select **Internet Options**.
- In the **Security** tab select **Internet** and uncheck "**Enable Protected Mode**".

Using AJAX

Asynchronous JavaScript and XML (AJAX) provides a very effective way of communicating with data which resides on a server. This means that Web pages (ASP.NET Web applications) can execute server-side logic and update various page elements without reloading the entire page. This is a highly efficient way to present data via Web applications. It saves page reloading time and provides a more streamlined user experience.

AJAX allows **GridView for ASP.NET Web Forms** to load data without having to do a postback to the server, minimizing load time and greatly improving the end-user experience. Using C1GridView's **Callback** feature, the grid calls back to the server to retrieve only the information that is requested, unlike with a server postback, where the whole page must be reloaded to update the grid. End-users can quickly modify a grid without that annoying flicker of the screen during load time.

You can use the **Callback** feature for editing the grid, grouping and outlook grouping, paging, row selection, sorting,

and scrolling the grid by simply setting **C1GridView**'s Action property to one of the following options:

- **None**: All actions are performed via postback.
- **ColMove**: Column moving is performed via callbacks.
- **Editing**: Editing is performed via callbacks.
- **Paging**: Paging is performed via callbacks.
- **Selection**: Row selection is performed via callbacks.
- **Sorting**: Sorting is performed via callbacks.
- **Filtering**: Filtering is performed via callbacks.
- **All**: All actions on the grid are performed via callbacks.

For more information on using **GridView for ASP.NET Web Forms** and AJAX, see the Updating the Grid with AJAX topic.

Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio .NET, know what a DataSet is, and know how to use bound controls in general. By following the steps outlined in the help, you will be able to create projects demonstrating a variety of C1GridView features and get a good sense of what C1GridView can do.

The help uses the standard Northwind Access database, Nwind.mdb, for bound data, which is installed by default in the **ComponentOne Samples\Common** folder in your **Documents** folder. The database is referred to by filename instead of the full pathname for the sake of brevity.

> **Note:** Depending on where you store the projects and database files, you may need to change the location of the Nwind.mdb reference in the projects.

Each task-based help topic also assumes that you have created a new ASP.NET project, have placed a bound C1GridView control on the form, and have used the **Imports** (Visual Basic) or **using** (C#) statement for the **C1.Web.Wijmo.Controls.C1GridView** namespace and relevant System namespaces. For additional information, see Binding the Grid to a Data Source.

Binding the Grid to a Data Source

The following topics demonstrate how to bind C1GridView to a data source control. The steps for binding C1GridView to a data source are slightly different in a Web site project from a Web application project.

Binding the Grid in a Web Site Project

To bind C1GridView to a data source control in a Web Site project, complete the following steps:

1. In the Solution Explorer, right click the project name and choose **Add | New Folder**. Name the new folder "App_Data".
2. In the Solution Explorer, right-click the **App_Data** folder, and then click **Add Existing Item**.
3. Locate the Nwind.mdb file, installed by default in the samples folder.
4. In the **Add Existing item** dialog box, click the Nwind.mdb file, and then click **Add**.
5. Go back to the Default.aspx page.
6. Select the C1GridView control, and click the Smart Tag to open the **C1GridView Tasks** menu.
7. Click the **Choose Data Source** drop-down arrow and select **<New data source>**.
8. In the Data Source Configuration Wizard, select **Access Database** and click **OK**.
9. Click **Browse** and select **App_Data** under **Project folders** in the **Select Microsoft Access Database** dialog box.
10. Choose Nwind.mdb in the **Contents of folder** pane on the right-hand side and click **OK**.
11. Click **Next**. The **Configure the Select Statement** window appears.
    a. Confirm that the **Specify columns from a table or view** option is selected.
    b. In the **Name** drop-down list, select **Products**.
    c. In the **Columns** box, select **ProductName**, **QuantityPerUnit**, and **UnitsInStock** or desired other check boxes.
12. Click **Next**. The **Test Query** page appears. Optionally, click **Test Query** to test your query.
13. Click **Finish** to close the wizard and add the data source.

Binding the Grid in a Web Application Project

To bind C1GridView to a data source control in a Web Application project, complete the following steps:

1. In the project, select **Project | Add Existing Item**.
2. Locate and select the **Nwind.mdb** file, installed by default in the samples folder, and click **Add**. The **Data Source Configuration Wizard** appears.
3. Check the **Tables** check box and click **Finish**.
4. Right-click the C1GridView control and then click **Show Smart Tag**.
5. On the **C1GridView Tasks** menu, click the **Choose Data Source** drop-down arrow and select **<New data source...>.** The **Data Source Configuration Wizard** appears.
6. Select **Access Database** and click **OK**.
7. Click the **Browse** button and select the **Nwind.mdb** that appears under **Contents of folder**.
8. Click **OK** and then **Next**. The **Configure the Select Statement** window appears.
    a. Make sure the **Specify columns from a table or view** option is selected.
    b. In the **Name** drop-down list, select **Products**.
    c. In the **Columns** box, select **ProductID**, **ProductName**, **QuantityPerUnit**, and **UnitsInStock** or other desired check boxes.
9. Click **Next**. The **Test Query** page appears. Optionally, click **Test Query** to test your query.
10. Click **Finish** to close the wizard and add the data source.

Automatically Updating the Data Source

This topic illustrates how to create an editable grid using templates and a DataSource control. Complete the following steps:

1. Create a grid and bind it to a **DataSource** control; see Binding the Grid to a Data Source topic for details. Use the **Suppliers** table in the Northwind database and retrieve the *SupplierID*, *CompanyName*, *ContactName*, and *Address* fields.
2. Configure UpdateQuery:
    a. Right-click the **AccessDataSource1** component and then click **Properties**. In the **Properties** tab click the **ellipsis** button next to **UpdateQuery**. The **Command and Parameter Editor** dialog box appears.
    b. Click **Query Builder**, select the **Suppliers** table and click **Add**.
    c. Insert the following text in UPDATE command and click **OK** to close the Query Builder:
    `UPDATE Suppliers SET CompanyName = ?, ContactName = ?, Address = ? WHERE (SupplierID = ?)`
    d. Click the **Add Parameter** button and rename the parameter "SupplierID".
    e. Click **Show advanced properties** and set the **Type** property to **Int32**.
    f. In the same way, add *CompanyName*, *ContactName*, *Address* parameters, but set their **Types** to *String*.
    g. Click **OK** to close the **Command and Parameter Editor** dialog box.
3. Right-click the C1GridView control and then click **Show Smart Tag**.
4. On the **C1GridView Tasks** menu, in the **Choose Data Source** box, click **AccessDataSource1**, if necessary.
5. Right-click the **C1GridView** control and click **Properties**. In the Properties window set the DataKeyNames value to *SupplierID*.
6. In the Properties window, set the AllowClientEditing property to **True**.

Runtime Databinding of Template Field

This topic demonstrates how to bind template fields to a C1GridView at runtime.

In this example, the C1GridView control is bound to the Customers table of C1Nwind.mdb database, and the field Country is attached to a C1TemplateField containing a C1ComboBox.

In the Designer

Complete the following steps:

1. Right-click the C1GridView control and select Show Smart Tag from the context menu to bind it to the *C1Nwind.mdb* database, which is located by default in the samples directory. For detailed steps go to Step 1 of 3: Binding C1GridView to a DataSource.
2. Add a template field to the Country field column and add C1ComboBox in the ItemTemplate section in the editor document. For detailed steps go to Adding Controls to a Column.

In Source View

To add a Template column in C1GridView, modify the `<cc1:C1GridView ></cc1:C1GridView >` tag as shown

below:

```
<cc1:C1GridView ID="C1GridView1" runat="server" AutogenerateColumns="False"
DataKeyNames="CustomerID" DataSourceID="SqlDataSource1">
            <Columns>
                <cc1:C1BoundField DataField="CustomerID" HeaderText="CustomerID"
ReadOnly="True" SortExpression="CustomerID">
                </cc1:C1BoundField>
                <cc1:C1BoundField DataField="CustomerName" HeaderText="CustomerName"
SortExpression="CustomerName">
                </cc1:C1BoundField>
                  <cc1:C1TemplateField HeaderText="Country">
                <ItemTemplate>
                    <span>
                        <%# Eval("Country") %></span>
                </ItemTemplate>
                <EditItemTemplate>
                   <cc1:C1ComboBox ID="dlCountry" runat="server">
                   </cc1:C1ComboBox>
                 </EditItemTemplate>
            </cc1:C1TemplateField>
            <cc1:C1CommandField ShowEditButton="True">
            </cc1:C1CommandField>
        </Columns>

        </cc1:C1GridView>
```

In Code

1. To bind the C1ComboBox in each row, add the following code to handle the RowDataBound event :

   To write the code in Visual Basic

   Visual Basic

```
 Protected Sub C1GridView1_RowDataBound(sender As Object, e As
C1.Web.Wijmo.Controls.C1GridView.C1GridViewRowEventArgs) Handles C1GridView1.RowDataBound
      If (e.Row.RowState And C1GridViewRowState.Edit) > 0 Then
         ' Retrieve the underlying data item. In this example
         ' the underlying data item is a DataRowView object.
         Dim rowView As DataRowView = DirectCast(e.Row.DataItem, DataRowView)
         ' Retrieve the country value for the current row.
         Dim country As [String] = rowView("Country").ToString()
         ' Retrieve the DropDownList control from the current row and DataBind it.
         Dim dlCountry As C1ComboBox = DirectCast(e.Row.FindControl("dlCountry"), C1ComboBox)
         dlCountry.DataSource = GetData()
         dlCountry.DataTextField = "Country"
         dlCountry.DataValueField = "Country"
         dlCountry.DataBind()
         'Retrieve item from dropdown and set it as selected.
         For Each item As C1ComboBoxItem In dlCountry.Items
            If item.Text = country Then
               dlCountry.SelectedValue = item.Text
            End If
         Next
      End If
   End Sub
```

```
    Protected Function GetData() As DataTable
        Dim con As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0; Data Source=" +
Server.MapPath("~/App_Data/C1NWind1.mdb"))
        Dim cmd As New OleDbCommand("Select Country from Countries", con)
        Dim da As New OleDbDataAdapter(cmd)
        Dim dt As New DataTable()
        da.Fill(dt)
        Return dt
    End Function
```

To write the code in C#

C#

```
protected void C1GridView1_RowDataBound(object sender,
C1.Web.Wijmo.Controls.C1GridView.C1GridViewRowEventArgs e)
    {
        if ((e.Row.RowState & C1GridViewRowState.Edit)>0) {
            // Retrieve the underlying data item. In this example
            // the underlying data item is a DataRowView object.
            DataRowView rowView = (DataRowView)e.Row.DataItem;
            // Retrieve the country value for the current row.
            String country = rowView["Country"].ToString();
            // Retrieve the DropDownList control from the current row and DataBind it.

            C1ComboBox dlCountry = (C1ComboBox)e.Row.FindControl("dlCountry");
            dlCountry.DataSource = GetData();
            dlCountry.DataTextField = "Country";
            dlCountry.DataValueField = "Country";
            dlCountry.DataBind();

            //Retrieve item from dropdown and set it as selected.

            foreach (C1ComboBoxItem item in dlCountry.Items) {
                if (item.Text == country) {
                    dlCountry.SelectedValue = item.Text;
                }
            }
        }
    }
    DataTable GetData() {
        OleDbConnection con = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0; Data
Source="+Server.MapPath("~/App_Data/C1NWind1.mdb"));
        OleDbCommand cmd = new OleDbCommand("Select Country from Countries",con);
        OleDbDataAdapter da = new OleDbDataAdapter(cmd);
        DataTable dt = new DataTable();
        da.Fill(dt);
        return dt;
    }
```

2. To provide the parameters, i.e. SelectedValue of C1ComboBox, to the Update command, add the following code to handle the RowUpdating event:

To write the code in Visual Basic

Visual Basic

```
 Protected Sub C1GridView1_RowUpdating(sender As Object, e As
C1.Web.Wijmo.Controls.C1GridView.C1GridViewUpdateEventArgs) Handles C1GridView1.RowUpdating
        ' Retrieve the row being edited.
        Dim row As C1GridViewRow = C1GridView1.Rows(C1GridView1.EditIndex)
        ' Retrieve the DropDownList control from the row.
        Dim combo As C1ComboBox = DirectCast(row.FindControl("dlCountry"), C1ComboBox)
        ' Add the selected value of the DropDownList control to
        ' the NewValues collection. The NewValues collection is
        ' passed to the data source control, which then updates the
        ' data source.
        e.NewValues("Country") = combo.SelectedValue
        SqlDataSource1.UpdateCommand = "Update Customers Set CustomerName=@CustomerName,
Country=@Country where CustomerID=@CustomerID"
        SqlDataSource1.UpdateParameters.Add("CustomerID", e.Keys("CustomerID").ToString())
        SqlDataSource1.UpdateParameters.Add("CustomerName", e.NewValues("CustomerName").ToString())
        SqlDataSource1.UpdateParameters.Add("Country", e.NewValues("Country").ToString())
        SqlDataSource1.Update()
    End Sub
```

To write the code in C#

C#

```
 protected void C1GridView1_RowUpdating(object sender,
C1.Web.Wijmo.Controls.C1GridView.C1GridViewUpdateEventArgs e)
    {
        // Retrieve the row being edited.
        C1GridViewRow row = C1GridView1.Rows[C1GridView1.EditIndex];
        // Retrieve the DropDownList control from the row.

        C1ComboBox combo = (C1ComboBox)row.FindControl("dlCountry");
        // Add the selected value of the DropDownList control to
        // the NewValues collection. The NewValues collection is
        // passed to the data source control, which then updates the
        // data source.

        e.NewValues["Country"] = combo.SelectedValue;
        SqlDataSource1.UpdateCommand = "Update Customers Set CustomerName=@CustomerName,
Country=@Country where CustomerID=@CustomerID";
        SqlDataSource1.UpdateParameters.Add("CustomerID", e.Keys["CustomerID"].ToString());
        SqlDataSource1.UpdateParameters.Add("CustomerName", e.NewValues["CustomerName"].ToString());
        SqlDataSource1.UpdateParameters.Add("Country", e.NewValues["Country"].ToString());
        SqlDataSource1.Update();
    }
```

Runtime Data Operations

The following topics explain how to perform data operations like sorting, filtering, paging and grouping to your grid when data is bounded dynamically.

- **Sorting**

  To implement sorting, you need to handle the **Sorting** and **Sorted** events. Use the following code to rebind the grid in the Sorted event:

  To write the code in Visual Basic

```
Visual Basic
 Protected Sub C1GridView1_Sorting(sender As Object, e As
C1.Web.Wijmo.Controls.C1GridView.C1GridViewSortEventArgs) Handles C1GridView1.Sorting
   End Sub
   'Handles Sorting
   Protected Sub C1GridView1_Sorted(sender As Object, e As EventArgs) Handles C1GridView1.Sorted
      C1GridView1.DataSource = BindGrid()
      C1GridView1.DataBind()
   End Sub
```

  To write the code in C#

```
C#
 protected void C1GridView1_Sorting(object sender,
C1.Web.Wijmo.Controls.C1GridView.C1GridViewSortEventArgs e)
   {
   }
//Handles Sorting
protected void C1GridView1_Sorted(object sender, EventArgs e)
   {
      C1GridView1.DataSource = BindGrid();
      C1GridView1.DataBind();
   }
```

- **Filtering**

  To implement filtering, you need to handle the **Filtering** and **Filtered** events. Use the following code to rebind the grid in the Filtered event:

  To write the code in Visual Basic

```
Visual Basic
Protected Sub C1GridView1_Filtering(sender As Object, e As
C1.Web.Wijmo.Controls.C1GridView.C1GridViewFilterEventArgs)
End Sub
  'Handles Filtering
  Protected Sub C1GridView1_Filtered(sender As Object, e As EventArgs) Handles C1GridView1.Filtered
      C1GridView1.DataSource = BindGrid()
      C1GridView1.DataBind()
End Sub
```

To write the code in C#

```
C#
 protected void C1GridView1_Filtering(object sender,
C1.Web.Wijmo.Controls.C1GridView.C1GridViewFilterEventArgs e)
   {
   }
   //Handles Filtering
   protected void C1GridView1_Filtered(object sender, EventArgs e)
   {
      C1GridView1.DataSource = BindGrid();
      C1GridView1.DataBind();
   }
```

- **Paging**

To implement paging, you need to handle the **Paging** event. Use the following code to rebind the grid in the grid after you set the **NewPageIndex** as the **PageIndex** of C1GridView:

To write the code in Visual Basic

```
Visual Basic
'Handles Paging
Protected Sub C1GridView1_PageIndexChanging(sender As Object, e As
C1.Web.Wijmo.Controls.C1GridView.C1GridViewPageEventArgs) Handles C1GridView1.PageIndexChanging
      C1GridView1.PageIndex = e.NewPageIndex
      C1GridView1.DataSource = BindGrid()
      C1GridView1.DataBind()
   End Sub
```

To write the code in C#

```
C#
//Handles Paging
   protected void C1GridView1_PageIndexChanging(object sender,
C1.Web.Wijmo.Controls.C1GridView.C1GridViewPageEventArgs e)
   {
      C1GridView1.PageIndex = e.NewPageIndex;
      C1GridView1.DataSource = BindGrid();
      C1GridView1.DataBind();
   }
```

- **Grouping**

To implement grouping, you need to handle the **ColumnGrouped** and **ColumnUngrouped** events.

> **Note:** For grouping C1GridView, set AllowColMoving and ShowGroupArea properties to true.

Use the following code to rebind the grid in the ColumnGrouped event while providing HeaderText of the dragged column as the parameter:

To write the code in Visual Basic

```
Visual Basic
'Handles Column UnGrouping
Protected Sub C1GridView1_ColumnUngrouped(sender As Object, e As
C1.Web.Wijmo.Controls.C1GridView.C1GridViewColumnUngroupedEventArgs) Handles
C1GridView1.ColumnUngrouped
    End Sub
'Handles Column Grouping
    Protected Sub C1GridView1_ColumnGrouped(sender As Object, e As
C1.Web.Wijmo.Controls.C1GridView.C1GridViewColumnGroupedEventArgs) Handles
C1GridView1.ColumnGrouped
        C1GridView1.DataSource = BindGrid(e.Drag.HeaderText)
        C1GridView1.DataBind()
    End Sub
```

To write the code in C#

```
C#
//Handles Column UnGrouping
    protected void C1GridView1_ColumnUngrouped(object sender,
C1.Web.Wijmo.Controls.C1GridView.C1GridViewColumnUngroupedEventArgs e)
    {
    }
    //Handles Column Grouping
    protected void C1GridView1_ColumnGrouped(object sender,
C1.Web.Wijmo.Controls.C1GridView.C1GridViewColumnGroupedEventArgs e)
    {
        C1GridView1.DataSource = BindGrid(e.Drag.HeaderText);
        C1GridView1.DataBind();
    }
```

Using the Property Builder

The Property builder allows you to easily customize the appearance and behavior of the C1GridView control. Using the Property builder, you can determine whether to show headers and footers, customize paging and page navigation, and format the appearance of the C1GridView control.

To access the Property builder, select **Property builder** from the **C1GridView Tasks** menu.
The **C1GridView Properties** dialog box appears. For more information about the Property builder and the available options, see the Property Builder topic.

Setting Properties Using the Property Builder

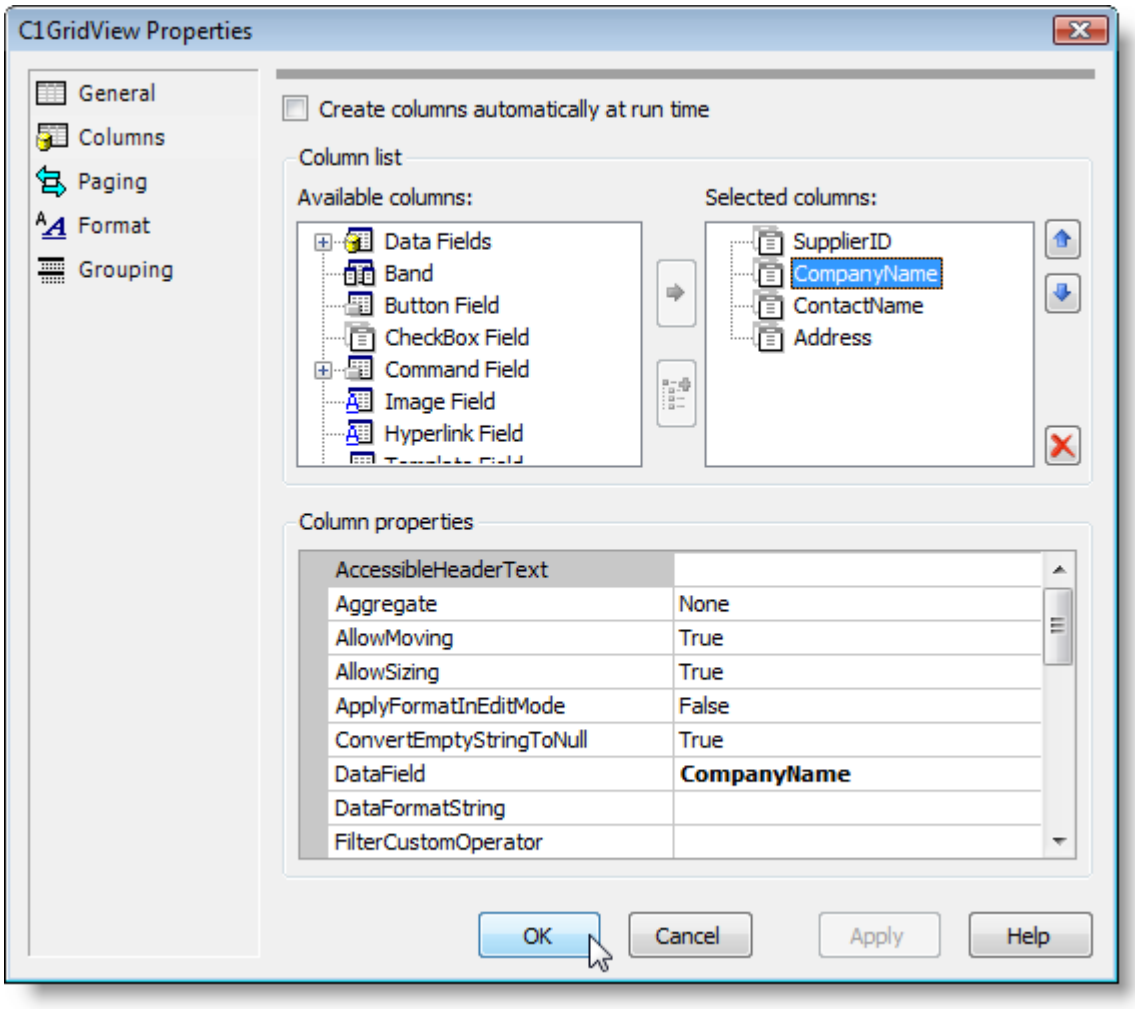
The **C1GridView Properties** dialog box allows you to easily set properties to customize the C1GridView control's appearance and behavior. To set properties using the Property builder, complete the following steps:

1. Right-click **C1GridView** and select **Show Smart Tag** from the context menu. Select **Property builder** from the **C1GridView Tasks** menu.
   The **C1GridView Properties** dialog box appears.
2. From the **C1GridView Properties** dialog box, select one of the following tabs in the left column. For this

example, the **Columns tab** is selected.
The **C1GridView Properties** dialog box includes five tabs as follows:

Tab	Description
General	Displays the data source and allows you to set headers, footers, and sorting properties. See General tab for more information.
Columns	Allows you to specify the types of columns that appear and set the properties for each. See Columns tab for more information.
Paging	You can determine whether paging is used, and you can customize how and where it appears. See Paging tab for more information.
Format	Allows you to set the font, color and alignment of the grid and everything within it, including the headers, footers, the pager, specific items, and columns. See Format tab for more information.
Grouping	You can determine whether a column is grouped on and how the group header and footer rows should be formatted and displayed. See Grouping tab for more information.

3. Choose one of the columns in the list of **Selected columns**.
4. Set the desired properties under **Column properties** and click **Apply**.
5. Once you are finished setting the properties, click **OK**.



Alternatively, C1GridView properties can be set using the Properties window at design time.

1. Select the C1GridView control.
2. Select **Properties Window** from the **View** menu on the Visual Studio toolbar.
3. In the **Properties** window, set any of the desired properties.

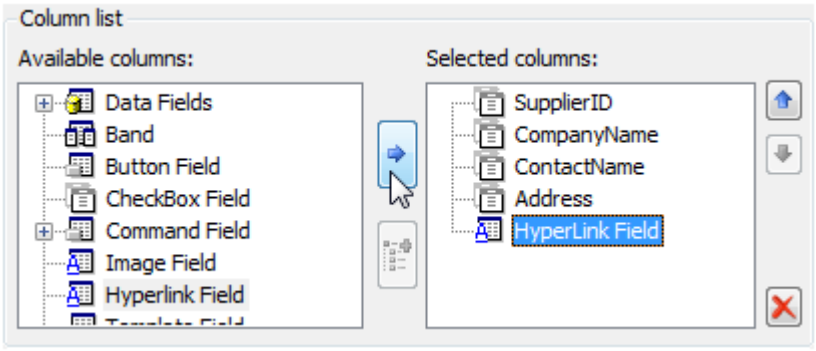
Adding Columns Using the Property Builder

Using the **C1GridView Properties** dialog box, you can also add and remove columns and determine the types of columns to display. Available column types include:

- C1Band: displays a banding column used to create multilevel column headers.
- C1BoundField: displays a column bound to a field in a data source.
- C1ButtonField: displays a command button for each item.
- C1CheckBoxField: displays a Boolean check box for each item.
- C1CommandField: displays a column containing editing commands for each item.
- C1HyperLinkField: displays each item as a hyperlink.
- C1TemplateField: displays each item in the column following a specified template.

To add columns, complete the following steps:

1. Right-click C1GridView control and select **Show Smart Tag** from the context menu. Select **Property builder** from the **C1GridView Tasks** menu. The **C1GridView Properties** dialog box appears.
2. Click the **Columns** tab in the left pane. For more information about the **Columns** tab, see the **Columns** tab topic.
3. Under **Column List**, select the desired column from the list of **Available columns**.
4. Click the arrow button between the column lists to create a column under the list of **Selected columns**. In the image below a **HyperLink** column was added:



Note: When the check box next to **Create columns automatically at run time** is selected (default), bound columns are automatically created. If you deselect the box, you must manually create columns.

Formatting the Grid's Content

The following task-based help topics explain how to format your grid, including customizing the grid's content, column headers, footers, and more.

Creating a Sorted Grid

To configure the grid to automatically sort a column, complete the following steps:

1. Select the C1GridView control and set the C1GridView.AllowSorting property to True.

   To write the code in Visual Basic:

   ```
   Visual Basic
   C1GridView1.AllowSorting = True
   ```

   To write the code in C#:

   ```
   C#
   C1GridView1.AllowSorting = true;
   ```

2. Set the **C1GridView.DataSourceID** property.
3. Run your application and click the HeaderText of the column you want to sort.

Sample Project Available
For the complete sample, see the **Sorting page** located in the ControlExplorer sample. See GridView for ASP.NET Web Forms Samples for more information.

Hiding Specified Columns

You can choose to hide columns in the C1GridView control by setting the Visible property for those columns to **False**. To specify the columns to be displayed in C1GridView, complete the following steps:

1. Right-click **C1GridView** and select **Show Smart Tag** from the context menu.
2. Select Property builder from the **C1GridView Tasks** menu. The **C1GridView Properties** dialog box appears.
3. Click the **Columns** tab in the left pane.
4. Deselect the **Create columns automatically at run time** check box if it is selected.
5. Select a column that you do not want to appear from the list of **Selected columns**.
6. Under **Column** properties, set the Visible property to **False**. Do this for any column in the data source that you do not want to appear in the grid.
   Note that you can also remove columns from being displayed by selecting the column in the *Selected columns* list and clicking the **Delete** button ("X").
7. Click **OK** to save your settings and close the **C1GridView Properties** dialog box.

What You've Accomplished
Run your application and observe that the columns that you changed the visibility for do not appear in the grid at run time.

Changing the Color of Column Headers

You can easily use themes to customize the grid's appearance. For more information about themes and available themes, see the Themes topic. In the following steps, you'll learn to customize the appearance of the Grid control other than using themes.

> **Note:** These steps assume that you have already created an Asp.Net project containing a GridView. For more information, see the GridView for ASP.NET Web Forms Quick Start topic.

Complete the following steps:

1. Click the **Source** button to switch to Source view.
2. Add the following code inside the <head> tag on your page:

```
<style type="text/css">
        .wijgridth {
            background-image: none !important;
            background-color: Yellow !important;
            height: 27px;
            text-align: -moz-center;
            border: solid 1px #4c535c;
            vertical-align: middle;
            border-right: none;
            font-weight: normal;
            color: #000;
            border-top: none;
        }
    </style>
```

Note: ASP.NET controls contain images in its background; therefore before making any changes to the background of a control's element like header, toolbar or cell, you need to remove the image first.

What You've Accomplished

Run your application and observe that the column headers now appear yellow:

Product	Category	Price	Stock	Discontinued
Chai	Beverages	$18.00	39	☐
Chang	Beverages	$19.00	17	☐
Aniseed Syrup	Condiments	$10.00	13	☐
Chef Anton's Cajun Seasoning	Condiments	$22.00	53	☐

Customizing Column Data using ValueLists

This topic demonstrates how to change the display of cell data using the **ValueList** property.
With the **ValueList** property, you can substitute the actual cell data values of columns like those with ID data with understandable display values.

In Code

Add the following code to the **Page_Load** event to prepare a **ValueList** dictionary that contains the replacement text:

To write the code in VB:

Visual Basic

```
' Prepare ValueList dictionary
Dim ht As New Hashtable()
ht.Add("1", "Beverages")
ht.Add("2", "Condiments")
ht.Add("3", "Confections")
ht.Add("4", "Dairy Products")
ht.Add("5", "Grains/Cereals")
```

```
ht.Add("6", "Meat/Poultry")
ht.Add("7", "Produce")
ht.Add("8", "Seafood")
```

To write the code in C#:

C#

```
// Prepare ValueList dictionary
 Hashtable ht = new Hashtable();
 ht.Add("1", "Beverages");
 ht.Add("2", "Condiments");
 ht.Add("3", "Confections");
 ht.Add("4", "Dairy Products");
 ht.Add("5", "Grains/Cereals");
 ht.Add("6", "Meat/Poultry");
 ht.Add("7", "Produce");
 ht.Add("8", "Seafood");
```

Assign the column, **CategoryID** in this example, which contains the items to be replaced by the items in the **ValueList** dictionary:

To write the code in Visual Basic:

Visual Basic

```
' Assign values from the ValueList dictionary to the actual cell data value.
 CType(C1GridView1.Columns.ColumnByName("CategoryID"),
C1.Web.Wijmo.Controls.C1GridView.C1BoundField).ValueList = ht Sub
```

To write the code in C#:

C#

```
// Assign values from the ValueList dictionary to the
actual cell data value.
 ((C1.Web.Wijmo.Controls.C1GridView.C1BoundField)C1GridView1.Columns.ColumnByName("CategoryID")).ValueList
= ht;
```

What You've Accomplished

When your run the project, the items in the *CategoryID* column of the grid are replaced with the items in the **ValueList** dictionary and appear in the *CategoryID* column of the C1GridView.
The initial grid without substituted text appears similar to the following:

Product	Category	Price	Stock	Discontinued
Chai	1	$18.00	39	☐
Chang	1	$19.00	17	☐
Aniseed Syrup	2	$10.00	13	☐
Chef Anton's Cajun Seasoning	2	$22.00	53	☐
Chef Anton's Gumbo Mix	2	$21.35	0	☑

The grid with the substituted text appears similar to the following:

Product	Category	Price	Stock	Discontinued
Chai	Beverages	$18.00	39	☐
Chang	Beverages	$19.00	17	☐
Aniseed Syrup	Condiments	$10.00	13	☐
Chef Anton's Cajun Seasoning	Condiments	$22.00	53	☐
Chef Anton's Gumbo Mix	Condiments	$21.35	0	☑

Setting Alternate Row Styles

This topic demonstrates how to change the color of alternating rows of a C1GridView.

In the Designer

Complete the following steps:

1. Drag and drop a C1GridView control onto your Web Form.
2. Add data to the **C1GridView**. For more information, see Binding the Grid to a Data Source.
3. Click **OK**.

In Source View

To change the color of alternating rows, add the following code between the <head> </head> tags as shown below:

```
<style type = "text/css">
// Set the color of alternating rows
  .wijmo-wijgrid-alternatingrow
    {
         background-color:pink!important;
     }
</style>
```

In CSS

By default, the “aristo” theme supports alternate row coloring in C1GridView. In order to modify any existing theme to get alternate row styles follow the steps below:

1. Open the CSS file of the theme to be modified from the following link: http://wijmo.com/theming/
2. Create a new file in a text editor and copy the contents of the theme’s CSS file. Modify the code to set the color of the alternating rows as defined above under **In Source View** heading.
3. Save the file with “.css” extension.

4. Add the CSS file into your project.
5. Link the CSS file by adding the following code within the <head></head> tags in the source view as shown below:

```
<link rel="stylesheet" type="text/css" href="name.css">
```

Note: The stylesheet is the CSS file inside the "custom-theme" folder.

What You've Accomplished

When you run the project, the color of alternating rows is changed.

CategoryID	CategoryName	Description
1	Beverages	Soft drinks, coffees, teas, beers, and ales
2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
3	Confections	Desserts, candies, and sweet breads
4	Dairy Products	Cheeses
5	Grains/Cereals	Breads, crackers, pasta, and cereal
6	Meat/Poultry	Prepared meats
7	Produce	Dried fruit and bean curd
8	Seafood	Seaweed and fish

Setting Conditional Formatting

In some cases you may want to change the grid's appearance based on given conditions.

In Code

You can change the appearance of grid rows and cells matching specific criteria using the RowDataBound event.

To change the color of a specific row or a cell's font using the RowDataBound event, complete the following steps:

1. Specify the row number you want to change using the RowIndex property.
2. Set the desired color of the **C1GridViewRow.BackColor** property.

   For example, add the following code to the RowDataBound event:

   To write the code in Visual Basic:

```
Visual Basic
Protected Sub C1GridView1_RowDataBound(ByVal sender As Object, ByVal e As
C1.Web.UI.Controls.C1GridView.C1GridViewRowEventArgs) Handles C1GridView1.RowDataBound
    If (e.Row.RowIndex = 2) Then
        e.Row.BackColor = System.Drawing.Color.Red
    End If
 End Sub
```

   To write the code in C#:

```
Visual Basic
private void C1GridView1_RowDataBound(object sender,
C1.Web.UI.Controls.C1GridView.C1GridViewRowEventArgs e)
    {
        if ((e.Row.RowIndex == 2)) {
        e.Row.BackColor = System.Drawing.Color.Red;
    }
}
```

3. You can also change the color of the font used in a specific cell by specifying the text in the cell and the

desired color.

For example, add the following code to the RowDataBound event:

To write the code in Visual Basic:

```
Visual Basic
Protected Sub C1GridView1_RowDataBound(ByVal sender As Object, ByVal e As
C1.Web.UI.Controls.C1GridView.C1GridViewRowEventArgs) Handles C1GridView1.RowDataBound
    If (e.Row.Cells(0).Text = "Chang") Then
        e.Row.Cells(0).ForeColor = System.Drawing.Color.Green
    End If
End Sub
```

To write the code in C#:

```
C#
private void C1GridView1_RowDataBound(object sender,
C1.Web.UI.Controls.C1GridView.C1GridViewRowEventArgs e)
{
    if ((e.Row.Cells[0].Text == "Chang")) {
        e.Row.Cells[0].ForeColor = System.Drawing.Color.Green;
    }
}
```

Note: You may need to change the column index in the code. This code changes the font color of the cell consisting of the text "Chang" to green.

What You've Accomplished

The first code snippet changes the background color of the third row to red.

Product	Category	Price	Stock	Discontinued
Chai	Beverages	$18.00	39 unit(s)	☐
Chang	Beverages	$19.00	17 unit(s)	☐
Aniseed Syrup	Condiments	$10.00	13 unit(s)	☐
Chef Anton's Cajun Seasoning	Condiments	$22.00	53 unit(s)	☐
Chef Anton's Gumbo Mix	Condiments	$21.35	0 unit(s)	☑
Grandma's Boysenberry Spread	Condiments	$25.00	120 unit(s)	☐

The second code snippet changes the color of a specific row or a cell's font using the RowDataBound event.

Product	Category	Price	Stock	Discontinued
Chai	Beverages	$18.00	39 unit(s)	☐
Chang	Beverages	$19.00	17 unit(s)	☐
Aniseed Syrup	Condiments	$10.00	13 unit(s)	☐
Chef Anton's Cajun Seasoning	Condiments	$22.00	53 unit(s)	☐
Chef Anton's Gumbo Mix	Condiments	$21.35	0 unit(s)	☑
Grandma's Boysenberry Spread	Condiments	$25.00	120 unit(s)	☐

Creating a Scrollable Grid

Complete the followingsteps to enable vertical and horizontal scrolling in a C1GridView control.

In the Design View

1. Select the GridView control and right click to open Properties > Properties Window.
2. In the Properties Window, Expand Scrolling Settings and change the **Mode** to *Both*.
3. Enter **250px** in the Height textbox and **450px** in the Width textbox.

In Code

To write the code in Visual Basic:

```
Visual Basic
' Set the grid's height and width.
 C1GridView1.Height = 250
 C1GridView1.Width = 450
' Turn scrolling on and set both horizontal and vertical scrolling.
 C1GridView1.ScrollingSettings.Mode = C1.Web.Wijmo.Controls.C1GridView.ScrollMode.Both
```

To write the code in C#:

```
C#
// Set the grid's height and width.
 C1GridView1.Height = 250;
 C1GridView1.Width = 450;
// Turn scrolling on and set both horizontal and vertical scrolling.
 C1GridView1.ScrollingSettings.Mode = C1.Web.Wijmo.Controls.C1GridView.ScrollMode.Both;
```

In Source View

Modify the ScrollingSettings property within the <cc1:C1GridView> to enable both horizontal and vertical scrolling.

```
<cc1:C1GridView ID="C1GridView1" runat="server" Height="250px" Width="450px">
   <ScrollingSettings Mode="Both">
   </ScrollingSettings>
```

Note that the **Height** and **Width** properties must be specified in order for the scrollbars to appear and scroll properly.

Setting the Footer Text

This topic demonstrates how to show a footer and set its text in C1GridView at design time and programmatically.

In the Designer

Complete the following steps:

1. Right-click the C1GridView control and then click **Show Smart Tag**. On the **C1GridView Tasks** menu, click **Property builder**. The **C1GridView Properties** window appears.
2. On the **General** tab, check the **Show footer** check box.
3. Select the **Columns** tab and deselect **Create columns automatically at run time** if it is selected.
4. Choose a column in the list of **Selected columns**.
5. Enter the desired text in the FooterText property.

In Source View

Switch to Source view and complete the following: steps:

1. Set the following text to the <cc1:C1GridView> tag:
   - Set the AutoGenerateColumns property to False.
   - Set the ShowFooter property to True.

   It will appear similar to the following:
   ```
   <cc1:C1GridView ID="C1GridView1" runat="server" DataSourceID="AccessDataSource1" AutogenerateColumns="False" ShowFooter="True">
   ```
2. Set the footer text for individual columns using the FooterText property by adding the following FooterText="Total Price" text within a <cc1:C1BoundField> tag, so it appears similar to the following:
   ```
   <cc1:C1BoundField DataField="ProductName" HeaderText="ProductName" SortExpression="ProductName" FooterText="Footer">
   ```
   This will set the FooterText property.

In Code

Open the **Code Editor** and set the following properties in code:

- Set the AutoGenerateColumns property to **False**.
- Set the ShowFooter property to **True**.
- Set the footer text for individual columns using the FooterText property. This property must be set before the **DataBind** method is called.

For example, add the following code to the **Page_Load** event:

To write the code in Visual Basic:

```
Visual Basic
C1GridView1.AutoGenerateColumns = False
 C1GridView1.ShowFooter = True
 C1GridView1.Columns(0).FooterText = "Footer"
```

To write the code in C#:

```
C#
C1GridView1.AutoGenerateColumns = false;
 C1GridView1.ShowFooter = true;
 C1GridView1.Columns[0].FooterText = "Footer";
```

What You've Accomplished
This example sets the footer text for the first column to "Footer".

Note: The footer text can only be set for columns that are not automatically generated.

Creating a Non Scrollable Row or Column

You can use the StaticRowIndex or StaticColumnIndex to freeze a row/column in order to prevent it from scrolling. An integer value to determine the row/column number to be frozen is set in this property.

The following example fixes the first row in the grid so that it will not scroll.

In the Designer

Complete the following steps:

1. Create a scrollable grid. See Creating a Scrollable Grid for further details.
2. Right click the grid and choose **Properties** to view the associated properties and events of the C1GridView.
3. Set the StaticRowIndex property to the index value of the row that has to be frozen.
4. Click **OK** to save the setting and close the properties dialog.

In Source View

Complete the following steps:

1. Create a scrollable grid. See Creating a Scrollable Grid for further details.
2. Switch to Source view.
3. Set the StaticRowIndex property of C1GridView to the index value of the column to be frozen by adding **StaticRowIndex='0'** to the <cc1: C1GridView> tag as shown below:

```
<cc1:C1GridView ID="C1GridView1" runat="server" StaticRowIndex='0'>
```

This keeps the first row fixed when the grid is scrolled vertically.

In Code

Complete the following steps:

1. Create a scrollable grid. See Creating a Scrollable Grid for further details.
2. Set the first column's StaticRowIndex property to 0. This keeps the first row fixed when the grid is scrolled vertically.

To write the code in Visual Basic:

Visual Basic

```
' Fix the top row of the grid when scrolling vertically.
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
    C1GridView1.StaticRowIndex = 0
End Sub
```

To write the code in C#:

C#

```
// Fix the top row of the grid when scrolling vertically.

protected void Page_Load(object sender, EventArgs e)

{>

C1GridView1.StaticRowIndex = 0;

>}
```

Run your application and scroll the grid. Note that the first row does not scroll with the other rows of the grid.

Follow the same process using the StaticColumnIndex property to create non-scrollable columns.

Adding Controls to a Column

Adding arbitrary controls to grid columns is simple using template columns. The following task-based help topics will walk you through adding templates columns, adding controls to template columns, and binding controls in template columns.

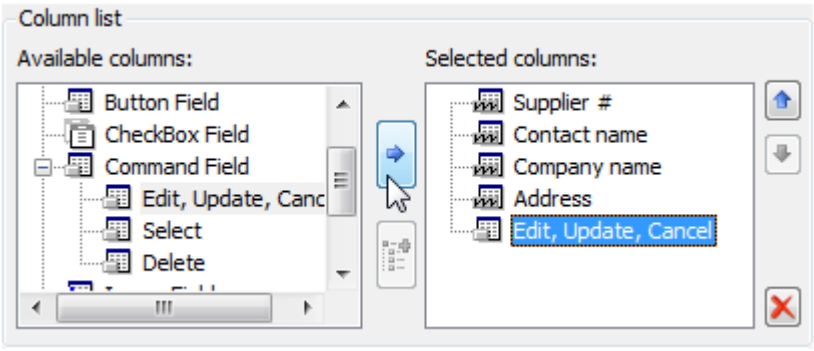
Adding Template Columns

To add template columns to C1GridView, complete the following steps:

1. Right-click the C1GridView control and then click **Show Smart Tag**. On the **C1GridView Tasks** menu, click **Property builder**. The **C1GridView Properties** window appears.
2. Click the **Columns** tab.
3. Click the  button to remove all columns from the **Selected columns** list.
4. Select **Template Field** from the **Available columns** list.
5. Click the arrow button between the column lists to copy the **Template Column** to the **Selected columns** list. Repeat this task three times to create three more (and a total of four) Template Columns.
6. Under **Column properties**, set each column's **HeaderText** property to "Supplier #", "Contact name", "Company name", and "Address" respectively and click **Apply**.

   > **Note:** To view changes to each Template Column, you must click the **Apply** button.

7. From the **Available columns** list, expand the **Command Field** node and select **Edit**, **Update**, **Cancel**.
8. Click the arrow button between the column lists to copy the **Edit**, **Update**, **Cancel** into the **Selected columns** list.The **Selected columns** list should look like this:



9. Click **OK** to close the **C1GridView Properties** dialog box.

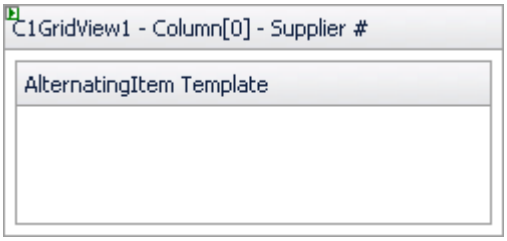
Binding Template Columns

This topic explains how to bind the following to a data source: a non-editable template column and a template column that can be edited at run time. Note that this topic assumes you have completed the Adding Template Columns topic.

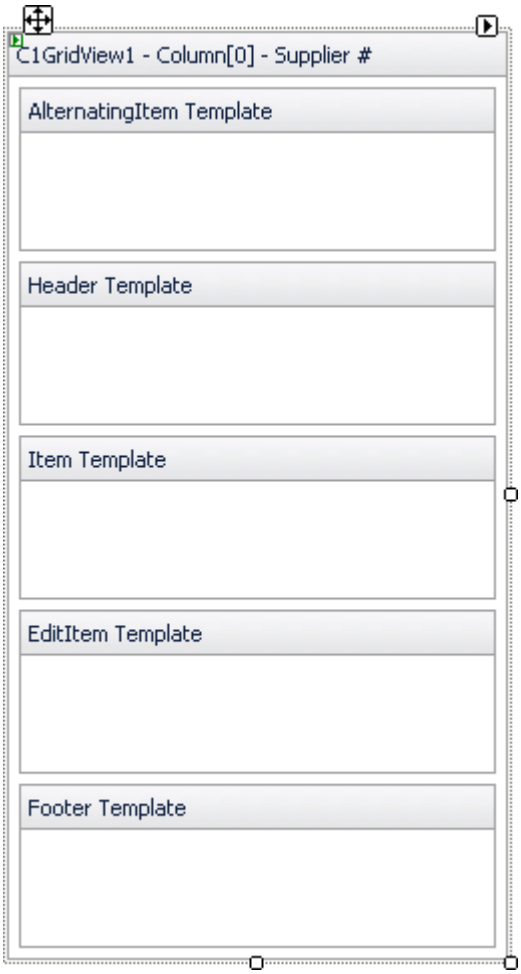
Binding a non-editable template column

To bind a non-editable template column *Supplier #* to a data source, complete the following steps:

1. Right-click the C1GridView control and then click **Show Smart Tag**.
2. On the **C1GridView Tasks** menu, click **Edit Templates**. The template editor appears:



3. Click the **C1GridView Smart Tag** and select **Column[0] – Supplier #** from the **Display** drop-down list. The template editor appears:



4. From the **Standard** tab in the Toolbox, drag a **Label** control into the **Item Template** section.
5. If the **Label Tasks** menu does not appear, click the **Label1 Smart Tag**, and then click **Edit DataBindings**. The **Label1 DataBindings** window appears.
6. Select **Text** in the **Bindable properties** list, and then select the *SupplierID* field from the **Bound to** box.
7. Click **OK**.
8. Click the **C1GridView Smart Tag**, and select **End Template Editing** from the **C1GridView Tasks** menu.

Binding editable template columns

To bind editable template columns *Contact name*, *Company name*, and *Address to a data source*, complete the following steps:

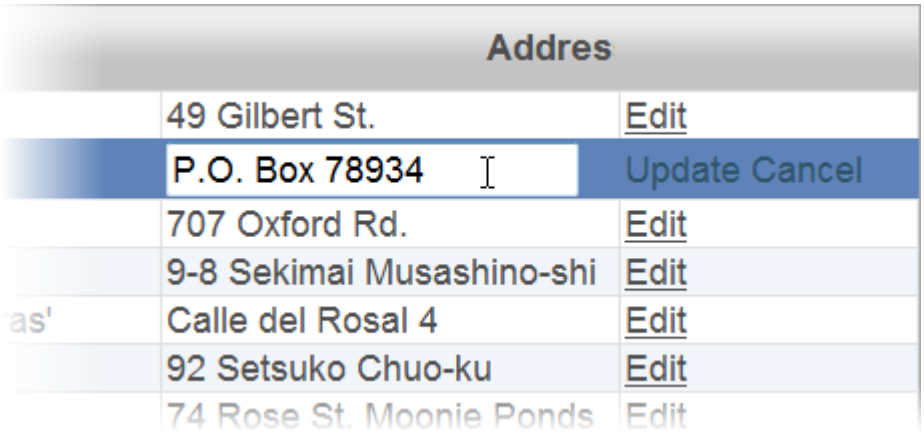
1. Click C1GridView's smart tag and select **Edit Templates**.
2. Select **Column[1] – Contact name** from the **Display** drop-down list.
3. From the **Standard** tab in the Toolbox, drag a **Label** control into the **Item Template** section.
4. If the **Label Tasks** menu does not appear, click the **Label2 Smart Tag**, and then click **Edit DataBindings**. The

Label2 DataBindings window appears.

5. Select **Text** in the **Bindable properties** list, and then select the *ContactName* field from the **Bound to combo** box.
6. Click **OK**.
7. From the **Standard** tab in the Toolbox, drag a **TextBox** control into the **EditItem Template** section.
8. If the **TextBox Tasks** menu does not appear, click the **TextBox1 Smart Tag**, and then click **Edit DataBindings**.
9. Select **Text** in the **Bindable properties** list, and then select the *ContactName* field from the **Bound to** check box.
10. Make sure that the **Two-way databinding** check box is selected and click **OK**.
11. Repeat the steps above for the *Company name* and *Address* columns. Bind them to the *CompanyName* and *Address* fields accordingly.
12. Click the **C1GridView Smart Tag** and select **End Template Editing** from the **C1GridView Tasks** menu.

What You've Accomplished

Run the project. When you click the **Edit** button in the first row, an editable text box appears for each column except for the *Supplier #* column. You can edit the information, and click **Update** to update the database or click **Cancel** to ignore any edits you made.



Adding CheckBox or ListBox Controls to a Column

To use the standard ASP.NET **CheckBox** and **ListBox** controls to display data for a column, complete the following steps:

1. Right-click C1GridView and select Show Smart Tag from the context menu. From the **C1GridView Tasks** menu, select **Property builder**. The **C1GridView Properties** dialog box appears.
2. Click the **Columns** tab in the left pane.
3. Under **Column List**, select **Template Field** from the list of **Available columns**.
4. Click the arrow button between the column lists to copy the **Template Field** to the list of **Selected columns**.
5. Select the new **Template Field** and set the desired properties under **Column** properties. This might include adding a header or footer to the column, fixing the column's size or position, and so on.
6. Click **OK** to return to the form.
7. Click the **C1GridView Smart Tag** and select **Edit Templates** from the **C1GridView Tasks** menu.
8. Click the **Display** drop-down arrow and choose the new **Template Field** you created.
   The template editor Document appears.
9. Select the ItemTemplate section.
10. Double-click the **CheckBox** or **ListBox** control in the Visual Studio Toolbox to add it to the ItemTemplate section. Note that you can format the **CheckBox** or **ListBox** using the Properties window.
11. Click the C1GridView smarttTag and select **End Template Editing** from the **C1GridView Tasks** menu.

Adding Input for ASP.NET Web Forms Controls

GridView for ASP.NET Web Forms is fully compatible with **Input for ASP.NET Web Forms** controls. Using the **Input for ASP.NET Web Forms** controls, you can easily add in data validation to your grid, including masked, date, numeric, percent, and currency editing.

> **Note:** For more information about **Input for ASP.NET Web Forms**, see the Input for ASP.NET Web Forms documentation.

In the following steps you'll create template columns in the C1GridView control and add the **C1InputNumeric** and **C1InputCurrency** controls to those columns. Note that in the follow steps it is assumed that you have created a C1GridView control and bound it to the *Products* table of the NorthWind database.

Complete the following steps:

1. Right-click C1GridView and select **Show Smart Tag** from the context menu. From the **C1GridView Tasks** menu, select **Property builder**. The **C1GridView Properties** dialog box appears.
2. Click the *Columns* tab in the left pane.
3. If the **Create columns automatically at run time check box** is selected, deselect it.
4. In the **Selected columns** list, remove the following columns by selecting the column and clicking the **Delete** button: *ProductID*, *SupplierID*, *CategoryID*, *UnitPrice*, *UnitsInStock*, *UnitsOnOrder*, and *ReorderLevel*.
   Note that you'll add template columns to replace most of these columns.
5. In the **Available columns** list, select *Template Field* from the list of column types.
6. Click the arrow button between the column lists to copy the *Template Field* to the list of **Selected columns**.
   Repeat this step two more times, so that you've added a total of three template columns.
7. Set the **HeaderText** property for each template column to "Price", "Stock", and "Ordered", respectively.
8. Using the **Down** arrow button, move the **Discontinued column** to the bottom of the **Selected columns** list.
9. Click **OK** to return to close the **C1GridView Properties** dialog box and return to the form.
10. Click the **C1GridView** smart tag and select **Edit Templates** from the **C1GridView Tasks** menu.
11. Add the **C1InputCurrency** control to the *Price* column:
    a. In the **C1GridView Tasks** menu, click the **Display** drop-down arrow and under *Column[2] – Price*, choose **Item Template**. The Item Template editor appears.
    b. Click in the body area of the template, navigate to the Visual Studio Toolbox, and double-click the **C1InputCurrency** control to add it to the **C1TemplateField.ItemTemplate**.
    c. Click the **C1InputCurrency** control's smart tag, and select **Edit DataBindings** from the **C1InputCurrency Tasks** menu. This will open the **C1InputCurrency DataBindings** dialog box.
    d. In the **C1InputCurrency DataBindings** dialog box, select the **Show all properties check box** if it is not already selected.
    e. Select **Value** in the list of **Bindable properties**, choose the **Field binding** radio button, and in the **Bound to** drop-down list, select **UnitPrice**.
    f. Click **OK** to save your settings and close the **C1InputCurrency DataBindings** dialog box.
    g. Click once on the **C1InputCurrency** control to select it and in the Properties window set the **Width** property to "60px".
12. Add the **C1InputNumeric** control to the *Stock* column:
    a. Click the **C1GridView** smart tag, click the **Display** drop-down arrow, and under *Column[3] – Stock*, choose *Item Template*.
    b. Click in the body area of the template, navigate to the Visual Studio Toolbox, and double-click the **C1InputNumeric** control to add it to the **C1TemplateField.ItemTemplate**.
    c. Click the **C1InputNumeric** control's smart tag, and select **Edit DataBindings** from the **C1InputNumeric Tasks** menu. This will open the **C1InputNumeric DataBindings** dialog box.
    d. In the **C1InputCurrency DataBindings** dialog box, select the **Show all properties** check box if it is not already selected.
    e. Select **Value** in the list of **Bindable properties**, choose the **Field binding** radio button, and in the **Bound to** drop-down list, select *UnitsInStock*.

f. Click **OK** to save your settings and close the **C1InputNumeric DataBindings** dialog box.
g. Click once on the **C1InputNumeric** control to select it and in the Properties window set the **Width** property to "60px".
h. Click the **C1InputNumeric** control's smart tag and, in the **C1InputNumeric Tasks** menu, enter "0" in the **DecimalPlaces** text box.

13. Add the **C1InputNumeric** control to the *Ordered* column:
    a. Click the C1GridView smart tag, click the **Display** drop-down arrow, and under *Column[4] – Ordered*, choose **Item Template**.
    b. Click in the body area of the template, navigate to the Visual Studio Toolbox, and double-click the **C1InputNumeric** control to add it to the **C1TemplateField.ItemTemplate**.
    c. Click the **C1InputNumeric** control's smart tag, and select **Edit DataBindings** from the **C1InputNumeric Tasks** menu. This will open the **C1InputNumeric DataBindings** dialog box.
    d. In the **C1InputCurrency DataBindings** dialog box, select the **Show all properties** check box if it is not already selected.
    e. Select **Value** in the list of **Bindable properties**, choose the **Field binding** radio button, and in the **Bound to drop**-down list, select *UnitsOnOrder*.
    f. Click **OK** to save your settings and close the **C1InputNumeric DataBindings** dialog box.
    g. Click once on the **C1InputNumeric** control to select it and in the Properties window set the **Width** property to "60px".
    h. Click the **C1InputNumeric** control's smart tag and, in the **C1InputNumeric Tasks** menu, enter "200" in the **MaxValue** text box.
    i. Click the C1GridView smart tag and select **End Template Editing** from the **C1GridView Tasks** menu.

What You've Accomplished

Run your application and observe that the grid now uses **Input for ASP.NET Web Forms** controls in the *Price*, *Stock*, and *Ordered columns*:

ProductName	QuantityPerUnit	Price	Stock	Ordere
Chai	10 boxes x 20 bags	$18.00	39	0.00
Chang	24 - 12 oz bottles	$19.00	17	40.00
Aniseed Syrup	12 - 550 ml bottles	$10.00	13	70.00
Chef Anton's Cajun Seasoning	48 - 6 oz jars	$22.00	53	0.00
Chef Anton's Gumbo Mix	36 boxes	$21.35	0	0.00
Grandma's Boysenberry Spread	12 - 8 oz jars	$25.00	120	0.00
Uncle Bob's Organic Dried Pears	12 - 1 lb pkgs.	$30.00	15	0.00

Setting Column Width

This topic demonstrates how to set the width of a column in C1GridView.

A column's width can be specified in code if the column is not automatically generated. To set the column width:

1. Set the C1GridView.AutoGenerateColumns property of C1GridView to False.
2. Specify the columns to appear in C1GridView. See Hiding Specified Columns for more information.
3. Set the width of the third column, for example, via the **Code Editor** or at design time:
   - Add the following code after the DataBind method is called:

To write the code in Visual Basic:

```
Visual Basic
C1GridView1.Columns(2).ItemStyle.Width = New Unit(500)
```

To write the code in C#:

C#

```
C1GridView1.Columns[2].ItemStyle.Width = new Unit(500);
```

- From the Property builder, select **Format** tab in the left pane and select a column from the list of **Columns** on the right. Change the **Width** property to **500 Pixels**, and click **OK**.
  OR
- Edit the column's markup to include a **Width** element:
  `wijmo:C1BoundField DataField="ContactName" HeaderText="ContactName" SortExpression="ContactName" Width="500">`

What You've Accomplished

In this topic you learned how to change the width of a column. In this example, the third column is 500 pixels:

SupplierID	CompanyName	ContactName	Address
1	Exotic Liquids	Charlotte Cooper	49 Gilbert St.
2	New Orleans Cajun Delights	Shelley Burke	P.O. Box 78934
3	Grandma Kelly's Homestead	Regina Murphy	707 Oxford Rd.
4	Tokyo Traders	Yoshi Nagase	9-8 Sekimai Musashino-shi
5	Cooperativa de Quesos 'Las Cabras'	Antonio del Valle Saavedra	Calle del Rosal 4
6	Mayumi's	Mayumi Ohno	92 Setsuko Chuo-ku
7	Pavlova, Ltd.	Ian Devling	74 Rose St. Moonie Ponds
8	Specialty Biscuits, Ltd.	Peter Wilson	29 King's Way
9	PB Knäckebröd AB	Lars Peterson	Kaloadagatan 13
10	Refrescos Americanas LTDA	Carlos Diaz	Av. das Americanas 12.890
11	Heli Süßwaren GmbH & Co. KG	Petra Winkler	Tiergartenstraße 5
12	Plutzer Lebensmittelgroßmärkte AG	Martin Bein	Bogenallee 51
13	Nord-Ost-Fisch Handelsgesellschaft mbH	Sven Petersen	Frahmredder 112a
14	Formaggi Fortini s.r.l.	Elio Rossi	Viale Dante, 75
15	Norske Meierier	Beate Vileid	Hatlevegen 5
16	Bigfoot Breweries	Cheryl Saylor	3400 - 8th Avenue Suite 210
17	Svensk Sjöföda AB	Michael Björn	Brovallavägen 231
18	Aux joyeux ecclésiastiques	Guylène Nodier	203, Rue des Francs-Bourgeois
19	New England Seafood Cannery	Robb Merchant	Order Processing Dept. 2100 Paul Revere Blvd.
20	Leka Trading	Chandra Leka	471 Serangoon Loop, Suite #402
21	Lyngbysild	Niels Petersen	Lyngbysild Fiskebakken 10
22	Zaanse Snoepfabriek	Dirk Luchte	Verkoop Rijnweg 22
23	Karkki Oy	Anne Heikkonen	Valtakatu 12
24	G'day, Mate	Wendy Mackenzie	170 Prince Edward Parade Hunter's Hill
25	Ma Maison	Jean-Guy Lauzon	2960 Rue St. Laurent
26	Pasta Buttini s.r.l.	Giovanni Giudici	Via dei Gelsomini, 153
27	Escargots Nouveaux	Marie Delamare	22, rue H. Voiron
28	Gai pâturage	Eliane Noz	Bat. B 3, rue des Alpes
29	Forêts d'érables	Chantal Goulet	148 rue Chasseur

Merging Rows

This topic demonstrates how to merge rows containing identical text in C1GridView.

The RowMerge property allows the selected column to merge rows with the same text. For example in the following image, setting the RowMerge property to **Free** allows the *Property* column to merge the rows with the same text:

Name	OrderID	Quantity	Totals
Chai	10,317	20	288.00
	10,285	45	648.00
	10,294	18	259.20
	10,348	15	216.00
	10,354	12	172.80
Chang	10,264	35	532.00
	10,298	40	608.00
	10,258	50	760.00
	10,255	20	304.00
	10,327	25	380.00
	10,335	7	106.40
	10,342	24	364.80

In Code

Use the RowMerge property of the C1GridView class to determine if and how rows are merged:

To write the code in Visual Basic:

Visual Basic

```
CType(C1GridView1.Columns(0), C1.Web.Wijmo.Controls.C1GridView.C1Field).RowMerge = C1.Web.Wijmo.Controls.C1GridView.RowMerge.Free
```

To write the code in C#:

C#

```
((C1.Web.Wijmo.Controls.C1GridView.C1Field)C1GridView1.Columns[0]).RowMerge = C1.Web.Wijmo.Controls.C1GridView.RowMerge.Free;
```

In Source View

Set the RowMerge property in the column's markup:

```
<cc1:C1GridView ID="C1GridView1" runat="server" DataSourceID="SqlDataSource1" AutoGenerateColumns="false">
    <Columns>
        <cc1:C1BoundField DataField="ProductName" SortExpression="ProductName" HeaderText="Name" RowMerge="Free" />
        <cc1:C1BoundField DataField="OrderID" SortExpression="OrderID" HeaderText="OrderID" />
```

```
        <cc1:C1BoundField DataField="Quantity" SortExpression="Quantity" HeaderText="Quantity" />
        <cc1:C1BoundField DataField="Total" SortExpression="Total" HeaderText="Totals" />
    </Columns>
</cc1:C1GridView>
```

In Design View

The RowMerge property can also be accessed in the **Property Builder**. Simply select a column from the list of **Selected columns**, click the drop-down arrow next to the RowMerge property and choose **Free** or **Restricted**. The default is **None**.

Sample Project Available
For the complete sample, see the Merging page located in the ControlExplorer sample. See GridView for ASP.NET Web Forms Samples for more information.

Adding Paging

This topic demonstrates how to display the contents of C1GridView on multiple pages, navigate through those pages and set the number of items to be displayed on each page. For more information, see the Paging topic.

In the Designer

Complete the following steps:

1. Right-click the C1GridView control and then click **Show Smart Tag**. On the **C1GridView Tasks** menu, click Property builder. The **C1GridView Properties** window appears.
2. Select the **Paging** tab and check the **Allow paging** check box. This sets the AllowPaging property to **True**.
3. Specify the number of rows to appear on the page in the **Page size** text box, for example "4".
4. The **Show navigation** check box under **Page navigation** is selected by default. This places the page navigation buttons on the page. You can specify their location on the page by clicking the Position drop-down arrow and selecting *Bottom*, *Top*, or *TopAndBottom*. The Mode property is set to *Numeric* by default, so numeric navigation buttons appear on the page.

In Source View

Complete the following steps:

1. Add **AllowPaging**="True" and PageSize="4" to the <cc1:C1GridView> tag to set the AllowPaging property to True and the PageSize property to move through the pages of data from the data source four items at a time:
   ```
   <cc1:C1GridView ID="C1GridView1" runat="server" AutoGenerateColumns="False" DataSourceID="AccessDataSource1"
      VisualStylePath="~/C1WebControls/VisualStyles" AllowPaging="True" PageSize="4">
   ```
   The PagerSettings.Mode property is set to Numeric by default, so numeric navigation buttons appear on the page.
2. If you want to change the position of the navigation buttons, set the Position to *Top* or *TopAndBottom* by adding <PagerSettings Position="TopAndBottom"/> or <PagerSettings Position="TopAndBottom"/> between the <cc1:C1GridView> and </cc1:C1GridView> tags.

In Code

Complete the following steps:

1. Set AllowPaging property to True and PageSize to 4 to move through the pages of data from the data source four items at a time. The Mode property is set to *Numeric* by default, so numeric navigation buttons appear on the page.

   To write the code in Visual Basic:

Visual Basic
C1GridView1.AllowPaging = True C1GridView1.PageSize = 4

   To write the code in C#:

C#
C1GridView1.AllowPaging = true; C1GridView1.PageSize = 4;
2. If you want to change the position of the navigation buttons, set the Position to *Top* or *TopAndBottom*.

What You've Accomplished

Run your application and observe that paging controls appear at the bottom of the grid, and that 4 items are displayed on each page:

Tracking the Current Cell Position

By default client-side selection information is unavailable on the server. However, you can track when the current cell position changes and send this data

to server at postback.

For example, complete the following steps:

1. Add the following script to your application's markup:
```
<script type="text/javascript">
    function onCurrentCellChanged(e, args) {
        $("#currentCellValue").val($(e.target).c1gridview("currentCell").value());
    }
</script>
```
2. Edit the C1GridView control's markup so it appears similar to the following:
```
<cc1:C1GridView ID="C1GridView1" runat="server" DataSourceID="AccessDataSource1" AutogenerateColumns="false"
    OnClientCurrentCellChanged="onCurrentCellChanged">
        <Columns>
            <cc1:C1BoundField DataField="OrderID" HeaderText="OrderID" />
            <cc1:C1BoundField DataField="Quantity" HeaderText="Quantity" />
        </Columns>
    </cc1:C1GridView>
```
3. Right-click the window and select **View Code** to switch to Code view.
4. Add the following code to the **Page_Load** event in your application:

To write the code in Visual Basic:

Visual Basic
```
Page.ClientScript.RegisterHiddenField("currentCellValue", Nothing)
 If IsPostBack Then
  Dim currentCellValue As String = Page.Request("currentCellValue")
 End If
```

To write the code in C#:

C#
```
Page.ClientScript.RegisterHiddenField("currentCellValue", null);
 if (IsPostBack)
 {
    string currentCellValue = Page.Request["currentCellValue"];
 }
```

This code allows you to track when the current cell position changes and send this data to server at postback.

Getting a Value from a Cell

You can easily read the value of a cell in a grid by using the NewEditIndex property to get the index of the row being edited.

For example, use the following code:

To write the code in Visual Basic:

Visual Basic
```
' Get the index of the row being edited.
Protected Sub C1GridView1_RowEditing(sender As Object, e As
C1.Web.Wijmo.Controls.C1GridView.C1GridViewEditEventArgs)
        Handles C1GridView1.RowEditing
        Dim idx As Integer
        Dim id As String
        idx = e.NewEditIndex
' Get the text in a cell in the edited row.
        id = C1GridView1.Rows(idx).Cells(0).Text
End Sub
```

To write the code in C#:

C#

```
// Get the index of the row being edited.
protected void C1GridView1_RowEditing(object sender,
C1.Web.Wijmo.Controls.C1GridView.C1GridViewEditEventArgs e)
     {
       int idx = 0;
       string id;
       idx = e.NewEditIndex;

       // Get the text in a cell in the edited row.
       id = C1GridView1.Rows[idx].Cells[0].Text;
      }
```

The above code will get a numeric value in a specified column in the row that has been edited.

Updating the Grid with AJAX

The following task-based help items demonstrate setting **C1GridView** properties to use AJAX to update the grid, including moving columns, paging, sorting, filtering and more.

Note that multiple Actions may be selected for a grid, or you can set the Action property to **All** for all actions on the grid to be performed via callbacks.

Moving Columns

You can use AJAX to update the grid when the user moves columns at run time. For example, you can update the grid with AJAX on column move in the Designer, in Source view, and in code.

In the Designer

To enable AJAX when moving a column, complete the following steps:

1. Select the C1GridView control and set the AllowColMoving property to True in the Properties window. If you run the program now, you will notice that the whole page refreshes when you move a column.
2. In the project, with C1GridView still selected, expand the **CallbackSettings** node, click the drop-down arrow next to the Action property in the Properties window, and check the **ColMove** check box.

In Source View

Switch to Source view and add AllowColMoving="True" and CallbackOptions="ColMove" to the <cc1:C1GridView> tag, so it appears similar to the following:

```
<cc1:C1GridView ID="C1GridView1" runat="server" AutoGenerateColumns="False" DataSourceID="AccessDataSource1" VisualStylePath="~/C1WebControls/VisualStyles" AllowColMoving="True" CallbackOptions="ColMove">
```

In Code

To enable AJAX when moving a column, add the following code to the **Page_Load** event:

To write the code in Visual Basic:

Visual Basic

```
C1GridView1.AllowColMoving = True
C1GridView1.CallbackOptions = CallbackOptions.ColMove
```

To write the code in C#:

C#

```
C1GridView1.AllowColMoving = true;
C1GridView1.CallbackOptions = CallbackOptions.ColMove;
```

What You've Accomplished

Run the program and move a column. You will notice only the grid refreshes, not the entire page.

Editing a Record

You can use AJAX to update the grid when the user edits a record at run time. For more information about editing records, see the Editing Rows topic.

To enable AJAX when editing a record, complete the following steps:

1. Select the C1GridView control and click the Smart Tag to open the **C1GridView Tasks** menu.
2. Select **Property builder**. The **C1GridView Properties** dialog box (the Property builder) appears.
3. Click the *Columns* tab and expand the *Command Column* node in the list of **Available columns**.
4. Click **Edit, Update, Cancel** and then click the arrow button to add the **Edit, Update, Cancel** button column to the list of **Selected columns**.
5. Click **OK** to close the Property builder.

6. With the C1GridView control still selected, expand the **CallbackSettings** node, click the drop-down arrow next to the Action property in the Properties window, and check the **Editing** check box.

What You've Accomplished

Run the program and notice that only the grid refreshes when you click the **Edit** button. The entire page does not have to reload.

Paging the Grid

You can use AJAX to update the grid when the user pages the grid at run time. For example, you can update the grid with AJAX on grid paging in the Designer, in Source view, and in code. For more information about paging, see Paging and Creating a Pageable Grid.

In the Designer

To enable AJAX when paging through a C1GridView, complete the following steps:

1. Select the C1GridView control and click the Smart Tag to open the **C1GridView Tasks** menu.
2. Select **Property builder**. The **C1GridView Properties** dialog box (the Property builder) appears.
3. Click the **Paging** tab and select the **Allow paging** check box in the **Paging** section.
4. Click **OK** to close the Property builder.
5. With the C1GridView control still selected, expand the *CallbackSettings* node, click the drop-down arrow next to the Action property in the Properties window, and check the **Paging** check box.

In Source View

Switch to Source view and add AllowPaging="True" and CallbackOptions="Paging" to the <cc1:C1GridView> tag, so it appears similar to the following:

```
<cc1:C1GridView ID="C1GridView1" runat="server" AutoGenerateColumns="False" DataSourceID="AccessDataSource1" VisualStylePath="~/C1WebControls/VisualStyles" AllowPaging="True" CallbackOptions="Paging">
```

In Code

To enable AJAX when filtering a column, add the following code to the **Page_Load** event:

To write the code in Visual Basic:

```
Visual Basic
C1GridView1.AllowPaging = True
C1GridView1.CallbackOptions = CallbackOptions.Paging
```

To write the code in C#:

```
C#
C1GridView1.AllowPaging = true;
C1GridView1.CallbackOptions = CallbackOptions.Paging;
```

What You've Accomplished

Run the program and click the paging navigation at the bottom of the grid. Notice that only the grid refreshes as you page through it. The entire page is not reloaded.

Selecting a Record

You can use AJAX to update the grid when the user selects a record at run time. To enable AJAX when selecting a record, complete the following steps:

1. Select the C1GridView control and click the Smart Tag to open the **C1GridView Tasks** menu.
2. Select **Property builder**. The **C1GridView Properties** dialog box (the Property builder) appears.
3. Click the *Columns* tab and expand the **Command Field** node in the list of **Available columns**.
4. Click **Select** and then click the arrow button to add a **Select button** column to the list of **Selected columns**.
5. Click the **Format** tab and choose **Selected Rows** under **Rows** in the list of Objects.
6. Click the **BackColor** ellipsis button and choose a color. This will highlight the selected record in the grid.
7. Click **OK** to close the Property builder.
8. With the **C1GridView** control still selected, expand the **CallbackSettings** node, click the drop-down arrow next to the CallbackSettings.Action property in the Properties window, and check the **Selection** check box.

What You've Accomplished

Run the program and notice that only the grid refreshes when you select a record in the grid.

Sorting Columns

You can use AJAX to update the grid when the user sorts a column at run time. For example, you can update the grid with AJAX on column sort in the Designer, in Source view, and in code. For more information about sorting, see the Sorting topic.

In the Designer

To enable AJAX when sorting on a column, complete the following steps:

1. Select the C1GridView control and click the Smart Tag to open the **C1GridView Tasks** menu.
2. Select **Property builder**. The **C1GridView Properties** dialog box (the Property builder) appears.
3. Click the **General** tab and select the **Allow sorting** check box in the Behavior section.
4. Click **OK** to close the Property builder. If you run the program now, you will notice that the whole page refreshes when you sort a column.
5. In the project, with the C1GridView control still selected, expand the **CallbackSettings** node, click the drop-down arrow next to the Action property in the Properties window, and check the **Sorting** check box.

In Source View

Switch to **Source** view and add AllowSorting="True" and CallbackOptions="Sorting" to the <cc1:C1GridView> tag, so it appears similar to the following:

```
<cc1:C1GridView ID="C1GridView1" runat="server" AutoGenerateColumns="False" DataSourceID="AccessDataSource1" VisualStylePath="~/C1WebControls/VisualStyles" AllowSorting="True" CallbackOptions="Sorting">
```

In Code

To enable AJAX when filtering a column, add the following code to the **Page_Load** event:

To write the code in Visual Basic:

```
Visual Basic
C1GridView1.AllowSorting = True
 C1GridView1.CallbackOptions = CallbackOptions.Sorting
```

To write the code in C#:

```
C#
C1GridView1.AllowSorting = true;
 C1GridView1.CallbackOptions = CallbackOptions.Sorting;
```

What You've Accomplished

Now when you run the program and sort a column, you will notice that only the grid, and not the entire page, refreshes.

Filtering Columns

You can use AJAX to update the grid when the user filters columns at run time. For example, you can update the grid with AJAX on filtering in the Designer, in Source view, and in code. For more information about filtering columns, see the Filtering topic.

In the Designer

To enable AJAX when filtering a column, complete the following steps:

1. Select the C1GridView control and navigate to the Properties window.
2. In the Properties window, set the ShowFilter property to True.
3. If you run the program now, you will notice that the whole page refreshes when you filter a column.
4. In the project, with the C1GridView control still selected, expand the CallbackSettings node, click the drop-down arrow next to the Action property in the Properties window, and check the **Filtering** check box.

In Source View

Switch to Source view and add **ShowFilter**="True" and **CallbackOptions**="Filtering" to the *<cc1:C1GridView>* tag, so it appears similar to the following:

```
<cc1:C1GridView ID="C1GridView1" runat="server" AutoGenerateColumns="False" DataSourceID="AccessDataSource1" VisualStylePath="~/C1WebControls/VisualStyles" ShowFilter="True" CallbackOptions="Filtering">
```

In Code

To enable AJAX when filtering a column, add the following code to the **Page_Load** event:

To write the code in Visual Basic:

```
Visual Basic
C1GridView1.ShowFilter = True
C1GridView1.CallbackOptions = CallbackOptions.Filtering
```

To write the code in C#:

```
C#
C1GridView1.ShowFilter = true;
C1GridView1.CallbackOptions = CallbackOptions.Filtering;
```

What You've Accomplished

Now when you run the program and filter a column, you will notice that only the grid refreshes.

Updating the Grid at Run Time

While you can use the RowUpdating event to edit the grid's underlying data source, another option is to use the client-side **Update()** method. The RowUpdating event gets fired when the client-side **Update()** method is called. In some situations you may prefer this method to send edits done by user to server in client-editing mode.

In this example, you'll add a button that updates the grid when clicked. Complete the following steps:

1. Select **View | Markup** to switch to Source view.
2. Add the following markup just above the *<cc1:C1GridView>* tag to add a button to the application.
   ```
   <asp:Button ID="btn1" runat="server" Text="Update" OnClientClick="btn_ClientClick()" />
   ```
3. Add the following markup inside the application's *<head></head>* tags:

```
<script type="text/javascript">
    function btn_ClientClick(sender, args) {
        var grid = $("#C1GridView1");
        grid.c1gridview("endEdit");
        grid.c1gridview("update");
    }
</script>
```

This script will update the grid when the button is clicked.

Enable CheckBox Filtering

This topic demonstrates how to use filters on a **CheckBox** column of a C1GridView control.

> **Note:** To use CheckBox Filters in the C1GridView control, the table in the database bound to the C1GridView must contain a yes/no (CheckBox) field.

In the Designer

Complete the following steps:

1. Drag and drop a C1GridView control to your Web Form.
2. Add data to the C1GridView. For more information, see Binding the Grid to a Data Source.
3. Click the Smart Tag on the top left corner of the C1GridView control to enable filters and select **Property Builder** from the drop-down menu.
4. In the **Property Builder** window, check the **ShowFilter** option and click **OK**. A filter bar appears at the top of the grid just under the grid's column headings.

In Source View

To enable filtering on the checkbox field, modify the *<cc1:C1GridView ></cc1:C1GridView >* tag as shown below:

```
<cc1:C1GridView ID="C1GridView1" runat="server" AutogenerateColumns="False" DataKeyNames="ID"
DataSourceID="SqlDataSource1" ShowFilter="True">
```

Set the ShowFilter property to True.

In Code

To enable checkbox filters, add the following code to the **Page_Load** event:

To write the code in Visual Basic:

```
Visual Basic
' Set ShowFilter to True
 C1GridView1.ShowFilter = True
```

To write the code in C#:

```
C#
// Set ShowFilter to true
 C1GridView1.ShowFilter = true;
```

Client-Side Selection

This topic demonstrates how to enable client side selection and change the background color of selected rows, columns and cells of a C1GridView.

In the Designer

Complete the following steps to create a bound grid:

1. Drag and drop a C1GridView control to your Web Form.
2. Add data to the C1GridView. For more information, see Binding the Grid to a Data Source.
3. Click **OK**.

In Source View

To change the background color of a single selected row, add ClientSelectionMode= "SingleRow" to the *<c1:C1GridView>* tag so that it appears similar to the following:

```
<cc1:C1GridView ID="C1GridView1" runat="server" AutogenerateColumns="False" DataKeyNames="EmployeeID" DataSourceID="SqlDataSource1" ClientSelectionMode= "SingleRow">
```

To define the background color, add the following code between the <head> </head> tag as shown below:

```
<style type = "text/css">
.wijmo-wijgrid .ui-state-highlight
{
     background-color: Green!important;
 }
</style>
```

To change background color of multiple rows, add ClientSelectionMode = "MultiRow" to the *<c1:C1GridView>* tag and attach the CSS code as shown above. ClientSelectionMode property can also be set for cells or columns.

What You've Accomplished

When you run the project, the background color of the selected rows are changed. To select multiple rows use the SHIFT or CTRL keys.

EmployeeID	LastName	Title	FirstName	City	Country
1	Davolio	Sales Representative	Nancy	Seattle	USA
2	Fuller	Vice President, Sales	Andrew	Tacoma	USA
3	Leverling	Sales Representative	Janet	Kirkland	USA
4	Peacock	Sales Representative	Margaret	Redmond	USA
5	Buchanan	Sales Manager	Steven	London	UK
6	Suyama	Sales Representative	Michael	London	UK
7	King	Sales Representative	Robert	London	UK
8	Callahan	Inside Sales Coordinator	Laura	Seattle	USA
9	Dodsworth	Sales Representative	Anne	London	UK

Client-Side Tutorials

The following tutorials will walk you through completing more complex applications using the C1GridView control's client-side scripting.

Client-Side Editing Tutorial

In this tutorial, you will create a grid bound to a datasource, enable client-side editing using the AllowClientEditing property, and validate the data provided by the user before it gets updated back to the server.

Step 1 of 4: Binding Data to the control

In this step you begin by creating a new project, adding the C1GridView control to it and then binding it to a datasource.
Note that in this example, you'll be using the Northwind database, C1Nwind.mdb, installed by default in the **ComponentOne Samples\Common** folder installed in your **Documents** folder.

In the Designer

1. From the Visual Studio **File** menu select **New** | **Project**. The **New Project** dialog box appears.
2. In the **New Project** dialog box expand the language node in the left pane and select Web.
3. In the right pane, choose **ASP.NET Empty Web Application**, enter a **Name** for your application, and select **OK**. A new application is created.
4. In the Solution Explorer, right-click the project name and choose **Add Reference**.
5. In the **Add Reference** dialog box, locate and select the **C1.Web.Wijmo.Controls** and **C1.Web.Wijmo.Controls.Design** assemblies and click **OK**. The references are added.
6. Right-click the project in the Solution Explorer and from the context menu choose **Add** | **New Item**.
7. In the **Add New Item** dialog box choose **Web Form** from the list of templates, name the item Default.aspx, and click **Add**. The new web form opens.
8. In the Solution Explorer, right click the project name and choose **Add | New Folder**. Name the new folder **App_Data**.
9. Navigate to the Visual Studio Toolbox and double-click the C1GridView icon to add the control to your project.
10. In the Design view, select the C1GridView control.
11. In the Solution Explorer window, right-click the **App_Data** folder and select **Add Existing Item** from the context menu.
12. In the **Add Existing Item** dialog box, navigate to where the Northwind database is located and select **C1Nwind.mdb**. By default the Northwind database is in the samples directory.
13. Click **Add** to close the dialog box and add the file to your project.

In Source View
Switch to source view and complete the following steps:

1. Click the C1GridView control to select it and navigate to the Properties window to set C1GridView control properties or set the following text to the *<cc1:C1GridView>* tag:
   a. Set the HighlightCurrentCell property to True.
   b. Set the AllowKeyboardNavigation property to True.
   c. Set the AutogenerateColumns property to False.
   d. Set the DataKeyNames property to **OrderID**.

   It will appear similar to the following:

```
<cc1:C1GridView ID="C1GridView1" runat="server" DataKeyNames="OrderID" HighlightCurrentCell="true" AllowKeyboardNavigation="true"
        AutogenerateColumns="false">
```

2. Add columns to bind data fields using the following code inside the <cc1:C1GridView> ... </cc1:C1GridView> tags.

```
<Columns>
    <cc1:C1BoundField DataField="OrderID" HeaderText="OrderID"></cc1:C1BoundField>
    <cc1:C1BoundField DataField="ShipName" HeaderText="ShipName"></cc1:C1BoundField>
    <cc1:C1BoundField DataField="ShipCity" HeaderText="ShipCity"></cc1:C1BoundField>
    <cc1:C1BoundField DataField="ShippedDate" HeaderText="ShippedDate"></cc1:C1BoundField>
    </Columns>
```

3. Add the following code after </cc1:C1GridView> to set the query:

```
<asp:AccessDataSource ID="AccessDataSource1" runat="server" DataFile="~/App_Data/C1Nwind.mdb" SelectCommand="SELECT TOP 10 [OrderID], [ShipName],
    [ShipCity], [ShippedDate] FROM ORDERS WHERE [ShippedDate] IS NOT NULL"></asp:AccessDataSource>
```

In Code

Open the code behind and add the following code:

1. Add the following code to the **Page_Load** event to refresh the view in the grid:

   To write code in Visual Basic:

```
Visual Basic
If Not IsPostBack Then
  UpdateView()
End If
```

   To write code in C#:

```
C#
if (!IsPostBack)
  {
    UpdateView();
  }
```

2. Add the following at the top of your page to add namespaces in your code:

   To write code in Visual Basic:

```
Visual Basic
Imports System.Collections
```

```
Imports System.Data
Imports System.Data.OleDb
```

To write code in C#:

```
C#
using System.Collections;
using System.Data;
using System.Data.OleDb;
```

3. Add the following code to bind the data source to the grid after the **Page_Load** event:

To write code in Visual Basic:

```
Visual Basic
Private Function GetDataSet() As DataTable
 Dim orders As DataTable = TryCast(Page.Session("ClinetOrders"), DataTable)
  If orders Is Nothing Then_
    Using connection As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data_
   Source=|DataDirectory|\C1Nwind.mdb;Persist Security Info=True")
      Using adapter As New OleDbDataAdapter("SELECT TOP 10 [OrderID], [ShipName],_
       [ShipCity], [ShippedDate] FROM ORDERS WHERE [ShippedDate] IS NOT NULL", connection)
         orders = New DataTable("Orders")
         adapter.Fill(orders)
         orders.PrimaryKey = New DataColumn() {orders.Columns("OrderID")}
         Page.Session("ClinetOrders") = orders
      End Using
    End Using
  End If
 Return orders
End Function
Private Sub UpdateView()
 ' Bind the data
 C1GridView1.DataSource = GetDataSet()
 C1GridView1.DataBind()
End Sub
```

To write the code in C#:

```
C#
private DataTable GetDataSet()
    {
        DataTable orders = Page.Session["ClinetOrders"] as DataTable;
        if (orders == null)
        {
            using (OleDbConnection connection = new OleDbConnection(@"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=|DataDirectory|\C1Nwind.mdb;Persist Security Info=True"))
            {
                using (OleDbDataAdapter adapter = new OleDbDataAdapter("SELECT TOP 10 [OrderID], [ShipName], [ShipCity], [ShippedDate] FROM ORDERS WHERE [ShippedDate] IS NOT NULL",
connection))
                {
                    orders = new DataTable("Orders");
                    adapter.Fill(orders);
                    orders.PrimaryKey = new DataColumn[] { orders.Columns["OrderID"] };
                    Page.Session["ClinetOrders"] = orders;
                }
            }
        }
        return orders;
    }
    private void UpdateView()
    {
     //Bind the data
        C1GridView1.DataSource = GetDataSet();
       C1GridView1.DataBind();
    }
```

What You've Accomplished

Run the project and observe that you now have a fully-functional grid application Orders table of the database.

OrderID	ShipName	ShipCity	ShippedDate
10,330	LILA-Supermercado	Barquisimeto	11/28/1994
10,331	Bon app'	Marseille	11/21/1994
10,332	Mère Paillarde	Montréal	11/21/1994
10,333	Wartian Herkku	Oulu	11/25/1994
10,334	Victuailles en stock	Lyon	11/28/1994
10,335	Hungry Owl All-Night Grocers	Cork	11/24/1994
10,336	Princesa Isabel Vinhos	Lisboa	11/25/1994
10,337	Frankenversand	München	11/29/1994
10,338	Old World Delicatessen	Anchorage	11/29/1994
10,339	Mère Paillarde	Montréal	12/5/1994

In the next step of this tutorial you'll customize the grid's functionality by setting the client side edit feature and explore the grid's run-time interactions.

Step 2 of 4: Enabling client-side editing

In the previous step of the tutorial you created a simple grid application and bound the grid to a data source. In this step you customize the grid application further by enabling client-side editng.

Complete the following steps to continue:

In Source View

Click the C1GridView control to select it and navigate to the Properties window to set C1GridView control AllowClientEditing property to True or set the following in the <cc1:C1GridView> tag:

```
AllowClientEditing="true"
```

It will appear similar to the following:

```
<cc1:C1GridView ID="C1GridView1" runat="server"
AllowClientEditing="true" DataKeyNames="OrderID" HighlightCurrentCell="true" AllowKeyboardNavigation="true"
AutogenerateColumns="false">
```

What You've Accomplished

Run the project and see that you can now edit the cell on double click.

OrderID	ShipName	ShipCity	ShippedDate
10,330	LILA-Supermercado	Barquisimeto	11/28/1994
10,331	Bon app'	Marseille	11/21/1994
10,332	Mère Paillarde	Montréal	11/21/1994
10,333	Changed Name ×	Oulu	11/25/1994
10,334	Victuailles en stock	Lyon	11/28/1994
10,335	Hungry Owl All-Night Grocers	Cork	11/24/1994
10,336	Princesa Isabel Vinhos	Lisboa	11/25/1994
10,337	Frankenversand	München	11/29/1994
10,338	Old World Delicatessen	Anchorage	11/29/1994
10,339	Mère Paillarde	Montréal	12/5/1994

In the next step of this tutorial add data validation to the grid values.

Step 3 of 4: Data validation before updating

In the previous step of the tutorial you enabled client-side editing. In this step you customize the grid application further by validating the data provided by the user before it is updated back to the server.

Complete the following steps to continue:

In Source View

1. Add the validation condition in the beforeCellUpdate function within the <head>...</head> tags. In this example, if the value in the Order ID field is less than 10000, a message appears :

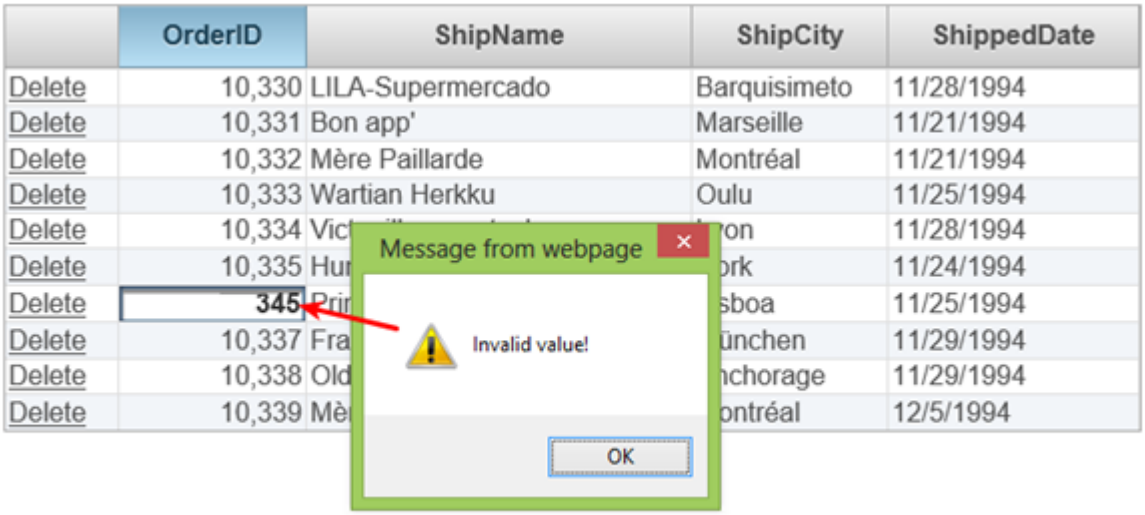
```
<script type="text/javascript">
                function beforeCellUpdate(e, args) {
                    if (args.cell.column().dataField === "OrderID") {
                        var editor = $(args.cell.tableCell()).find("input"),
                            value = parseInt(editor.val());

                        if (value < 10000) {
                            editor.addClass("invalid-value");
                            alert("Invalid value!");
                            return false;
                        }
                    }
                }
            </script>
```

2. Click the C1GridView control to select it and navigate to the Properties window to

set OnClientBeforeCellUpdate property to *beforeCellUpdate*, or add *OnClientBeforeCellUpdate="beforeCellUpdate"* in *<cc1:C1GridView>* tag.

What You've Accomplished

Run your project and observe that when you try to edit the data from the Orders table, the changed value is validated against the condition we set above. If the input provided by the user in any OrderID field less than 10000, then an "Invalid value" message appears.



In the next step of this tutorial, update the validated data back to the server.

Step 4 of 4: Updating Data back to the Server

In the previous step of the tutorial you validated the data entered by the user against a condition. In this step you update the new data back to the server.

Complete the following steps to continue:

In Source View

1. Add *<CallbackSettings Action="Editing" />* after the *<Columns>...</Columns>* tags.
2. Click the C1GridView control to select it and navigate to the Properties window to set C1GridView control **AutoGenerateDeleteButton** property to True or set the following in the <cc1:C1GridView> tag:
   `AutoGenerateDeleteButton="true"`

   It appears similar to the following:

```
<cc1:C1GridView ID="C1GridView1" runat="server"
AutoGenerateDeleteButton="true" AllowClientEditing="true" DataKeyNames="OrderID" HighlightCurrentCell="true" AllowKeyboardNavigation="true" AutogenerateColumns="false">
```

In Code

1. Add the following code to the **C1GridView1_RowEditing** event:

   To write the code in Visual Basic:

```
Visual Basic
Private Sub C1GridView1_RowEditing(sender As Object, e As C1.Web.Wijmo.Controls.C1GridView.C1GridViewEditEventArgs)
  C1GridView1.EditIndex = e.NewEditIndex
  UpdateView()
End Sub
```

   To write the code in C#:

```
C#
void C1GridView1_RowEditing(object sender, C1.Web.Wijmo.Controls.C1GridView.C1GridViewEditEventArgs e)
{
C1GridView1.EditIndex = e.NewEditIndex;
    UpdateView();
}
```

2. To handle the **C1GridView1_RowEditing** event, add the following code to the **Page_Init** event:

   To write the code in Visual Basic:

```
Visual Basic
Protected Sub Page_Init(sender As Object, e As EventArgs)
    AddHandler C1GridView1.RowEditing, AddressOf C1GridView1_RowEditing
End Sub
```

   To write the code in C#:

```
C#
protected void Page_Init(object sender, EventArgs e)
    {
        C1GridView1.RowEditing += C1GridView1_RowEditing;
    }
```

3. Update the dataset to handle the changes made by the user. Add the following code in the **C1GridView1_RowUpdating** event:

   To write the code in Visual Basic:

```
Visual Basic
Dim orders As DataTable = GetDataSet()

Dim row As DataRow = orders.Rows.Find(C1GridView1.DataKeys(e.RowIndex).Value)
 If row IsNot Nothing Then
    For Each entry As DictionaryEntry In e.NewValues
      row(DirectCast(entry.Key, String)) = entry.Value
    Next
 Else
    Throw New RowNotInTableException()
 End If
```

   To write the code in C#:

```
C#
DataTable orders = GetDataSet();
 DataRow row = orders.Rows.Find(C1GridView1.DataKeys[e.RowIndex].Value);
      if (row != null)
        {
            foreach (DictionaryEntry entry in e.NewValues)
            {
              row[(string)entry.Key] = entry.Value;
            }
        }
        else
        {
            throw new RowNotInTableException();
        }
```

4. Accept changes in the dataset so that the edited values are sent back to the server. Add the following code in the **C1GridView1_EndRowUpdated** event:

To write the code in Visual Basic:

```
Visual Basic
GetDataSet().AcceptChanges()
UpdateView()
```

To write the code in C#:

```
C#
GetDataSet().AcceptChanges();
UpdateView();
```

5. Delete a row when the corresponding **Delete** button is clicked. Add the following code in the **C1GridView1_RowDeleting** event:

To write the code in Visual Basic:

```
Visual Basic
Dim orders As DataTable = GetDataSet()

Dim row As DataRow = orders.Rows.Find(C1GridView1.DataKeys(e.RowIndex).Value)

  If row IsNot Nothing Then
    orders.Rows.Remove(row)
    orders.AcceptChanges()
    UpdateView()
  Else
    Throw New RowNotInTableException()
  End If
```

To write the code in C#:

```
C#
DataTable orders = GetDataSet();
 DataRow row = orders.Rows.Find(C1GridView1.DataKeys[e.RowIndex].Value);
        if (row != null)
        {
          orders.Rows.Remove(row);
          orders.AcceptChanges();
          UpdateView();
        }
        else
        {
          throw new RowNotInTableException();
        }
```

6. Handle the **C1GridView1_RowDeleting** event, add the following code to the **UpdateView()** method:

To write the code in Visual Basic:

```
Visual Basic
AddHandler C1GridView1.RowDeleting, AddressOf C1GridView1_RowDeleting
```

To write the code in C#:

```
C#
C1GridView1.RowDeleting += new C1.Web.Wijmo.Controls.C1GridView.C1GridViewDeleteEventHandler(C1GridView1_RowDeleting);
```

In Source View

Click the C1GridView control to select it and navigate to the properties window to handle C1GridView control events or set the following text to the **<cc1:C1GridView>** tag:

1. Set the EndRowUpdated event to **C1GridView1_EndRowUpdated**.
2. Set the RowDeleting event to **C1GridView1_RowDeleting**.
3. Set the RowUpdating event to **C1GridView1_RowUpdating**.

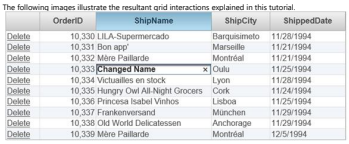
The tag appears similar to the following:

```
<cc1:C1GridView ID="C1GridView1" runat="server"
AutoGenerateDeleteButton="true" AllowClientEditing="true" DataKeyNames="OrderID" HighlightCurrentCell="true" OnRowUpdating="C1GridView1_RowUpdating" OnEndRowUpdated="C1GridView1_EndRowUpdated" OnRowDeleting="C1GridView1_RowDeleting" AllowKeyboardNavigation="true" AutogenerateColumns="false">
```

What You've Accomplished

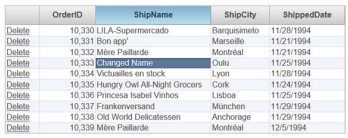
Run the project and see that the edited data value is updated on the server. You can also delete a row and update the grid with the changes.

The following images illustrate the resultant grid interactions explained in this tutorial.



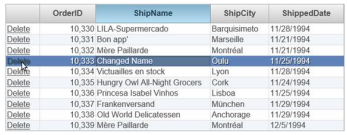
	OrderID	ShipName	ShipCity	ShippedDate
Delete	10,330	LILA-Supermercado	Barquisimeto	11/28/1994
Delete	10,331	Bon app'	Marseille	11/21/1994
Delete	10,332	Mère Paillarde	Montréal	11/21/1994
Delete	10,333	Changed Name	Oulu	11/25/1994
Delete	10,334	Victuailles en stock	Lyon	11/28/1994
Delete	10,335	Hungry Owl All-Night Grocers	Cork	11/24/1994
Delete	10,336	Princesa Isabel Vinhos	Lisboa	11/25/1994
Delete	10,337	Frankenversand	München	11/29/1994
Delete	10,338	Old World Delicatessen	Anchorage	11/29/1994
Delete	10,339	Mère Paillarde	Montréal	12/5/1994

Fig (i) - Editing column value in the grid



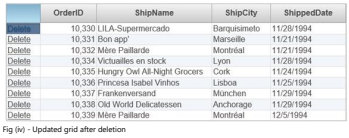
	OrderID	ShipName	ShipCity	ShippedDate
Delete	10,330	LILA-Supermercado	Barquisimeto	11/28/1994
Delete	10,331	Bon app'	Marseille	11/21/1994
Delete	10,332	Mère Paillarde	Montréal	11/21/1994
Delete	10,333	Changed Name	Oulu	11/25/1994
Delete	10,334	Victuailles en stock	Lyon	11/28/1994
Delete	10,335	Hungry Owl All-Night Grocers	Cork	11/24/1994
Delete	10,336	Princesa Isabel Vinhos	Lisboa	11/25/1994
Delete	10,337	Frankenversand	München	11/29/1994
Delete	10,338	Old World Delicatessen	Anchorage	11/29/1994
Delete	10,339	Mère Paillarde	Montréal	12/5/1994

Fig (ii) - Changed value updated in the grid



	OrderID	ShipName	ShipCity	ShippedDate
Delete	10,330	LILA-Supermercado	Barquisimeto	11/28/1994
Delete	10,331	Bon app'	Marseille	11/21/1994
Delete	10,332	Mère Paillarde	Montréal	11/21/1994
Delete	10,333	Changed Name	Oulu	11/25/1994
Delete	10,334	Victuailles en stock	Lyon	11/28/1994
Delete	10,335	Hungry Owl All-Night Grocers	Cork	11/24/1994
Delete	10,336	Princesa Isabel Vinhos	Lisboa	11/25/1994
Delete	10,337	Frankenversand	München	11/29/1994
Delete	10,338	Old World Delicatessen	Anchorage	11/29/1994
Delete	10,339	Mère Paillarde	Montréal	12/5/1994

Fig (iii) - Deleting a row in the grid



	OrderID	ShipName	ShipCity	ShippedDate
Delete	10,330	LILA-Supermercado	Barquisimeto	11/28/1994
Delete	10,331	Bon app'	Marseille	11/21/1994
Delete	10,332	Mère Paillarde	Montréal	11/21/1994
Delete	10,334	Victuailles en stock	Lyon	11/28/1994
Delete	10,335	Hungry Owl All-Night Grocers	Cork	11/24/1994
Delete	10,336	Princesa Isabel Vinhos	Lisboa	11/25/1994
Delete	10,337	Frankenversand	München	11/29/1994
Delete	10,338	Old World Delicatessen	Anchorage	11/29/1994
Delete	10,339	Mère Paillarde	Montréal	12/5/1994

Fig (iv) - Updated grid after deletion

Updating Database from Client Side

C1GridView enables editing the cells of the grid at client side, without defining Template columns, by setting the AllowClientEditing to true.

> **Note:** Oledb and Sql datasource can be used for binding.

In the Designer

Right-click the C1GridView control and select Show Smart Tag from the context menu to bind it to the *C1Nwind.mdb* database, which is located by default in the samples directory. For detailed steps go to Step 1 of 3: Binding C1GridView to a DataSource.

In Source View

1. To define the DataKeyNames and the columns, and to set the CallbackSettings for editing in C1GridView, modify the `<cc1:C1GridView ></cc1:C1GridView >` tag as shown below:

```
<cc1:C1GridView ID="C1GridView1" runat="server"
OnRowUpdating="C1GridView1_RowUpdating"
AutogenerateColumns="false" DataKeyNames="CustomerID"
ClientSelectionMode="SingleRow"
AllowClientEditing="true" ShowFilter="true" OnFiltering="C1GridView1_Filtering"
OnEndRowUpdated="C1GridView1_EndRowUpdated">
    <CallbackSettings Action="Editing, Filtering" />
    <Columns>
        <cc1:C1BoundField DataField="CustomerID" HeaderText="CustomerID"
SortExpression="CustomerID">
        </cc1:C1BoundField>
        <cc1:C1BoundField DataField="CompanyName" HeaderText="Company Name"
SortExpression="CompanyName">
        </cc1:C1BoundField>
        <cc1:C1BoundField DataField="ContactName" HeaderText="Contact Name"
SortExpression="ContactName">
        </cc1:C1BoundField>
        <cc1:C1BoundField DataField="City" HeaderText="City"
SortExpression="City">
        </cc1:C1BoundField>
        <cc1:C1BoundField DataField="Country" HeaderText="Country"
SortExpression="Country">
        </cc1:C1BoundField>
    </Columns>
</cc1:C1GridView>
```

2. To add a button named "Update", add the following code:

```
<asp:Button ID="btn1" runat="server" Text="Update C1GridView"
            OnClientClick="btn_ClientClick(); return false;" />
```

3. Use the following jQuery function to call the update() method:

```
function btn_ClientClick(sender, args) {
            var grid = $("#C1GridView1");
            grid.c1gridview("endEdit");
            grid.c1gridview("update");
        }
```

In Code

1. To set the Update command and to call the **Update()** method, add the following code :

   To write the code in Visual Basic

Visual Basic

```
Public Function GetDataTable() As DataTable
    Dim dt As DataTable = TryCast(Page.Session("Customers"), DataTable)
    Dim con As New OleDbConnection("provider=Microsoft.Jet.Oledb.4.0; Data Source=" +
Server.MapPath("~/App_Data/C1NWind.mdb"))
    Dim da As New OleDbDataAdapter()
    da.SelectCommand = New OleDbCommand("SELECT * FROM [Customers] Order By [CustomerID]", con)
    da.UpdateCommand = New OleDbCommand("Update [Customers] set [CompanyName]=?,
[ContactName]=?, [City]=?, [Country]=? where CustomerID = ?", con)
    da.UpdateCommand.Parameters.Add("@CompanyName", OleDbType.VarChar, 50, "CompanyName")
```

```
        da.UpdateCommand.Parameters.Add("@ContactName", OleDbType.VarChar, 50, "ContactName")
        da.UpdateCommand.Parameters.Add("@City", OleDbType.VarChar, 50, "City")
        da.UpdateCommand.Parameters.Add("@Country", OleDbType.VarChar, 50, "Country")
        da.UpdateCommand.Parameters.Add("@CustomerID", OleDbType.VarChar, 50, "CustomerID")
        If dt Is Nothing Then
            dt = New DataTable()
            da.Fill(dt)
            dt.PrimaryKey = New DataColumn() {dt.Columns("CustomerID")}
            Page.Session("Customers") = dt
        End If
        da.Update(dt)
        Return dt
    End Function
```

To write the code in C#

C#

```
public DataTable GetDataTable()
    {
        DataTable dt = Page.Session["Customers"] as DataTable;
        OleDbConnection con = new OleDbConnection("provider=Microsoft.Jet.Oledb.4.0; Data Source=" +
Server.MapPath("~/App_Data/C1NWind.mdb"));
        OleDbDataAdapter da = new OleDbDataAdapter();
        da.SelectCommand = new OleDbCommand("SELECT * FROM [Customers] Order By [CustomerID]", con);

        da.UpdateCommand = new OleDbCommand("Update [Customers] set [CompanyName]=?,
[ContactName]=?, [City]=?, [Country]=? where CustomerID = ?", con);
        da.UpdateCommand.Parameters.Add("@CompanyName", OleDbType.VarChar, 50, "CompanyName");
        da.UpdateCommand.Parameters.Add("@ContactName", OleDbType.VarChar, 50, "ContactName");
        da.UpdateCommand.Parameters.Add("@City", OleDbType.VarChar, 50, "City");
        da.UpdateCommand.Parameters.Add("@Country", OleDbType.VarChar, 50, "Country");
        da.UpdateCommand.Parameters.Add("@CustomerID", OleDbType.VarChar, 50, "CustomerID");
        if (dt == null)
        {
            dt = new DataTable();
            da.Fill(dt);
            dt.PrimaryKey = new DataColumn[] { dt.Columns["CustomerID"] };
            Page.Session["Customers"] = dt;
        }
        da.Update(dt);
        return dt;
    }
```

2. To update the row with the modified data, add the following code to handle the RowUpdating event:

To write the code in Visual Basic

Visual Basic

```
Protected Sub C1GridView1_RowUpdating(sender As Object, e As
C1.Web.Wijmo.Controls.C1GridView.C1GridViewUpdateEventArgs) Handles C1GridView1.RowUpdating
        Dim customers As DataTable = GetDataTable()
        Dim row As DataRow = customers.Rows.Find(C1GridView1.DataKeys(e.RowIndex).Value)
```

```
    If row IsNot Nothing Then
      For Each entry As DictionaryEntry In e.NewValues
        row(DirectCast(entry.Key, String)) = entry.Value
      Next
    Else
      Throw New RowNotInTableException()
    End If
  End Sub
```

To write the code in C#

C#

```
 protected void C1GridView1_RowUpdating(object sender,
C1.Web.Wijmo.Controls.C1GridView.C1GridViewUpdateEventArgs e)
  {
    DataTable customers = GetDataTable();
    DataRow row = customers.Rows.Find(C1GridView1.DataKeys[e.RowIndex].Value);
    if (row != null)
    {
      foreach (DictionaryEntry entry in e.NewValues)
      {
        row[(string)entry.Key] = entry.Value;
      }
    }
    else
    {
      throw new RowNotInTableException();
    }
    Page.Session["Customers"] = customers;
  }
```

3. To rebind the grid, add the following code to handle the EndRowUpdated event:

To write the code in Visual Basic

Visual Basic

```
 Protected Sub C1GridView1_EndRowUpdated(sender As Object, e As
C1.Web.Wijmo.Controls.C1GridView.C1GridViewEndRowUpdatedEventArgs) Handles
C1GridView1.EndRowUpdated
    C1GridView1.DataSource = GetDataTable()
    C1GridView1.DataBind()
  End Sub
```

To write the code in C#

C#

```
 protected void C1GridView1_EndRowUpdated(object sender,
C1.Web.Wijmo.Controls.C1GridView.C1GridViewEndRowUpdatedEventArgs e)
  {
    C1GridView1.DataSource = GetDataTable();
    C1GridView1.DataBind();
```

```
}
```

You can change the selection to some other row and edit the values to save the updated values. Double-click the cells to make them editable.

Tip: You cannot edit a grid containing a single row, as there is no other row which can be clicked to change the selection and make the changes. The workaround for the same is to call the client side update() method of C1GridView. For details please visit our online blog.

Handling client side Key events

This topic demonstrates how to enable handling key events, in C1GridView control, at client side. Since C1GridView does not have any keyboard events of its own, you need to handle its OnClientBeforeCellEdit event and define the keyup event for each cell. Thus, if you make a change in any cell then this event will fire, and you can check the content entered by the end user.

Use the following code in OnClientBeforeCellEdit event in the source view:

```
function BeforeCellEdit(e, args) {
                var cell = args.cell;
                var grid = $("#<%=C1GridView1.ClientID %>");
                cell.container().keyup(function (event) {
                    var count = 0;
                    var key = event.which;
                    if (key >= 65 && key <= 90) {
                        count = 1;
                    }
                    else if (key >= 48 && key <= 57) {
                        count = 1;
                    }                    if (count == 1) {
                        $(grid).c1gridview("endEdit");
                        var row = cell.rowIndex();
                        var col = cell.cellIndex();
                        $(grid).c1gridview("currentCell", col + 1, row);
                    }
                });
            }
```

In the above code, you retrieve the key entered by the user with the help of the event, and if it is a letter or a number then the focus moves to the next cell. Similarly, you can check the user input and can take corresponding action on the basis of same.

Samples

Please be advised that this ComponentOne software tool is accompanied by various sample projects and/or demos which may make use of other development tools included with the ComponentOne Studios. Samples can be accessed from the **ComponentOne ControlExplorer**. The **ControlExplorer** sample includes a section that details some of the functionality available in the **C1GridView** control.

C# Samples

The following pages within the **ControlExplorer** sample installed with ASP.NET Web Forms Edition detail the C1GridView control's functionality:

for ASP.NET Web Forms

Sample	Description
Overview	The **C1GridView** displays the values of a data source in a table where each column represents a field and each row represents a record. This sample demonstrates a simple sortable grid.
CustomEditing	The **C1GridView** supports server editing. This sample shows how to use custom data binding to bind the **C1GridView** to a dataset stored in the session object, how to use template fields to implement custom editors and how to use **UpdateBindings** collections of the template fields to automatically update the dataset with the Update method when custom data binding is used.
ClientSideSelection	**C1GridView** supports several modes of client-side cell selection. This sample shows these modes and demonstrates how to use the client-side object model to change selection mode without sending requests to the server.
Filtering	The **C1GridView** supports an intuitive user interface for filtering the rows in the grid. This sample demonstrates filtering.
Columns	This sample shows how to use bands to organize column headers into a hierarchical structure. It also shows you how to sort columns and use a drag-and-drop operation to reorder columns.
Paging	Paging is allowed if the **AllowPaging** property is set to True. This sample demonstrates paging.
GroupArea	The **C1GridView** supports data grouping against one or more columns. The column will be grouped if the **GroupInfo.Position** property is set to a value other than **None**. This sample demonstrates grouping.
Aggregates	The **C1GridView** supports aggregates calculation against one or more grouped columns. It is possible to calculate aggregate values for each group of data rows. An aggregate can be calculate for any column with compatible data type if the column is preceded by a grouped column. **C1GridView** supports aggregate functions like **Sum**, **Count**, **Min**, **Max** and other.
Scrolling	The **C1GridView** supports both horizontal and vertical scrolling. Scrolling is allowed if the **ScrollingSettings.Mode** property is set to value other than **None**. When scrolling is used, **C1GridView**'s header is automatically fixed.
CurrentCell	The **C1GridView** supports operations on the current cell. The current cell is a data cell having focus. **C1GridView** highlights the current cell position, tracks it changes, and allows the retrieval of data value.

GroupRowsCustomStyling	In this sample group header cells are styled using the .wijmo-wijgrid .wijmo-wijgrid-groupheaderrow CSS sequence.
ClientSideEditing	The **C1GridView** supports client editing. This sample shows how to update a dataset stored in the session object when client editing is used and how to provide client-side custom editors.
Merging	The **C1GridView** supports row merging. Identical values of the particular column can be spanned within a single cell. Row merging is allowed if the **RowMerge** property of the column is set to value other than **None**.
Sorting	This sample demonstrates sorting. Sorting is allowed if the **AllowSorting** property is set to **True**. End-user sorting is allowed if the **SortExpression** property of the column is set to non-empty value.

Client-side Reference

As part of the amazing ComponentOne Web stack, the Wijmo jQuery UI widgets are optimized for client-side Web development and utilize the power of jQuery for superior performance and ease of use.

The ComponentOne Wijmo website at http://wijmo.com/widgets/ provides everything you need to know about Wijmo widgets, including demos and samples, documentation, theming examples, support and more.

The client-side documentation provides an overview, sample markup, options, events, and methods for each widget. To get started with client-side Web development for **GridView for ASP.NET Web Forms**, click one of the external links to view our Wijmo wiki documentation. Note that the **Overview** topic for each of the widgets applies mainly to the widget, not to the server-side ASP.NET Web Forms control.

GridView

- wijgrid documentation
- wijgrid API