DataGrid Control Reference

Display and edit tabular data.

Reference **Controls** **Interaction** DataGrid

The DataGrid, in the context of the components, refers to the graphical element for visualization and handling structured data in your solutions displays. It allows users to view, edit, sort, filter, and manage data within a grid structure. Key functionalities include data binding, editing capabilities, sorting, filtering, and customizable columns. Applications include industrial monitoring systems, where real-time data needs visualization and interaction. Before usage, configure data sources, binding modes, and column properties. Usage involves setting up DataGrid in the display, defining columns, and enabling desired features like editing, sorting, and filtering through the configuration menu.

Requirements
Configuration
- General
- Columns

Customizations
- Columns with 2 Decimal Places
- Set Selected Row as First (WPF Only displays)
- Select a Row by Content and Scroll It Into View — portable (WPF + HTML5)
- Frozen Columns
- Build Columns Dynamically in CodeBehind
- Scrolling Controls for DataGrid
- Customizing window appearance
- Responsive Sizing

Runtime Execution

Requirements

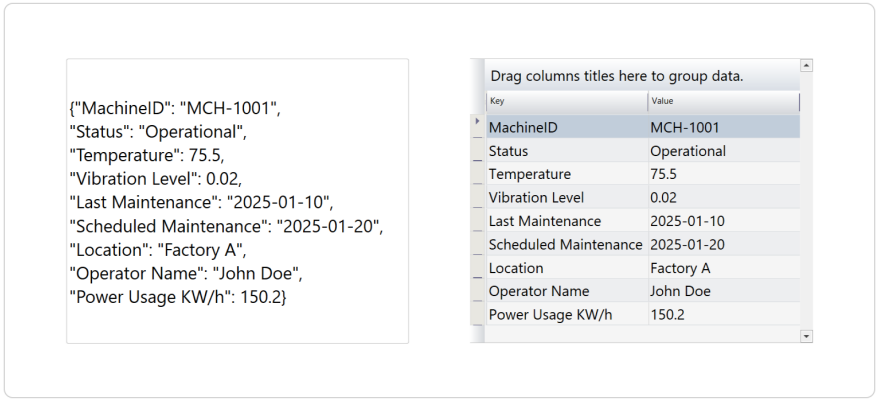
This component is Portable. It runs both on Windows (WPF) and on Web Pages hosted in any platform.

Configuration

Double-click any DataGrid window object in the display to configure it.

General

DataGrid General Settings	
Field	**Description**
Data Source	Defines the dataset table or query from which the data grid will get the data. It includes support for several data types, including JSON.
Control Name	Defines a name for the control so it can be accessed in the code-behind script. See Display CodeBehind.
Selected Values	Captures the contents of the selected row using a string tag or array tag.
Selected Index	Captures the number of the currently selected row, starting with 0, using a tag.
Line Count	Receives the number of lines in the window using a selected tag.



Pause Update	Determines whether the DataGrid updates when the data changes by setting an object.
Grid Settings	Stores the DataGrid configuration using a retentive text tag, maintaining settings after a shutdown or restart.
Filter	Limits the displayed data using an SQL statement.
Binding	Determines the binding mode for the selected component. Options include: **TwoWay:** Updates the data source and UI interchangeably; **OutputOnly:** Allows data output without user input; **InputOnly:** Accepts user input without updating the data source
Theme	Selects the visual style for the DataGrid.
Allow Edit	Enables editing of the selected values.
Allow Insert	Allows users to add rows.
Allow Delete	Allows row deletion.
Column Titles	Displays column titles.
Allow Sort	Allows users to sort data.
Transpose	Enables transposing of columns to rows.
Auto Columns	This option automatically includes all table or query columns in the window. Select this option to have the system include all columns without manual configuration.
OneClick Edit	Allows editing of cells on the first click.
Multiple Selection	Enables selecting multiple rows or cells simultaneously.
Header	Provides space for dragging column titles to group data.

Understanding SelectedValues Behavior in DataGrid

The SelectedValues property in the DataGrid assigns values differently based on the DataType of the configured parameter:

UserType (IsTargetUserType = true): Matches column names in the DataTable and assigns values accordingly.
Array (IsArray = true): Receives DataRow.ItemArray, assigning values based on index order.
Default case: Takes the value from the first column of the DataTable.

Understanding this behavior helps ensure correct data handling in your configuration.

Columns

DataGrid Configuration	
Field	**Description**
Field	Specifies the name of the column in the DataGrid.
Title	Sets the text displayed as the header for the column.
Width	Defines the initial width of the column.
Sort	Determines the initial sort order for the column. The options are: **None**: Sets a column to disable sorting. Users cannot sort the data in this column, maintaining the original order of the data. **Ascending**: Sets a column to sort data in ascending order. When applied, the data in this column will be arranged from the smallest to the largest value, or from A to Z for text entries. **Descending**: Sets a column to sort data in descending order. When applied, the data in this column will be arranged from the largest to the smallest value, or from Z to A for text entries.
Visible	Shows or hides the column in the DataGrid.
Editable	Allows or disallows editing the column's data.

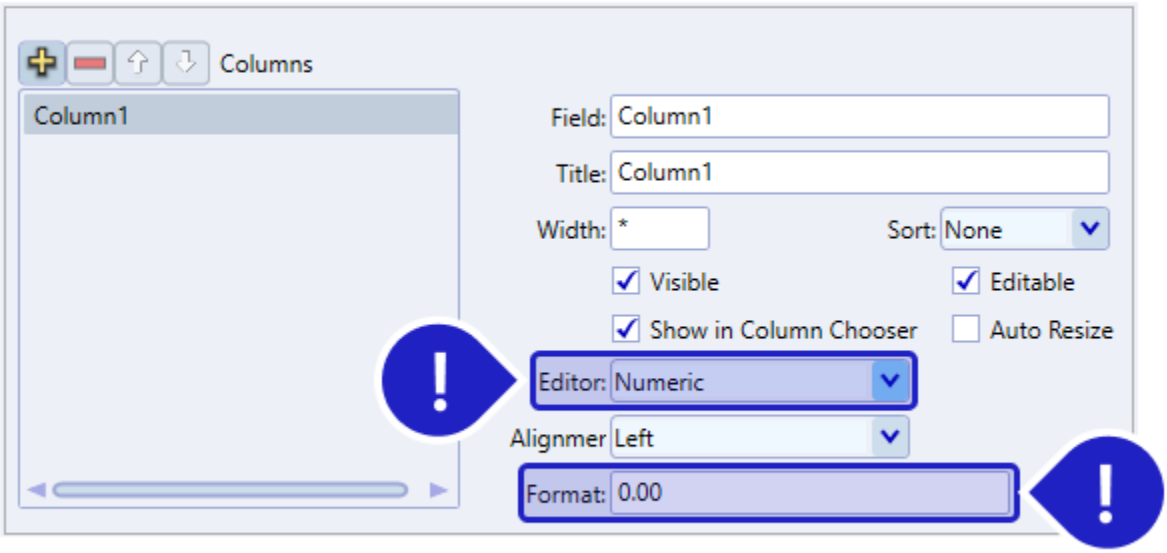
Show in Column Chooser	Includes the column in the column chooser, allowing users to select which columns to display.
Auto Resize	Automatically adjusts the column width based on content.
Editor	Sets the data type of the column (e.g., text, numeric). The options are: **Text**: Sets a column to allow users to input and edit plain text. Ideal for columns that require alphanumeric data, such as names or descriptions. **Numeric**: Sets a column to accept only numerical input. Useful for columns that store integer or decimal values, such as quantities or prices. **CheckBox**: Sets a column to display a checkbox for each cell. Users can toggle the checkbox to represent binary data, such as active/inactive status or true/false values. **MaskedText**: Sets a column to enforce a specific input format. Used for fields like phone numbers or social security numbers, ensuring the data adheres to a required pattern. **DateTime**: Sets a column to allow dates and times. Useful for columns that record scheduling information, such as appointment dates or timestamps. **Interval**: Sets a column to accept time spans. Users can input periods or durations, making it suitable for columns that track intervals, like project durations or time logs. **ComboBox**: Sets a column to present a dropdown list of predefined options. Useful for columns where users need to select from multiple values, such as categories or statuses. **Color**: Sets a column to allow users to select colors. Particularly useful for columns that require color coding or selection, such as priority indicators or design elements.
Alignment	Specifies the horizontal alignment of the column content (e.g., left, right, center). The options are: **Left**: Aligns the data to the left side in the column. **Right**: Aligns the data to the right side in the column. **Center**: Aligns the data to the center in the column. **Stretch**: Stretches the column horizontally.
Format	Defines the format for displaying the column's data (e.g., number format, date format).

Customizations

Columns with 2 Decimal Places

There are cases where, even after setting the tag format to display in the grid and adjusting the property window to show only two decimals, it does not work. The grid may still display more decimals than specified in the property window.

To resolve this issue, set the column to numeric and enter ‘0.00’ in the format field within the DataGrid settings menu. This will limit the number of decimal places displayed in the grid.



Set Selected Row as First (WPF Only displays)

```
TDataGrid grid = CurrentDisplay.GetDataGrid("Grid");

Xceed.Wpf.DataGrid.DataGridControl dg = grid.TableGridControl.DataGridObj;

ScrollViewer myScrollViewer = (ScrollViewer)dg.Template.FindName("PART_ScrollViewer", dg);
if (myScrollViewer == null || dg.Items.Count == 0)
    return;

string columnName = "Message";

int selIndex = -1;
for (int i = 0; i < dg.Items.Count; i++)
{
     DataRowView rowView = dg.Items[i] as DataRowView;
     if (TConvert.ToString(rowView.Row[columnName]).StartsWith(@Tag.txtfind))
     {
          dg.Items.MoveCurrentTo(rowView);
          dg.SelectedItem = rowView;
          selIndex = i;
          break;
     }
}

if (selIndex >= 0)
    myScrollViewer.ScrollToVerticalOffset(selIndex);
```

Select a Row by Content and Scroll It Into View — portable (WPF + HTML5)

```
TDataGrid grid = this.CurrentDisplay.GetDataGrid("Grid");
if (grid == null)
        return;

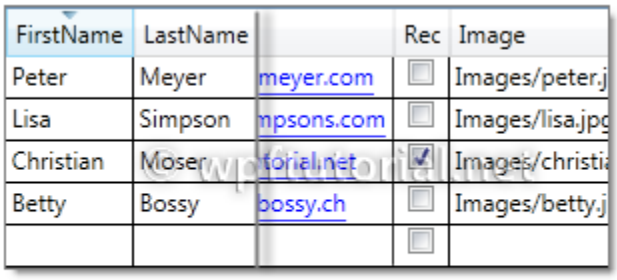
// GetItems() returns the grid's current rows (portable on both clients).
System.Data.DataTable items = grid.GetItems();
if (items == null || !items.Columns.Contains("Message"))
    return;

for (int i = 0; i < items.Rows.Count; i++)
{
    string cell = System.Convert.ToString(items.Rows[i]["Message"]);
    if (cell != null && cell.StartsWith(txtfind))
    {
        grid.SelectRow(i, true);   // select + scroll into view (both clients)
        break;
    }
}
```

Frozen Columns

The data grid supports freezing columns, keeping them visible while you scroll horizontally. This feature is useful for keeping reference columns, like IDs or names, always visible for better orientation.

To freeze a number of columns just set the `FrozenColumnCount` property to the desired number of columns.




```
<DataGrid ItemsSource="{Binding Customers}" FrozenColumnCount="2"  />
```

The code below demonstrates how to freeze the left-most columns in a DataGridWindow element using CodeBehind.

- The `FixedColumnCount` property specifies the number of columns (from left to right) to be fixed.
- The `ShowFixedColumnSplitter` property allows you to select columns using the mouse (by clicking and dragging)."

```
public void DisplayOpening()
{
        TDataGrid gridWindow = CurrentDisplay.GetDataGrid("grid");
        gridWindow.AfterInitilizationEvent += AfterInitilization;
}

public void DisplayIsOpen()
{
        // Add your code here
}

public void DisplayClosing()
{
        TDataGrid gridWindow = CurrentDisplay.GetDataGrid("grid");
        gridWindow.AfterInitilizationEvent -= AfterInitilization;
}

private void AfterInitilization(object sender, EventArgs e)
{
        TDataGrid gridWindow = CurrentDisplay.GetDataGrid("Grid");
        Xceed.Wpf.DataGrid.Views.TableView tableView = gridWindow.TableGridControl.DataGridObj.View as Xceed.
Wpf.DataGrid.Views.TableView;
        tableView.FixedColumnCount = 1;
        tableView.ShowFixedColumnSplitter = true;
}
```

Build Columns Dynamically in CodeBehind

When the same DataGrid is reused to show different tables at runtime, its columns can be built from CodeBehind instead of being fixed in the configuration menu. Disable **Auto Columns**, clear the current columns, then create one `GridColumn` per source column — setting `FieldName` (the bound data column), `Title` (the header text) and `Editable` (whether the column accepts edits) — and call `DoRefresh()` to repaint the grid.

- The grid needs a **Control Name** (here `dg`) so CodeBehind can reach it with `CurrentDisplay.GetDataGrid("dg")`.
- Set `AutoCreateColumn = false` first; otherwise the columns you add are duplicated by the auto-generated ones.
- Per-column `Editable` only takes effect when the grid's **Allow Edit** option is enabled.

```
public async Task Act(object sender, System.Windows.Input.InputEventArgs e)
{
        DataTable myTable = @Tag.Table1;
        TDataGrid dg = this.CurrentDisplay.GetDataGrid("dg");
        dg.AutoCreateColumn = false;
        dg.Columns.Clear();
        foreach (DataColumn c in myTable.Columns)
        {
                GridColumn col = new GridColumn();
                col.FieldName = c.ColumnName;
                col.Title     = c.ColumnName;
                @Info.Trace("Column name: " + col.Title);
                if (c.ColumnName == "Value") // Could be a dynamic text tag
                {
                        col.Editable = true;
                }
                else
                {
                        col.Editable = false;
                }
                dg.Columns.Add(col);
        }
        dg.DoRefresh();
}
```

For cell edits to take effect, the DataGrid must have Binding = TwoWay and Allow Edit enabled in its properties.

This code must run after the DataTable is populated so it can configure the grid with the correct columns. If it doesn't run, the grid stays blank.

Scrolling Controls for DataGrid

The following methods and properties were created to enable scrolling for the DataGrid object:

Methods

```
//Scrolls the content upward by the specified step size.
public void ScrollUp(double step)
```

```
//Scrolls the content upward by the default step size of 1.
public void ScrollUp()
```

```
//Scrolls the content downward by the specified step size.
public void ScrollDown(double step)
```

```
//Scrolls the content downward by the default step size of 1.
public void ScrollDown()
```

Properties

```
//Manages the visibility of the vertical scrollbar for the control.
public ScrollBarVisibility VerticalScrollBarVisibility
```

```
//Manages the visibility of the horizontal scrollbar for the control.
public ScrollBarVisibility HorizontalScrollBarVisibility
```

The same methods and properties are available for AlarmViewer Control.

Customizing window appearance

The following properties enable window customization for the DataGrid control:

Properties

```
//Flag to toggle visibility of RowSelectorPane (WPF Only).
public bool ShowRowSelectorPane
```

```
//Set BorderThickness for Alarm window.
public Thickness GridBorderThickness
```

The same methods and properties are available for AlarmViewer Control.

Responsive Sizing

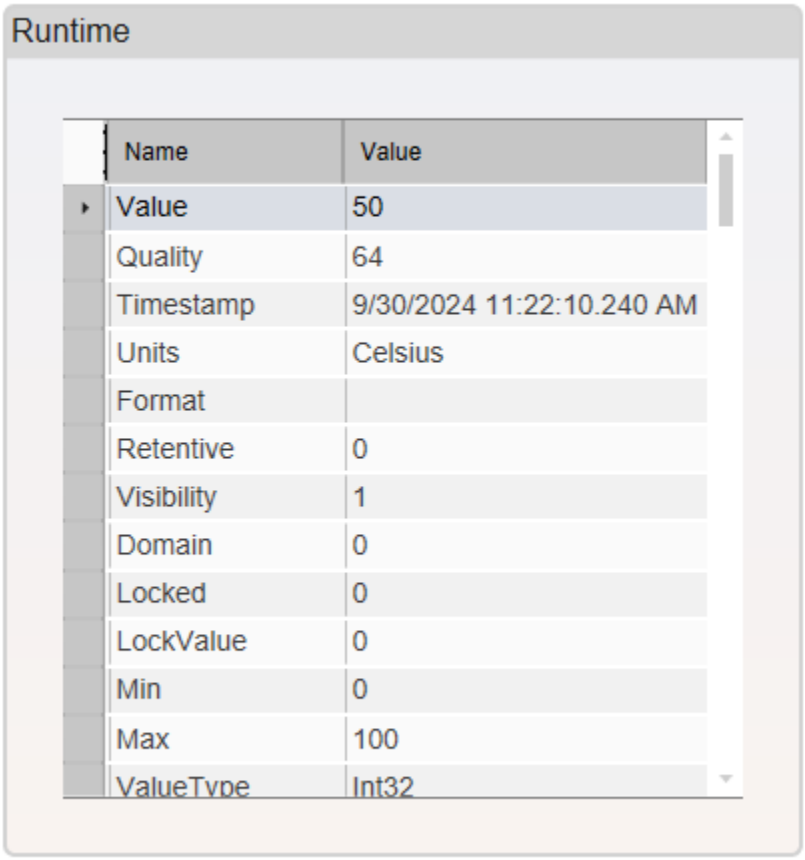
The DataGrid is a canonical prominent resizable element. To make it fill and grow as the Display resizes, set the Display's **OnResize** to **Responsive** and enable **IsWidthAlign** and **IsHeightAlign** on the DataGrid element. These flat element properties stretch the control with its container and take effect only when the Display is set to Responsive.

For the full responsive model, see Responsive Design and RESS.

Runtime Execution

Before using the DataGrid during runtime execution, configure data sources, binding modes, and column properties. To use the DataGrid, set it up in the display, define columns, and enable desired features like editing, sorting, and filtering through the configuration menu.

In a production environment, the DataGrid can be used to monitor real-time data from various sensors. An operator can sort and filter this data to quickly identify and address any anomalies.



In this section...
