



Essential Studio 2013 Volume 4 - v.11.4.0.26

## Essential Grid for Windows Forms



# Contents

<b>1</b>	<b>Overview</b>	<b>21</b>
1.1	Introduction to Essential Grid .....	21
1.2	Prerequisites and Compatibility .....	23
1.3	Choosing the Best Grid .....	24
1.3.1	Performance .....	25
1.3.2	Grouping .....	26
1.3.3	Summary .....	27
1.3.4	Sorting .....	27
1.3.4.1	Sort Icon Placement .....	28
1.3.4.1.1	Sample Link.....	29
1.3.4.1.2	Adding Sort Icon Placement to an Application .....	29
1.3.5	Excel Export .....	30
1.3.6	Filtering .....	31
1.3.6.1	Filter for Specific Columns .....	32
1.4	Documentation .....	34
<b>2</b>	<b>Installation and Deployment</b>	<b>36</b>
2.1	Installation.....	36
2.2	Sample and Location.....	36
2.3	Deployment Requirements .....	40
<b>3</b>	<b>Getting Started</b>	<b>43</b>
3.1	Tutorials .....	43
3.1.1	Before You Begin .....	44
3.1.2	Lesson 1: Grid Grouping Control Designer .....	44
3.1.2.1	Binding to an MDB File By Using VS 2003 .....	45
3.1.2.2	Binding to an MDB File By Using VS 2005 .....	64
3.1.3	Lesson 2: Grid Control Designer .....	77
3.1.3.1	Basic Grid Control .....	78
3.1.4	Lesson 3: Grid Data Bound Grid Designer.....	81
3.1.4.1	Basic Grid Data Bound Grid .....	82
3.1.4.2	Applying Special Column Formats .....	92

3.1.5	Lesson 4: Virtual Grid Tutorial.....	97
3.1.5.1	Creating the Project and Data Source.....	98
3.1.5.2	Adding Virtual Grid .....	101
3.1.5.3	Initializing the Virtual Grid.....	102
3.1.5.4	Style Properties .....	104
3.1.5.5	Handling Events to Retrieve Data for Virtual Grid.....	105
3.1.5.6	Saving Edited Values .....	107
3.1.5.7	Setting Properties in a Virtual Grid .....	110
3.1.5.8	Type Conversions.....	113
3.1.6	Lesson 5: Excel Export.....	113
3.1.6.1	Exporting the Grid Control or Grid Data Bound Grid To Excel.....	114
3.1.6.1.1	GridControl to Excel Conversion Process.....	115
3.1.6.2	Import an Excel Sheet into Essential Grid.....	117
3.1.6.3	Excel Converter Options .....	118
3.1.6.4	Exporting Grid Grouping Data To Excel.....	118
3.1.6.5	Exporting Multiple Grids .....	119
3.1.6.6	Exporting Grid Data to Excel 2010 .....	121
3.1.7	Lesson 6: Accessibility Information for Grid Windows Forms .....	122
3.2	List of Controls.....	130
<b>4</b>	<b>Grid Controls</b>	<b>132</b>
4.1	Grid Control .....	132
4.1.1	Creating Grid Control.....	133
4.1.1.1	Through Designer.....	133
4.1.1.2	Through Code.....	136
4.1.2	Elaborate Structure of the Control.....	138
4.1.3	Feature Summary.....	139
4.1.3.1	Cell Attributes .....	140
4.1.3.2	Cell Types.....	142
4.1.3.3	Data Binding .....	143
4.1.3.4	MS Office Simulation.....	143
4.1.3.5	Functionalities.....	144
4.1.3.6	General	146
4.1.4	Concepts and Features .....	146
4.1.4.1	Adding Special Controls to Grid Cells .....	147

4.1.4.1.1	Check Box .....	148
4.1.4.1.2	Color Edit.....	150
4.1.4.1.3	Combo Box.....	151
4.1.4.1.4	Control.....	155
4.1.4.1.5	Currency Edit.....	156
4.1.4.1.6	Formula Cell .....	158
4.1.4.1.7	Grid List Control .....	159
4.1.4.1.8	Header.....	161
4.1.4.1.9	Masked Edit.....	162
4.1.4.1.10	Month Calendar.....	164
4.1.4.1.11	Numeric Up Down .....	165
4.1.4.1.12	Progress Bar .....	166
4.1.4.1.13	Push Button.....	168
4.1.4.1.14	Rich Text .....	170
4.1.4.1.15	Slider .....	171
4.1.4.1.16	Static.....	172
4.1.4.1.17	Text Box .....	174
4.1.4.2	Cell Style Architecture .....	175
4.1.4.2.1	GridStyleInfo Class Overview.....	175
4.1.4.2.2	BaseStyles.....	181
4.1.4.2.3	GridRangeInfo .....	184
4.1.4.2.4	The GridControl.ChangeCells Method .....	185
4.1.4.2.5	Activating Current Cell Behavior .....	185
4.1.4.2.6	Scroll Cell into View.....	186
4.1.4.2.7	Managing Current Cell Operations.....	187
4.1.4.2.8	Show/Hide Current Cell Border.....	187
4.1.4.2.9	Refreshing Behavior of Current Cell .....	188
4.1.4.2.10	Style Properties .....	189
4.1.4.2.11	Grid New Feature .....	199
4.1.4.2.12	Browse-Only Grid .....	200
4.1.4.3	Deriving a Cell Control .....	201
4.1.4.3.1	GridCellModelBase .....	203
4.1.4.3.2	GridCellRendererBase .....	204
4.1.4.4	Custom Cell Types .....	206
4.1.4.4.1	Button Edit.....	206

4.1.4.4.2	Embeding OLE Objects in the Grid Cell.....	208
4.1.4.4.3	Calculator Text Box .....	209
4.1.4.4.4	Calendar .....	211
4.1.4.4.5	Date Time Picker .....	212
4.1.4.4.6	Enhanced Numeric Up Down .....	213
4.1.4.4.7	GridInCell.....	215
4.1.4.4.8	Link Label Cell.....	216
4.1.4.4.9	Picture Box .....	217
4.1.4.4.10	Slider .....	218
4.1.4.4.11	XHTML Cell .....	220
4.1.4.4.12	Chart Cell .....	222
4.1.4.4.13	Drop-DOWN Grid Cell .....	223
4.1.4.4.14	Drop-DOWN Form and User Control Cell .....	225
4.1.4.4.15	Integer Text Box .....	226
4.1.4.4.16	Double Text Box.....	227
4.1.4.4.17	Percent Text Box.....	228
4.1.4.5	MS Excel-like Features .....	230
4.1.4.5.1	Selection Frame .....	230
4.1.4.5.2	Current Cell .....	231
4.1.4.5.3	Workbook .....	232
4.1.4.5.4	Splitter .....	233
4.1.4.5.5	Freeze Pane .....	235
4.1.4.5.6	MultiLevel Undo and Redo .....	236
4.1.4.5.7	Find Replace .....	238
4.1.4.5.8	Unhide Column by Double-Clicking Disabled .....	240
4.1.4.5.9	Highlighting Row and Column Headers .....	241
4.1.4.6	Formula Support.....	243
4.1.4.6.1	Defining a FormulaCell.....	243
4.1.4.6.2	Using the Formula Library .....	244
4.1.4.6.3	Supported Arithmetic Operators and Calculation Precedence .....	244
4.1.4.6.4	Inside Essential Grid's Formula Support.....	245
4.1.4.6.5	Adding Formulas to the Formula Library.....	245
4.1.4.6.6	Function Reference Section .....	249
4.1.4.6.7	Cross Sheet Reference .....	378

4.1.4.6.8	Named Ranges .....	381
4.1.4.7	Populating a Grid Control .....	386
4.1.4.7.1	The Grid Control Indexer Technique .....	386
4.1.4.7.2	The GridControl.PopulateValues Method .....	387
4.1.4.7.3	Using the GridStyleInfo Class .....	389
4.1.4.8	Using the ReadOnly Attribute .....	395
4.1.4.8.1	Setting ReadOnly Grid Wide .....	395
4.1.4.8.2	Setting ReadOnly Cell by Cell .....	396
4.1.4.8.3	Making a Change to a ReadOnly Cell .....	396
4.1.4.9	Using Undo/Redo .....	397
4.1.4.9.1	The Basics .....	398
4.1.4.9.2	Transactions .....	399
4.1.4.9.3	Derived Commands .....	399
4.1.4.10	Serialization .....	400
4.1.4.10.1	Word Converter .....	400
4.1.4.10.2	Excel Export .....	402
4.1.4.10.3	PDF Export .....	405
4.1.4.10.4	SOAP, Binary and XML Serialization .....	409
4.1.4.10.5	Export as Image Support .....	409
4.1.4.11	Virtual Grids .....	413
4.1.4.11.1	Required Events .....	414
4.1.4.11.2	Optional Events .....	417
4.1.4.12	Pivot Grid .....	420
4.1.4.13	Appearance .....	426
4.1.4.13.1	GridFormatCellDialog .....	426
4.1.4.13.2	Grid Visual Styles .....	430
4.1.4.13.3	Grid Control Designer .....	432
4.1.4.13.4	Grid Properties .....	435
4.1.4.13.5	Custom Drawing .....	461
4.1.4.13.6	Formatting Drop-down List .....	464
4.1.4.14	Working with Rows and Columns .....	467
4.1.4.14.1	Hiding Rows and Columns .....	467
4.1.4.14.2	Header Rows and Columns .....	469
4.1.4.14.3	Frozen Rows and Columns .....	470
4.1.4.14.4	Moving Rows and Columns .....	472

4.1.4.14.5	Setting Column Widths and Row Heights .....	472
4.1.4.14.6	Setting Column Styles and Row Styles.....	473
4.1.4.14.7	Controlling the Resize Behavior .....	475
4.1.4.14.8	Resize To Fit .....	476
4.1.4.14.9	ResizeToFitOptimized .....	478
4.1.4.14.10	ResizeToFit Behavior in AutoSize.....	480
4.1.4.14.11	Autosizing Custom Cell .....	481
4.1.4.14.12	Enter Key Behavior .....	483
4.1.4.15	Covered Cells .....	486
4.1.4.16	Deriving GridPrintDocument.....	487
4.1.4.17	Floating Cells.....	489
4.1.4.18	TabBarSplitterControl.....	490
4.1.4.19	PrepareViewStyleInfo Event.....	492
4.1.4.20	Print Preview and Printing .....	493
4.1.4.21	Multiple Grid Printing .....	500
4.1.4.21.1	Adding Multi-Grid Printing to an Application.....	501
4.1.4.22	Drag Column Header .....	503
4.1.4.23	OLE Drag-and-Drop .....	504
4.1.4.24	Selection Modes .....	506
4.1.4.25	Banner Cells .....	508
4.1.4.26	Merge Cells Feature .....	510
4.1.4.27	Real-time Applications.....	513
4.1.4.27.1	Gaming Applications .....	513
4.1.4.27.2	MS Excel 2007-like UI .....	514
4.1.4.27.3	Grid Folder Browser .....	515
4.1.4.28	Multiple Document Interface (MDI) Support.....	518
4.1.4.29	Clipboard Support .....	519
4.1.4.29.1	Managing Clipboard Operations with GridModelCutPaste .....	520
4.1.4.29.2	Clipboard Events .....	525
4.1.4.30	Performance .....	525
4.1.4.30.1	High Frequency Real Time Updates .....	525
4.1.4.30.2	Data Handling.....	527
4.1.4.31	Zooming options .....	531
4.1.4.31.1	Cell-level and grid-level .....	531

4.1.4.32	Tile Image In Grid Cell.....	536
4.2	Grid Data Bound Grid .....	537
4.2.1	Creating Grid Data Bound Grid .....	538
4.2.1.1	Through Designer.....	538
4.2.1.2	Through Code.....	544
4.2.2	Concepts and Features .....	545
4.2.2.1	Binding to an ArrayList .....	546
4.2.2.1.1	ArrayList Class with IBindingList Support .....	548
4.2.2.2	Binding to a DataTable .....	549
4.2.2.3	Accessing Values in the Grid Data Bound Grid and in the Data Source	551
4.2.2.4	Using the CurrencyManager .....	552
4.2.2.5	Filtering a Grid Data Bound Grid.....	553
4.2.2.6	GridBoundColumns and Controlling the Column Format.....	557
4.2.2.6.1	Using the GridDataBoundGrid.Binder Class .....	560
4.2.2.7	Changing Column Order in a Grid Data Bound Grid.....	561
4.2.2.8	Using a Master-Details Relation.....	561
4.2.2.9	Foreign Key Columns: Showing One Value but Saving Another .....	563
4.2.2.10	Sorting .....	567
4.2.2.11	Sort by DisplayMember .....	568
4.2.2.12	Data Relations .....	571
4.2.2.12.1	Nested Drop-down Grids .....	571
4.2.2.12.2	Multiple Nested Relations.....	574
4.2.2.12.3	Hierarchical Grid with Tree Lines .....	575
4.2.2.12.4	ExpandAll and CollapseAll Methods .....	576
4.2.2.13	Multi Row Record .....	578
4.2.2.14	Record Navigation Bar .....	579
4.2.2.15	Multiple Headers.....	582
4.2.2.16	Grid Data Bound Grid Performance .....	583
4.2.2.17	Events.....	585
4.2.2.18	Enables Migration of .Net Grid to Essential Grid .....	587
4.2.2.19	Field Chooser for Grid Data Bound Grid .....	590
4.3	Grid Grouping Control .....	593
4.3.1	Creating Grid Grouping Control.....	593
4.3.1.1	Through Designer.....	594

4.3.1.2 Through Code.....	594
4.3.1.2.1 Binding a Grid Grouping Control to an MDB File .....	594
4.3.1.2.2 Bind a Grid Grouping Control to a Manual Data Source .....	596
4.3.2 Feature Summary.....	600
4.3.3 Major Control Classes Overview .....	610
4.3.4 Concepts and Features .....	612
4.3.4.1 Performance.....	612
4.3.4.1.1 Memory Performance - Engine Optimizations .....	616
4.3.4.1.2 ListChanged Performance.....	633
4.3.4.1.3 High Frequency Updates.....	652
4.3.4.1.4 Grouping Performance .....	667
4.3.4.1.5 IList Grouping Performance .....	669
4.3.4.2 Data Binding.....	672
4.3.4.2.1 Data Binding using ADO.NET .....	673
4.3.4.2.2 Binding to XML Data .....	674
4.3.4.2.3 Binding to Custom Collections .....	676
4.3.4.2.4 Unbound Mode .....	696
4.3.4.3 Data Representation.....	702
4.3.4.3.1 Grouping.....	702
4.3.4.3.2 Sorting .....	736
4.3.4.3.3 Summaries .....	744
4.3.4.3.4 Filters and Expressions .....	761
4.3.4.3.5 Relations and Hierarchy .....	795
4.3.4.4 Paging Support in GridGrouping Control .....	842
4.3.4.4.1 Paging support with a different data source .....	843
4.3.4.5 Appearance .....	844
4.3.4.5.1 Appearance Options.....	848
4.3.4.5.2 Format Cells Dialog Support .....	864
4.3.4.5.3 Conditional Formatting .....	866
4.3.4.5.4 Dynamic Formatting .....	869
4.3.4.5.5 Merge Cells Dynamically.....	872
4.3.4.5.6 BaseStyles.....	873
4.3.4.5.7 Get Cell Styles.....	878
4.3.4.5.8 Look and Feel.....	881

4.3.4.5.9	Table Options .....	885
4.3.4.6	Grid Layout.....	892
4.3.4.6.1	Stacked MultiHeaders .....	892
4.3.4.6.2	Multi Row Record .....	901
4.3.4.6.3	Grid Field Chooser .....	905
4.3.4.6.4	Field Chooser for Stacked Header .....	909
4.3.4.7	Selections .....	913
4.3.4.7.1	Model Based Selection.....	913
4.3.4.7.2	Record Based Selection.....	916
4.3.4.7.3	Focused Selection.....	923
4.3.4.7.4	Multiple Record Selection.....	931
4.3.4.7.5	Selected Ranges Collection .....	936
4.3.4.8	Serialization .....	938
4.3.4.8.1	Xml Serialization.....	938
4.3.4.8.2	Excel Export .....	946
4.3.4.8.3	Word Converter .....	960
4.3.4.8.4	PDF Converter .....	962
4.3.4.9	Real Time Applications.....	965
4.3.4.9.1	Portfolio Grid.....	965
4.3.4.9.2	Outlook 2007 Demo .....	969
4.3.4.10	Grid Designer .....	971
4.3.4.11	Navigation Bar .....	985
4.3.4.12	Print and Print Preview .....	987
4.3.4.13	Advanced Features .....	992
4.3.4.13.1	Custom Grouping .....	993
4.3.4.13.2	Custom Sorting.....	1000
4.3.4.13.3	Custom Summary.....	1005
4.3.4.13.4	Custom Filter Dialog .....	1009
4.3.4.13.5	Filter Optimization .....	1020
4.3.4.14	Events.....	1021
4.3.4.14.1	Table Events .....	1022
4.3.4.14.2	Cell Events .....	1040
4.3.4.14.3	Current Cell Events .....	1047
4.3.4.14.4	Control Events.....	1050
4.3.4.14.5	Mouse Events.....	1052

4.3.4.14.6	Key Events .....	1057
4.3.4.14.7	Selection Events for Filter Bar.....	1059
4.3.4.15	Delete Collection of Records in GridGroupingControl.....	1061
4.3.4.16	Select Collection of Records In the GridGroupingControl.....	1063
4.3.4.17	Export Summary as Caption .....	1065
4.3.4.17.1	Searching for text in multiple columns .....	1066
4.4	Grid List Control.....	1068
4.4.1	Creating Grid List Control.....	1068
4.4.1.1	Through Designer.....	1069
4.4.1.2	Through Code.....	1074
4.4.2	Concepts and Features .....	1075
4.4.2.1	Data binding and Selection Modes.....	1075
4.4.2.2	ComboBoxBase Feature .....	1077
4.4.2.3	Customizing List control .....	1078
4.5	Grid Record Navigation Control .....	1083
4.5.1	Creating Grid Record Navigation Control.....	1083
4.5.1.1	Through Designer.....	1084
4.5.1.2	Through Code.....	1086
4.5.2	Built-in Navigation Support for RecordNavigationControl in GridGroupingControl .....	1087
4.6	Grid Aware Text Box .....	1090
4.7	Grid Helper Classes .....	1091
4.7.1	Grid to PDF Conversion .....	1091
4.7.2	Resizing Heights of Individual Rows in Grid.....	1096
4.7.3	Grid Dynamic Filter.....	1097
4.7.4	Setting up Foreign Key Relations.....	1100
4.7.5	Custom Cell Types .....	1102
4.7.6	Grid Field Chooser .....	1117
4.7.7	Filtering By Display Member .....	1118
4.7.8	Grid Format Cell Dialog .....	1121
4.7.9	Printing .....	1123
4.7.10	Word Converter .....	1127
4.7.11	Fill Series.....	1134
4.7.12	Excel-Like Filters .....	1138
4.7.12.1	Office2007Filter .....	1138

4.7.12.2	Optimized GridExcelFilter.....	1140
4.7.12.3	Filtering Null Values from the Grid .....	1142
4.7.13	Card View Layout .....	1143
4.7.13.1	Tables for Properties, Methods, and Events .....	1143
4.7.13.2	Enable the Card View Layout.....	1146
4.8	Office2010 Theme in Windows Grids.....	1146
4.9	Enhanced Visual Styles for the Syncfusion Windows Grids .....	1151
4.9.1	Applying Enhanced GridVisualStyle to the Application .....	1152
4.9.2	Combo Box.....	1153
4.10	Metro Theme for Essential Grid Controls .....	1154
4.10.1	Tables for Properties and Events .....	1154
4.10.2	Applying Metro Theme to a Control.....	1155
4.10.3	Applying User-Defined Colors as Metro Themes.....	1156
4.11	Coded UI Support in Windows Grids.....	1158
4.11.1	Deploying Extension Assembly .....	1159
4.11.2	Preparing the Grid Application .....	1160
4.11.3	Creating Unit Tests with VS2010 .....	1160
4.11.4	Testing the Application with Generated Coded UI Tests .....	1162
4.11.5	Properties .....	1166
4.11.6	In-built Coded UI Support for 3.5 and 4.0 Frameworks.....	1177
4.12	Built-in Error Provider Support.....	1178
4.12.1	Enabling Error Alert .....	1179
4.13	Support for Skin Manager in Windows Forms Grid.....	1182
4.13.1	Adding Skin Manager to an Application .....	1183
4.14	Localization Support.....	1184
4.14.1	Localization Support for ComboBox Cell.....	1188
4.15	Zoom Support.....	1190
4.15.1	Zooming Grid Controls .....	1192
4.15.2	Zooming Individual Cells of the Grid .....	1195
<b>5</b>	<b>Frequently Asked Questions</b>	<b>1197</b>
5.1	GridControl .....	1197
5.1.1	How to Align Radio Buttons.....	1197
5.1.2	How to Add a Sort Icon (Up / Down Arrow) in a GridControl's Column Header .....	1198
5.1.3	How to Avoid the A, B, C and / or 1, 2, 3 in the Headers .....	1199

5.1.4	How to Avoid RangeStyles being Written out to Code in a Derived GridControl .....	1200
5.1.5	How to Change the Appearance of a Single Header Cell .....	1201
5.1.6	How to Change the Backcolor of a Column .....	1202
5.1.7	How to Change the Backcolor of a Single Cell .....	1202
5.1.8	How to Change the Backcolor of a Single Row .....	1203
5.1.9	How to Change a Cell's Font to a Particular Font Family .....	1204
5.1.10	How to Change the Look of a Cell's Border .....	1204
5.1.11	How to Create a Multi-Select DropDownList in a Cell.....	1206
5.1.12	How to Create a Cell that contains Hypertext Link along with Different Formatted Texts .....	1207
5.1.13	How to Exclude Hidden Rows and Column for Scrolling? .....	1209
5.1.14	How to Get all the Data in a GridControl as an Array .....	1210
5.1.15	How to hide the numbered row and column headers while printing or print previewing in GridControl .....	1211
5.1.16	How to Implement a Fill Paste in the Grid.....	1212
5.1.17	How to Include an Icon in the Column Header.....	1214
5.1.18	How to Optimize Pixel Scrolling in a GridControl .....	1215
5.1.19	How to pre-select a checkbox inside a grid cell .....	1218
5.1.20	How to Retrieve the Text From a Cell .....	1219
5.1.21	How to save the contents of a Grid in memory rather than a file system to export to another grid .....	1220
5.1.22	How to Set the Cell Properties for a Range of Cells .....	1220
5.1.23	How to Set the Number of Rows / Columns.....	1221
5.1.24	How to Set the Height of a Row .....	1222
5.1.25	How to Set a Value into a Cell.....	1223
5.1.26	How to Set the Text in a Header Cell .....	1224
5.1.27	How to Set the Width of a Column .....	1225
5.1.28	How to Set the Text Color that Appears in a Cell.....	1225
5.1.29	How to Set Transparent Backcolor for a GridControl .....	1226
5.1.30	How to Show a ContextMenu in the Center of CurrentCell or Selected Cells Irrespective of the Mouse Click .....	1227
5.1.31	How to Show Multiple Images in a Cell .....	1228
5.1.32	How to Swap Rows and Columns .....	1229
5.1.33	How to Synchronize the Selection of Multiple Grids .....	1230
5.1.34	How to Use a Combo Box in a Cell .....	1232

5.1.35	How to Use a ColorEdit Control in a Cell and Retrieve its Value .....	1235
5.1.36	How to Use a PushButton in a Cell and Catch the User Clicking It .....	1236
5.1.37	What is the Difference between TextAlign, HorizontalAlignment and VerticalAlignment?.....	1237
5.1.38	How can we make the Current Row Bold?.....	1238
5.1.39	How to save the ComboBox cell value instantly after the dropdown is closed? 1240	
5.2	GridDataBoundGrid.....	1241
5.2.1	How to Change the Appearance of a Single Header Cell .....	1241
5.2.2	How to Change the Backcolor of a Column .....	1242
5.2.3	How to Change the Backcolor of a Single Cell .....	1243
5.2.4	How to Change the Backcolor of a Single Row .....	1244
5.2.5	How to Change the Column Header Text .....	1245
5.2.6	How to Change the Look of a Column's Border .....	1246
5.2.7	How to change the row header to display line numbers instead of the black triangle in GridDataBoundGrid.....	1247
5.2.8	How to Copy a Range of Cells to the Clipboard.....	1249
5.2.9	How to draw a check box cell in a GridDataBoundGrid header .....	1250
5.2.10	How to Insert an Unbound CheckBox Column.....	1252
5.2.11	How to Include an Icon in the Column Header.....	1257
5.2.12	How to Make the Particular Cells ReadOnly .....	1258
5.2.13	How to Make the GridFilterBar Use an Autocomplete Combo Box .....	1259
5.2.14	How to merge two columns in a GridDataBoundGrid .....	1261
5.2.15	How to Paste Clipboard Contents Bigger than GridDataBoundGrid Row and Column Count.....	1262
5.2.16	How to Prevent Showing the '+' Sign Next to the Parent Rows with no Children .....	1265
5.2.17	How to QueryCellFormattedText and SaveCellFormattedText.....	1269
5.2.18	How to Retrieve the DataRow from the GridDataBoundGrid with the RowIndex.....	1270
5.2.19	How to Set the Width of a Column .....	1271
5.2.20	How to Set the Height of a Row .....	1272
5.2.21	How to Stop the Errors Thrown when Pasting Larger Clipboard Contents in a GridDataBoundGrid .....	1273
5.2.22	How to Support Tri-State Sorting in a GridDataBoundGridControl .....	1274
5.2.23	How to Use a Combo Box in a Column.....	1277
5.2.24	How to Use a ColorEdit Control in a Column and Retrieve its Value.....	1280

5.2.25	How to Use a PushButton in a Column and Catch the User Clicking It .....	1281
5.2.26	How to Get Selected Rows in GridDataBoundGrid.....	1282
5.2.27	How to Get Selected Ranges in GridDataBoundGrid .....	1283
5.2.28	How to Determine that No Cell is Selected .....	1285
5.2.29	How to Disable Sorting While Record Added.....	1286
5.2.30	How to Apply Filtering on the GridDataBound Grid based on a Node Selected in the Tree View? .....	1287
5.2.31	How to select all the contents in a cell after double-clicking on the contents .....	1290
5.2.32	How to resolve the “Object reference not set to an instance of an object” error generation in GridDataBoundGrid Designer.....	1291
5.2.33	How to achieve Numeric Column Sorting in GridDataBoundGrid .....	1292
5.3	GridGrouping Control .....	1293
5.3.1	General.....	1293
5.3.1.1	How to Access the Corresponding Parent Table's Row from the Child .....	1293
5.3.1.2	How to access the current record.....	1295
5.3.1.3	How to add and format unbound columns in GridGrouping control .....	1296
5.3.1.4	How to change the caption text .....	1296
5.3.1.5	How to disable the resizing of rows and columns .....	1297
5.3.1.6	How to Efficiently Customize the Child Table / Group using a Custom Engine.....	1298
5.3.1.7	How to group a column programmatically.....	1298
5.3.1.8	How to Hide the Custom Option in the FilterBar DropDown .....	1298
5.3.1.9	How to Hide / Unhide the Columns in a Grouping Grid.....	1303
5.3.1.10	How to Place a Checkbox in a Header Cell .....	1303
5.3.1.11	How to Place a UserControl in a Header Cell .....	1305
5.3.1.12	How to Prevent Columns Resizing for Child Tables .....	1307
5.3.1.13	How to reject the changes made to the GridGroupingControl .....	1308
5.3.1.14	How to resize the columns to fit in a page while exporting to PDF .....	1309
5.3.1.15	How to Make Use of the Journal Control using GridGrouping Controls .....	1310
5.3.1.16	How to Get New Form Window in Front of an Active Window while Double Clicking a Cell .....	1313
5.3.2	Nested Tables .....	1314

5.3.2.1 How to access child table's display elements .....	1314
5.3.2.2 How to Access the Nested Tables in a GridGroupingControl .....	1315
5.3.2.3 How to Get a Value from the Parent Record of the Nested Table when the Dropdown List is Clicked .....	1316
5.3.2.4 How to hide the row headers of a child table in the GridGroupingControl .....	1318
5.3.2.5 How to Set Column Style Properties for the Nested Tables in a Grouping Grid .....	1319
5.3.2.6 How to set style properties of a nested table .....	1321
5.3.2.7 How does one add currency symbol for parent table and child table	1322
5.3.2.8 Foreign Key Reference For Child Table.....	1323
5.3.3 Summaries .....	1326
5.3.3.1 How to align the summary cells in the GridGroupingControl .....	1326
5.3.3.2 How to create weighted summaries in the GridGroupingControl.....	1327
5.3.3.3 How to define a summary row in the grid.....	1328
5.3.3.4 How to format summary rows.....	1329
5.3.3.5 How to retrieve a summary item.....	1330
5.3.3.6 How to show Summary Results in the Inner Most Table on every Parent Level in Nested Related Tables.....	1330
5.3.3.7 How to Update Summaries Immediately Upon Changing a Field .....	1332
5.3.3.8 What are the options in the summary columns? .....	1333
5.3.3.9 How to Get Summary Cell Values in sorting? .....	1333
Or	1333
How to Incorporate Summary Cells while Sorting?.....	1333
Or	1333
How to Sort the Grid based on the Summary Cell Values? .....	1333
5.3.3.10         How to Change the Format of the Summary Cell in Group Caption .....	1334
5.3.4 DisplayElements .....	1335
5.3.4.1 How to access all the DisplayElements or a particular DisplayElement in the GridGrouping control .....	1336
5.3.4.2 How to access a particular DisplayElement if RowIndex is provided .....	1336
5.3.4.3 How to find a DisplayElement Type .....	1337
5.3.5 Layout and Appearance .....	1337
5.3.5.1 How to Allocate Equal Size for Each of the Columns in all the Tables	1337

5.3.5.2 How to create multiple rows per record in a GridGroupingControl .....	1338
5.3.5.3 How to freeze specified columns.....	1339
5.3.5.4 How to set conditional formatting in the GroupingGrid.....	1340
5.3.5.5 How to set texts in the preview record .....	1340
5.3.5.6 How to set the cursor to GridGroupingControl instead of the default one? .....	1341
5.3.5.7 How to apply text color for a RichText celltype in GridGroupingControl .....	1342
5.3.6 Context Menu .....	1344
5.3.6.1 How to attach a context menu to a cell in the GridGrouping control 1344	
5.3.6.2 How to attach a context menu to the GridGrouping control .....	1345
5.3.7 Expressions .....	1346
5.3.7.1 How to add Expression columns.....	1346
5.3.7.2 What are the various ExpressionColumn options? .....	1347
5.3.8 Selections .....	1348
5.3.8.1 How to access selected records.....	1348
5.3.8.2 How to select a record programmatically .....	1349
5.3.8.3 How to set ListBoxSelectionModes .....	1349
5.3.8.4 What are the events fired when the records are selected? .....	1350
5.3.9 Records .....	1351
5.3.9.1 How to access a particular record from a table .....	1351
5.3.9.2 How to access a record given the row index.....	1352
5.3.9.3 How to access sorted or filtered records .....	1352
5.3.9.4 How to access unfiltered records .....	1353
5.3.9.5 How to get a record index given the rowindex .....	1353
5.3.9.6 How to get a rowindex given the record index .....	1354
5.3.9.7 How to get the row position of the AddNewRecord in the GridGroupingControl .....	1354
5.3.9.8 What are the events fired when records are collapsing or collapsed? .....	1355
5.3.9.9 What are the events fired when the records are expanded / expanding? .....	1356
5.3.9.10         What are the events fired when the record values are changed? .....	1357
5.3.10 Datasource .....	1358

5.3.10.1	How to Get the Position of a Row in the DataSource from the Current Record.....	1358
5.3.10.2	How to set up a datasource to the grouping grid .....	1359
5.3.11	Groups 1360	
5.3.11.1	How to access all the groups .....	1360
5.3.11.2	How to access a particular group .....	1361
5.3.11.3	How to access the group from the DisplayElements.....	1362
5.3.11.4	How to access the group from the Record.....	1363
5.3.11.5	How to apply grouping properties for a particular column.....	1363
5.3.11.6	How to apply grouping properties for ChildLevelGroups .....	1364
5.3.11.7	How to apply grouping properties for TopLevelGroups.....	1365
5.3.11.8	How to derive user defined groups .....	1366
5.3.11.9	What are the events fired when the group is collapsing / collapsed? .....	1368
5.3.11.10	What are the events fired when the group is expanded / expanding? .....	1370
5.3.12	How to Move Groups Upward/Downward Using Custom Comparer .....	1371
5.3.13	Sorting 1374	
5.3.13.1	How to Do Custom Sorting for a Specified String Column.....	1374
5.3.13.2	How to perform Custom Sorting in GridGroupingControl.....	1375
5.3.13.3	How to prevent sorting if the column has the same value in each cell?.....	1378
5.4	Common to GridControl, GridDataBoundGrid and GridGrouping .....	1380
5.4.1	How to Add Conditional Formatting to Rows.....	1381
5.4.2	How to Capture Function Keys When the Current Cell is in Active Edit Mode.....	1382
5.4.3	How to Capture Mouse and Key Events When the Text Box Cell is in an Active State .....	1383
5.4.4	How to Change the Color of All Headers .....	1385
5.4.5	How to Change the Size of the Combo Box Button .....	1386
5.4.6	How to Change the Mouse Cursor for a GridControl .....	1386
5.4.7	How to Control the Number of Visible Items in a Combo Box Cell .....	1388
5.4.8	How to Control the Way the Grid Handles Exceptions.....	1389
5.4.9	How to Control Whether OLE Drag-and-Drop is Supported in the Grid.....	1390
5.4.10	How to convert the contents of a GridControl or GridDataBoundGridControl to Excel .....	1391
5.4.11	How to Disable Clipboard Cut / Copy / Paste in a Grid.....	1392

5.4.12	How to Display Placeholder Characters when Cell Content Exceeds Cell Width	1393
5.4.13	How Does the Helper class Support Percentage Sizing in GridControl / GridDataBoundGrid .....	1396
5.4.14	How to export CellCommentTips to Excel using GridExcelConverterControl .....	1400
5.4.15	How to Get the Selected Ranges in a Grid .....	1401
5.4.16	How to Get the Cell Coordinates Under a Given Point .....	1403
5.4.17	How to Get the Top / Bottom / Left / Right Viewable Row and Column Indexes	1404
5.4.18	How to Get the Screen Point for the Given Cell Coordinates .....	1404
5.4.19	How to Get the New Value and the Old Value when an Item is Selected in a Combobox Cell .....	1405
5.4.20	How to Have Character Casing Settings for a Cell .....	1407
5.4.21	How to Hide a Row / Column .....	1408
5.4.22	How to Make a Cell Display '...' if it is Not Wide Enough .....	1408
5.4.23	How to Make the Grid Behave Like a List Box .....	1409
5.4.24	How to make resizing possible in additional row headers of a GridControl and GridDataBoundGrid .....	1410
5.4.25	How to Move to the Next Row from the Last Cell of a Row .....	1411
5.4.26	How to Prevent Resizing a Specific Column in a GridControl .....	1411
5.4.27	How to Print a Grid .....	1412
5.4.28	How to Print Preview a Grid .....	1414
5.4.29	How to Put a CheckBox in a Header Cell in a GridControl or GridDataBoundGrid .....	1415
5.4.30	How to Put a Cell in Overstrike Mode so Characters get Replaced instead of Inserted as you type .....	1417
5.4.31	How to Put a ComboBox in a Header Cell in a GridControl or GridDataBoundGrid .....	1419
5.4.32	How to Set the Background Color for a Grid .....	1422
5.4.33	How to Select All Text in a Grid After Clicking a Cell .....	1423
5.4.34	How to Size Column Widths or Row Heights to Fit the Text .....	1424
5.4.35	How to Validate Changes Made to a Grid Cell .....	1425
5.4.36	How to Write Syntax for Grid Model Properties .....	1426
5.4.36.1	Show or Hider Header .....	1428
5.4.36.2	Customize the Appearance .....	1429
5.4.36.3	Printing Options .....	1431
5.4.36.4	Grid Model Options .....	1432

5.4.37 How to Overcome SendKey Exception in Currency Cell .....	1434
5.4.38 How To Release the Tab Focus from Grid.....	1434
5.4.39 How To Resize Row Height Based On the Font of the Grid Cell Content .....	1435
5.4.40 How to Align Image Button at the Centre of a Cell.....	1437
5.4.41 How to Hold the Row Selection After the Cell is Deactivated .....	1438
5.4.42 How to Suppress KeyDown Event .....	1439
5.4.43 How to Restrict Alphabetical Characters in NumericUpDown Cell .....	1441
5.4.44 How to Enable ColorStyle Settings in Windows Grids .....	1441
5.4.45 How to trigger an event when the ComBox DropDownList has null values or has no datasource bound to it? .....	1443
5.4.46 How to get the selected text from a TextBox cell? .....	1444
5.4.47 How to enable double-click on the formula cell? .....	1445
5.4.48 Why does the call to the Application.DoEvents never return while updating the Grid using the BeginUpdate() method? .....	1445
5.4.49 How can I implement an Excel accounting format in the Syncfusion WinForms GridControl? .....	1447
5.4.50 How to achieve Excel-like Text Alignment in Grid Control? .....	1448
5.4.51 How to export the user-defined function from Grid to Excel workbook? .....	1449

# 1 Overview

---

This section covers information on Essential Grid, its key features, prerequisites to use the control, its compatibility with various OS and browsers, and finally the documentation details complimentary with the product. It comprises the following sub-sections:

## 1.1 Introduction to Essential Grid

Essential Grid is a powerful grid control that is implemented by using a unique styles architecture. It offers Microsoft Excel-like grid functionality and high performance grouping support for both flat and hierarchical data sources. It is a 100% Native .NET UI library that provides several packages for building modern Windows applications by using the Microsoft .NET framework. The Grid Grouping control is a powerful databound grid control that supports multi-column sorting, filtering, Outlook style grouping, summaries, expression fields, conditional formats and unbound fields.

Essential Grid controls are useful for those users who want to build a Windows application which can be used in any .NET environment including C#, VB.NET and C++. It is beneficial to those users who want to create true nested grids with hierarchical data, and also display multiple, unrelated tables in one grid.

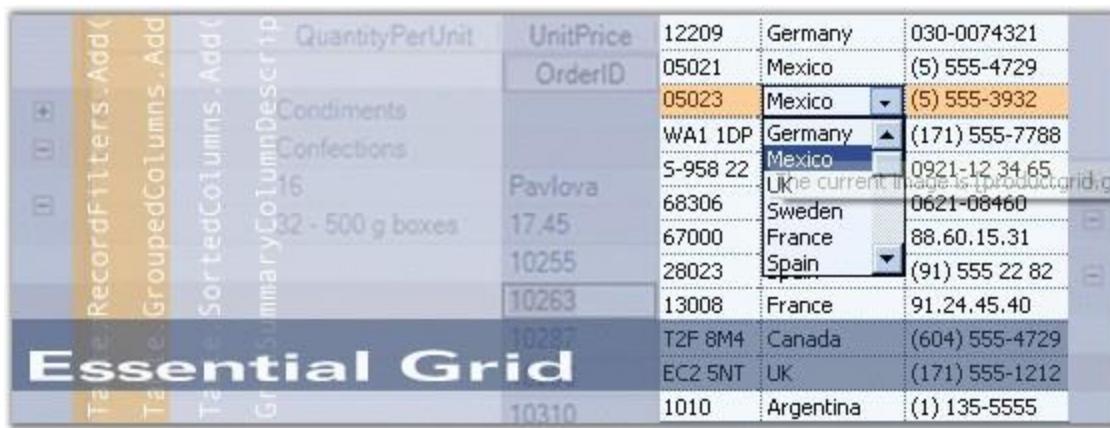


Figure 1: Essential Grid

### Key Features

- Multi-level Undo/Redo, shared scrollbar support, data/view separation, floating cells, more than 18 cell types, and unmatched extensibility are unique features of the Essential Grid.
- Grid cell can have specialized controls such as a Text Box, Check Box or Combo Box.
- Grid supports data binding techniques such as ADO.NET, Virtual mode, bound and unbound mode, and so on.

- Essential Grid supports Office features like dynamic splitters and undo/redo. It can have a tabbed workbook format; also supports IntelliMouse Scrolling similar to MS Excel.
- Grid supports functionalities like OLE drag-and-drop, resizing of rows and columns through property settings, and special event handling. It allows extremely detailed customization down to the cell level.

Grid Grouping control has rich design time support. The grouping grid supports variety of data sources which are used to automatically populate the grid with data. It has full ADO+ support, and also allows any component that implements the **IList**, **IBindingList**, **ITypedList** or **IListSource** interface.

- Grid data can be arranged based on matching field values to form groups. Essential Grid supports nested grouping by hierarchically combining the groups in different levels. The number of levels of grouping is unlimited.
- Grid supports data presentation techniques like sorting, grouping, filtering, and so on.

The product comes with numerous samples as well as an extensive documentation to guide you. This User Guide provides detailed information on the features and functionalities of the Grid, Grid Data Bound Grid, Grid Grouping, Grid List, Grid Record Navigation and Grid Aware Text Box controls. It is organized into the following sections:

- **Overview**-This section gives a brief introduction to our product and its key features.
- **Installation and Deployment**-This section elaborates on the install location of the samples, license, and so on.
- **What's New**-This section lists the new features implemented for every release.
- **Getting Started**-This section guides you on getting started with Grid controls.
- **Grid Controls**-The features of the Grid, Grid Data Bound Grid, Grid Grouping, Grid List, Grid Record Navigation and Grid Aware Text Box controls are illustrated with use case scenarios, code examples and screen shots under this section.
- **Frequently Asked Questions**-This section discusses specific solutions for very specific tasks.

## **Document Conventions**

The conventions listed below will help you to quickly identify the important sections of information, while using the content:

Convention	ICON	Description
Note	<b>Note:</b>	Represents important information
Example	Example	Represents an example
Tip		Represents useful hints that will help you in using the controls/features
Additional Information		Represents additional information on the topic

## 1.2 Prerequisites and Compatibility

This section covers the requirements mandatory for using Essential Grid. It also lists operating systems and browsers compatible with the product.

### Prerequisites

The prerequisites details are listed below:

<b>Development Environments</b>	<ul style="list-style-type: none"><li>• Visual Studio 2013</li><li>• Visual Studio 2012 (Ultimate, Premium, Professional, and Express)</li><li>• Visual Studio 2010 (Ultimate, Premium, Professional and Express)</li><li>• Visual Studio 2008 (Team, Professional, Standard and Express)</li><li>• Visual Studio 2005 (Team, Professional, Standard and Express)</li><li>• Borland Delphi for .NET</li><li>• SharpCode</li></ul>
<b>.NET Framework versions</b>	<ul style="list-style-type: none"><li>• .NET 4.5</li><li>• .NET 4.0</li><li>• .NET 3.5</li><li>• .NET 2.0</li></ul>
<b>Operating Systems</b>	<ul style="list-style-type: none"><li>• Windows 8.1 (32 bit and 64 bit)</li><li>• Windows Server 2013 (32 bit and 64 bit)</li><li>• Windows Server 2008 (32 bit and 64 bit)</li><li>• Windows 7 (32 bit and 64 bit)</li><li>• Windows Vista (32 bit and 64 bit)</li><li>• Windows XP</li></ul>

	<ul style="list-style-type: none"><li>• Windows 2003</li></ul>
--	--

## Compatibility

The compatibility details are listed below:

<b>Operating Systems</b>	<ul style="list-style-type: none"><li>• Windows Server 2008 (32 bit and 64 bit)</li><li>• Windows 7 (32 bit and 64 bit)</li><li>• Windows Vista (32 bit and 64 bit)</li><li>• Windows XP</li><li>• Windows 2003</li></ul>
--------------------------	---

## 1.3 Choosing the Best Grid

The **GridControl** (GC) is a powerful grid control that is implemented using unique styles architecture. Grid control, which is similar to Microsoft Excel, allows extremely detailed customization down to the cell level. Multilevel undo-redo, shared scrollbar support, data view separation, floating cells, more than 18 cell types, and unmatched extensibility are the unique features of the Essential Grid. This is a complete native .NET UI library, which provides several packages for building modern Windows applications using the Microsoft .NET framework.

The **GridGroupingControl** (GGC) is a column-row oriented grid that is used to bind to a data source. Essential Grouping allows you to easily group, sort, filter, and summarize your data. It can display true nested grids with hierarchical data, and it can also display multiple, unrelated tables in one grid.

The **GridDataBoundGrid** (GDBG) has been designed to be used as a grid bound to a data source such as an ADO.NET data set or data table. No data values are stored in the GridDataBoundGrid object. In GDBG, individual columns act as single entity (column-centric). Unlike the **GridControl**, GDBG is data bound.

This section analyses the efficiency of the Grids based on the most important features. The following are the important features:

- Performance
- Grouping
- Sorting
- Summary
- ExcelExport
- Filtering

Control	Grouping	Sorting	Summary	Filtering	Export
GGC	✓	✓	✓	✓	✓
GDBG	-	✓	-	✓	✓
GC	-	*	-	-	✓

\*Sorting can be performed through customizing the *GridSortColumnHeaderCellModel*.

### 1.3.1 Performance

#### GridGroupingControl

The **GridGroupingControl** has an extremely high-performance standard. It can handle very high frequency updates and refresh scenarios. It also provides complete support for Virtual Mode wherein the data will be loaded only on demand. By simply setting a few properties, you can have the grid work with large amounts of data without affecting the performance.

 **Note:** For more details, refer to the following section:

[Performance](#)

#### **GridControl and GridDataBoundGrid**

**GridControl** has an extremely high performance standard. It can handle high frequency updates and work with large amounts of data without affecting the performance, similarly **GridDataBoundGrid** can also handle large amount of data without affecting the performance.

 **Note:** For more details, refer to the following section:

[Grid Data Bound Grid Performance](#) and [Performance](#)

### **1.3.2 Grouping**

#### **GridGroupingControl**

The **GridGroupingControl** alone supports grouping. This control is exclusively designed for grouping. This control supports grouping data at design time too. Grouping at design time displays the grid as a tree view. For non-nested data tables, you can use this control to quickly provide custom views of data.

The Grid Grouping control allows you to group the data based on one or more columns. When grouping is applied, the data will be organized into a hierarchical structure based on the matching field values. This control enables you to combine identical value to form a group. Each group is identified by its *GroupCaptionSection*. You can expand this to view the records in the group. The *GroupCaptionSection* provides the information about a particular group such as the group name, number of items in the group and so on. The Grid Grouping control also provides Expand/Collapse button for every group.

 **Note:** For more details, refer to the following section:

[Grouping](#)

#### **Sample**

A sample of this feature is available in the following location:

`<Install Location>\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Grouping.Windows\Samples\2.0\Grouping\Grouping Demo`

 **GridControl** and the **GridDataBoundGrid** do not support Grouping.

### 1.3.3 Summary

#### **GridGroupingControl**

The **GridGroupingcontrol** alone supports calculating summaries. This allows you to display summaries for each group. This enables you to derive additional information from your data such as average, maximum and minimum, summation, count, and so on.

The engine supports summaries that operate on vectors such as Distinct Count, Median, 25% and 75% Quartile. You can also easily add custom summaries.



**Note:** For more details, refer to the following section:

[Summaries](#)

#### **Sample**

A sample of this feature is available in the following location:

`<Install Location>\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Grouping.Windows\Samples\2.0\Calculate Summary\Summary Tutorial`

 **GridControl** and the **GridDataBoundGrid** do not support Summary.

### 1.3.4 Sorting

#### **GridGroupingControl**

The **GridGroupingcontrol** provides in-built support for sorting. This allows you to sort the table data against one or more columns. This provides support to sort unlimited number of columns. While sorting, the grid will rearrange the data to match with the current sort criteria. This provides support for multi-column sorting.



**Note:** For more details, refer to the following section:

[Sorting](#)

#### **Sample**

A sample of this feature is available in the following location:

*<Install Location>\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Grouping.Windows\Samples\2.0\Sort\Sort Method Demo*

### **GridDataBoundGrid**

**GridDataBoundGrid** allows you to arrange items in sequence, in different sets or in both. The following is the list of sorting behavior options that are available in **GridDataBoundGrid**:

- **SingleClick**: Sort column on single click once.
- **DoubleClick**: Sort column on double-click.
- **None**: Sorting is disabled.



**Note:** For more details, refer to the following section:

[Sorting By Display Member](#)

### **Sample**

A sample of this feature is available in the following location:

*<Install Location>\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Windows\Samples\2.0\Data Bound\Sort By DisplayMember Demo*

### **GridControl**

In **GridControl**, sorting is not directly supported. Sorting can be performed through customizing the *GridSortColumnHeaderCellModel*.

### **Sample**

The sample in the following location illustrates how to implement header-click sorting for the **GridControl** based grids.

*<Install Location>\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Windows\Samples\2.0\MS Excel-Style Features\Grid Control Sort Demo*

#### **1.3.4.1 Sort Icon Placement**

This feature is used to place the sort icon in different positions of the column header cell of the grid. The default position of the sort icon is to the right.

The position options include:

- Right
- Top
- Left

#### **Properties**

Property	Description	Type	Data Type
----------	-------------	------	-----------

SortIconPlacement.Right	Gets or sets the icon placement to the right side of the column header cell.	SortIconPlacement	enumeration
SortIconPlacement.Top	Gets or sets the icon placement to the top side of the column header cell.	SortIconPlacement	enumeration
SortIconPlacement.Left	Gets or sets the icon placement to the left side of the column header cell.	SortIconPlacement	enumeration

#### 1.3.4.1.1 Sample Link

{Installed  
Path}\Syncfusion\EssentialStudio\{Version}\Windows\Grid.Grouping.Windows\Samples\2.0\Sorting\Sorting Demo

#### 1.3.4.1.2 Adding Sort Icon Placement to an Application

To enable this feature, use the following code:

[C#]

```
this.gridGroupingControl1.SortIconPlacement = SortIconPlacement.Top;
```

[VB]

```
Me.gridGroupingControl1.SortIconPlacement = SortIconPlacement.Top
```

	CustomerID	CompanyName	ContactName
WOLZA	Wolski Zajazd	Zbyszek Piestrzeniewicz	
WILMK	Wilman Kala	Matti Karttunen	
WHITC	White Clover Markets	Karl Jablonski	
WELLI	Wellington Importadora	Paula Parente	
WARTH	Wartian Herkku	Pirkko Koskitalo	
VINET	Vins et alcools Chevalier	Paul Henriot	

Figure 2: Sort Icon at Top of Header Cell

### 1.3.5 Excel Export

The Essential Grid control has built-in support for Excel. You can download the data from the Grid control or Grid Data Bound Grid or Grouping Grid control into an Excel spreadsheet for offline verification and computation.

#### **GridGroupingControl**

This control automatically copies the Grid's styles, formats, groups, summary rows and expression fields to Excel. This enables you to export the grid with or without nested tables. It provides support for exporting grid with four different views. They are:

- Default
- Visible
- RowHeader
- ColumnHeader.



**Note:** For more details, refer to the following section:

[Exporting Grid Grouping Control To Excel](#)

#### **Sample**

A sample of this feature is available in the following location:

`<Install Location>\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Grouping.Windows\Samples\2.0\Export\MS Excel Export Demo`

#### **GridControl and GridDataBoundGrid**

The `GridExcelConverter` class enables you to exporting data from a Grid control or Grid Data Bound Grid into Excel. This control automatically copies the Grid's styles and formats to Excel.



**Note:** For more details, refer to the following section:

[Exporting the Grid Control or Grid Data Bound Grid To Excel](#)

#### **Sample**

A sample of this feature is available in the following location:

For GDBG:

<Install Location>\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Windows\Samples\2.0\Export\DBG XLS Export Demo

For GC:

<Install Location>\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Windows\Samples\2.0\Export\GC XLS Export Demo

### 1.3.6 Filtering

#### GridGroupingControl

This control enables you to specify the logical condition for filtering. Similar to an Expression Field, this control also provides you the option of typing an expression or entering a condition using an editor dialog for filtering.

#### Filterbar

Grouping Grid provides in-built support for displaying a *Filter Bar* across the columns. It can be used to filter and clear filter at run time. This is user interactive and has more advantages. One of the main reasons for its wide usage is this can display various filter options for the columns. You can also add your own filter criteria.

#### Dynamic filter

Dynamic Filter serves as good replacement for the Filter Bar. This provides advanced filtering capabilities. This is built on the foundation of the regular filter bar with added provisions to support dynamic filtering. The dynamic filter can be used with Nested Tables and Nested Groups.



**Note:** For more details, refer to the following section:

#### Record Filtering

#### Sample

A sample of this feature is available in the following location:

<Install Location>\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Grouping.Windows\Samples\2.0\Filters and Expressions\

#### GridDataBoundGrid

You can enable filtering for the grid based on GridDataBoundGrid, by adding a row of drop-down cells at the top of a simple (non-hierarchical) Grid Data Bound Grid. This allows you to filter match values from the drop-down. This control also supports filter by DisplayMember.



**Note:** For more details, refer to the following section:

[Filtering a Grid Data Bound Grid](#)

Sample

A sample of this feature is available in the following location:

<Install Location>\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Windows\Samples\2.0\Data Bound\Filter By DisplayMember Demo

### 1.3.6.1 Filter for Specific Columns

When a filter is wired to a grid, it is wired to the entire grid. This makes it difficult to use different filters (e.g., dynamic filter, Excel-style filter, and filter by display member) in a single grid.

Here is a short description of each filter:

- **Dynamic Filter**—Filters the content using a list of comparison operators.
- **Ordinary Filter**—Filters the content based on the selected text and index.
- **Excel-Style Filter**—Filters the content based on multiple values and can sort the results, similar to Microsoft Excel filtering.
- **Filter by Display Member**—Filters the content by displaying the member instead of the value member of a combo-box column.

This feature enhances the use of different filters within a single grid. The filters are wired to each column by changing the cell type of the corresponding column, which enables users to apply many filters.

A filter can be applied to an individual column by setting the **AllowIndividualColumnWiring** property of the filter to **true**.

#### Properties

Property	Description	Type	Data Type
AllowIndividualColumnWiring	Gets or sets whether the filters can be wired to an individual column.	Bool	Bool

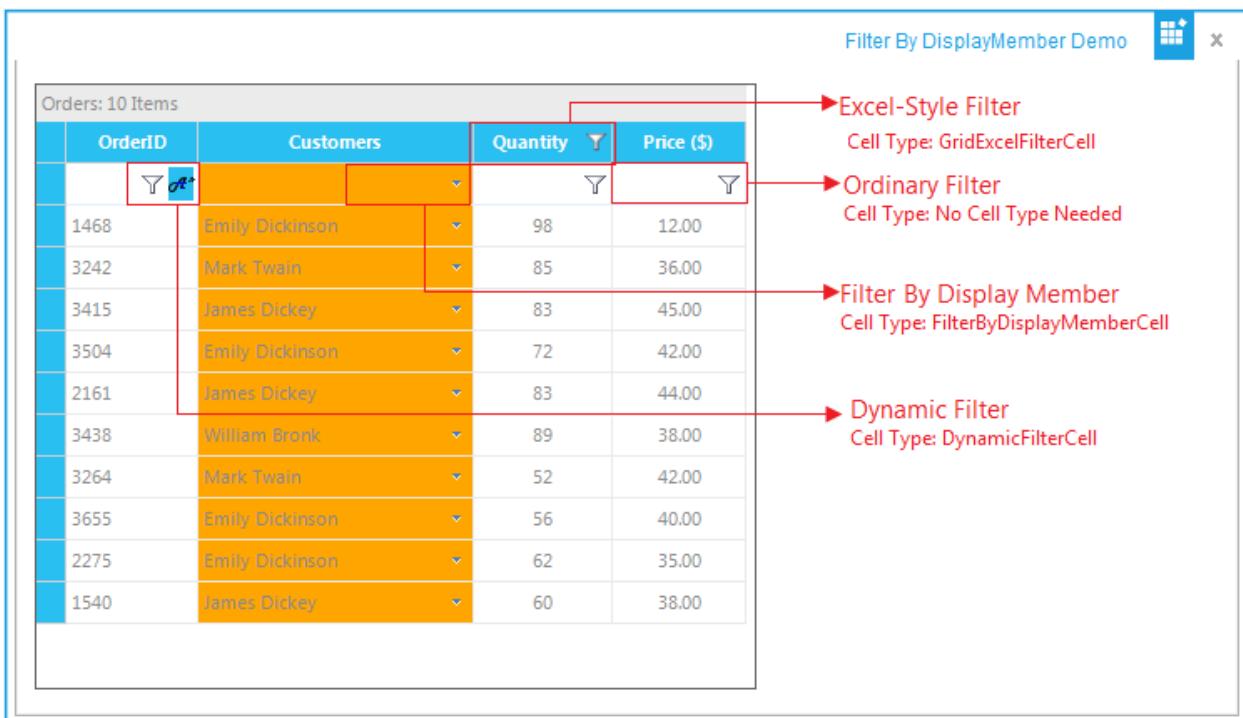


Figure 3: Filters on Specific Columns

### Sample Link

A sample is available in the following location:

[Installed

Drive]\AppData\Local\Syncfusion\EssentialStudio\[Version]\Windows\Grid.Grouping.Windows\Samples\2.0\Filters and Expressions\Filter By DisplayMember Demo

### Enable Filters on Desired Columns

The following code is used to enable filters in specific columns:

[C#]

```
GridDynamicFilter filter = new GridDynamicFilter();
filter.AllowIndividualColumnWiring = true;
filter.WireGrid(gridGroupingControl1);
```

[VB]

```
Dim filter As New GridDynamicFilter()
filter.AllowIndividualColumnWiring = true;
filter.WireGrid(gridGroupingControl1);
```

Dynamic filter is wired to column 0 by using the following code:

[C#]

```
this.gridGroupingControl1.TableDescriptor.Columns[0].Appearance.FilterBarCell.CellType = "DynamicFilterCell";
```

[VB]

```
Me.gridGroupingControl1.TableDescriptor.Columns[0].Appearance.FilterBarCell.CellType = "DynamicFilterCell";
```

Other filters can be wired to the grids column by using the previous code.

## 1.4 Documentation

Syncfusion includes the following documentation segments to provide all necessary information for using Essential Grid controls in an efficient manner.

Type of Documentation	Location
Readme	[drive:]\\Program Files\\Syncfusion\\Essential Studio\\x.x.x\\Infrastructure\\Data\\Release Notes\\readme.htm
Release Notes	[drive:]\\Program Files\\Syncfusion\\Essential Studio\\x.x.x\\Infrastructure\\Data\\Release Notes\\Release Notes.htm
User Guide (this document)	<b>Online</b> <a href="http://help.syncfusion.com/resources">http://help.syncfusion.com/resources</a> (Navigate to the Grid for Windows Forms User Guide.)  <b>Note:</b> Click Download as PDF to access a PDF version.  <b>Installed Documentation</b> Dashboard -> Documentation -> Installed Documentation.
Class Reference	<b>Online</b> <a href="http://help.syncfusion.com/resources">http://help.syncfusion.com/resources</a> (Navigate to the Windows Forms User Guide. Select Grid in the second text box, and then click the Class Reference link found in the upper right section of the page.)  <b>Installed Documentation</b>

	Dashboard -> Documentation -> Installed Documentation.
--	--

## **2 Installation and Deployment**

---

This section covers information on the install location, samples, licensing, patches update, and updation of the recent version of Essential Studio. It comprises the following subsections:

### **2.1 Installation**

For step-by-step installation procedure for the installation of Essential Studio, refer to the **Installation** topic under **Installation and Deployment** in the **Common UG**.

#### **See Also**

For licensing, patches, and information on adding or removing selective components, refer the following topics in **Common UG** under **Installation and Deployment**.

- Licensing
- Patches
- Add/Remove Components

### **2.2 Sample and Location**

This section covers the location of the installed samples and describes the procedure to run the samples through the sample browser. It also lists the location of source code.

#### **Sample Installation Location**

The **Grid Windows Forms** samples are installed at the following location locally on the disk:

**C:\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Windows\Samples\2.0**

The **Grid Grouping Windows Forms** samples are installed at the following location locally on the disk:

**C:\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Grouping.Windows\Samples\2.0**

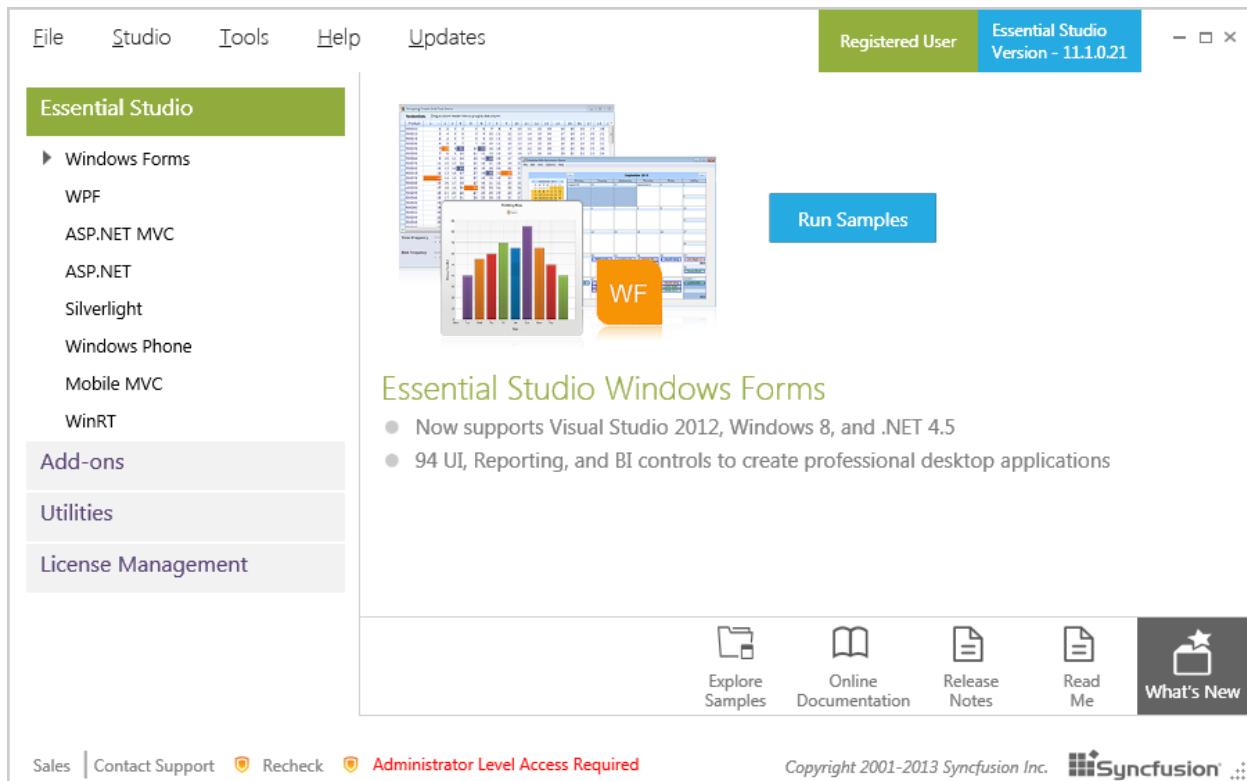
The **GridDataBound** control's samples are installed at the following location locally on the disk:

**C:\Syncfusion\EssentialStudio\Version Number\Windows\GridDataBound.Windows\Samples\2.0**

## **Viewing Samples**

To view the samples, follow the steps below:

1. Click **Start > All Programs > Syncfusion > Essential Studio <version number> > Dashboard.**



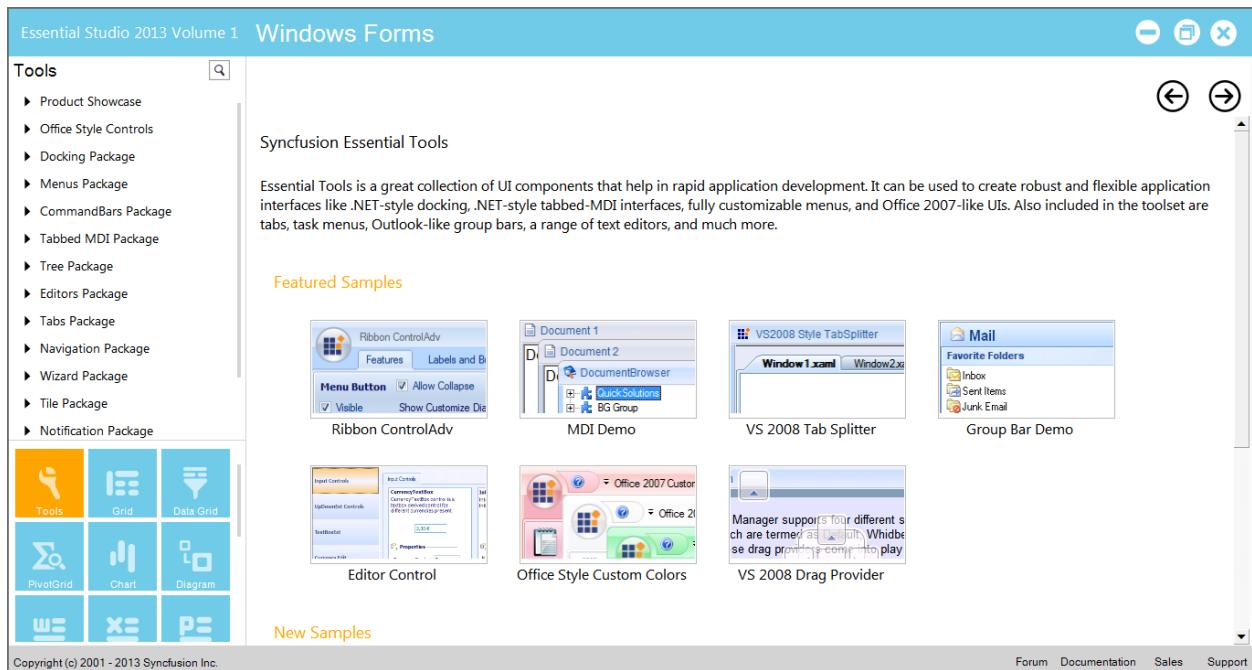
*Figure 4: Syncfusion Essential Studio Dashboard*

2. In the Dashboard window, click **Run Samples** for Windows Forms under UI Edition. The UI Windows Form Sample Browser window is displayed.



**Note:** You can view the samples in any of the following three ways:

- **Run Samples** - Click to view the locally installed samples.
- **Online Samples** - Click to view online samples.
- **Explore Samples** - Explore BI Web samples on disk.



*Figure 5: User Interface Edition Windows Forms Sample Browser*

3. To view Grid samples, click **Grid** at the bottom-left pane.

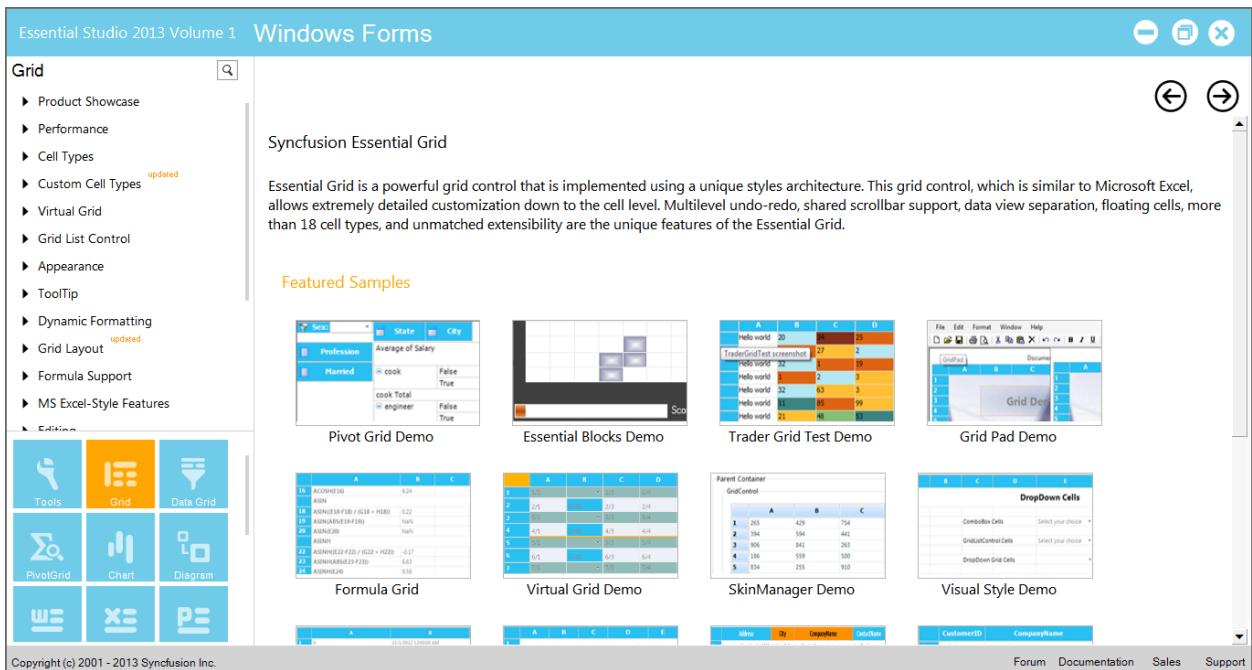


Figure 6: Grid Samples Selected Under Contents Tab

- To view Grid grouping samples, click **Data Grid** at the bottom-left pane.

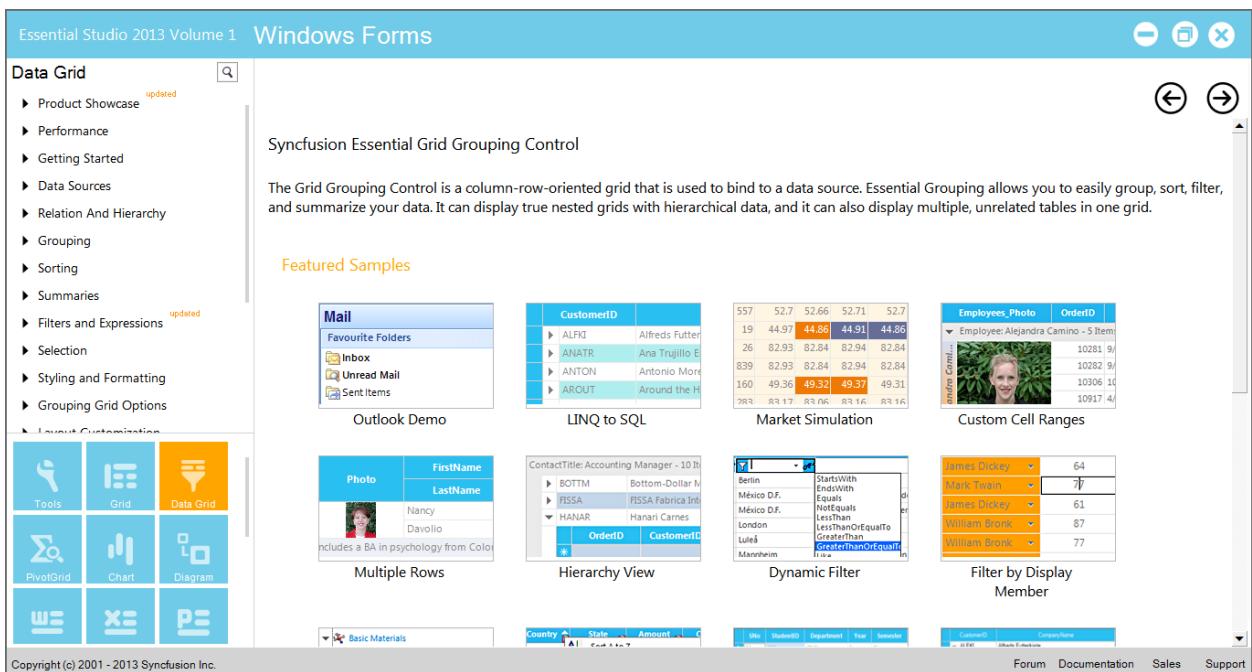


Figure 7: Grid Grouping Samples Selected Under Contents Tab

- To view Grid data bound samples, click **GridDataBound** from the bottom-left pane.

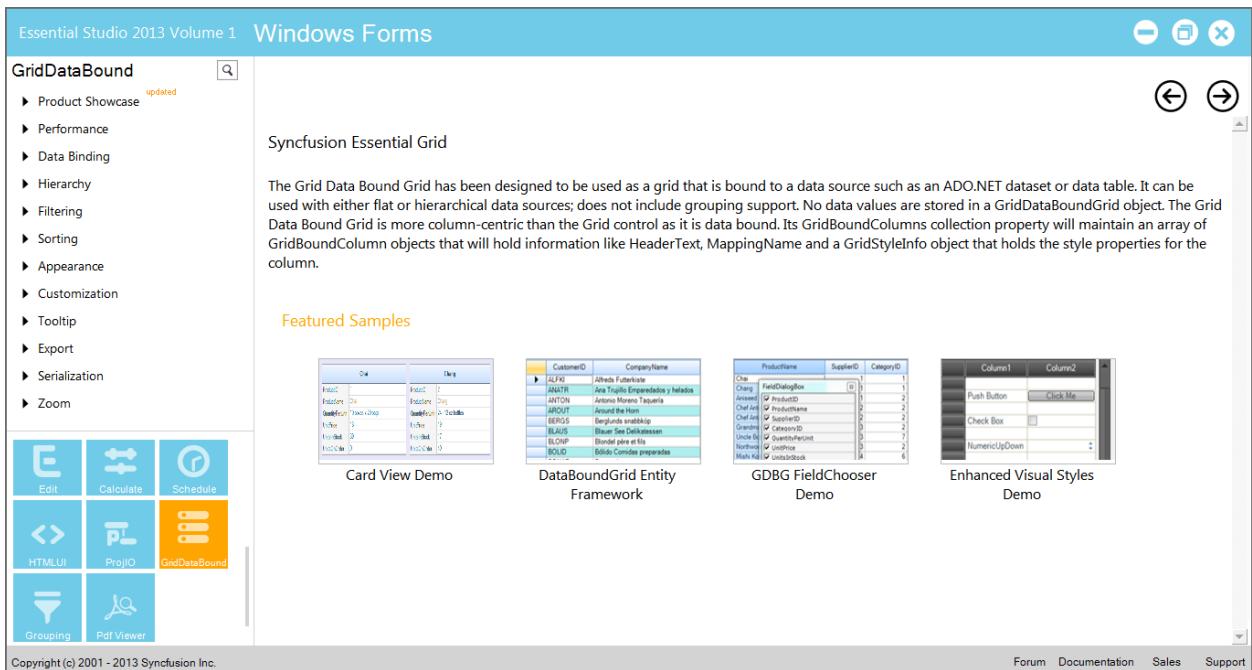


Figure 8: Grid Data Bound Grid

6. Select any sample and browse through the features.

### Source Code Location

The default location of the **Grid Windows** source code is:

**[System Drive]:\Program Files\Syncfusion\Essential Studio\Version Number\Windows\Grid.Windows\Src**

The default location of the **Grid Grouping Windows** source code is:

**[System Drive]:\Program Files\Syncfusion\Essential Studio\Version Number\Windows\Grid.Grouping.Windows\Src**

## 2.3 Deployment Requirements

### Toolbox Entries

Essential Grid places six controls into your Visual Studio .NET toolbox from where you can drag any of these controls onto a form and start working with them.

- Grid Control
- Grid Data Bound Grid
- Grid Grouping Control
- Grid List Control
- Grid Record Navigation Control
- Grid Aware Text Box

For details, see [List of Controls](#).

## DLLs List

While deploying an application that references a Syncfusion Essential Grid assembly, the following dependencies also need to be included in the distribution.

### **Syncfusion.Grid.Base.dll**

- Syncfusion.Core.dll
- Syncfusion.Shared.Base.dll

### **Syncfusion.Grid.Windows.dll**

- Syncfusion.Core.dll
- Syncfusion.Grid.Base.dll
- Syncfusion.Shared.Base.dll
- Syncfusion.Shared.Windows.dll

### **Syncfusion.Grid.Grouping.Base.dll**

- Syncfusion.Core.dll
- Syncfusion.Shared.Base.dll

### **Syncfusion.Grid.Grouping.Windows.dll**

- Syncfusion.Core.dll
- Syncfusion.Grid.Grouping.Base.dll
- Syncfusion.Grid.Base.dll
- Syncfusion.Grid.Windows.dll
- Syncfusion.Grouping.Base.dll
- Syncfusion.Grouping.Windows.dll
- Syncfusion.Shared.Base.dll
- Syncfusion.Shared.Windows.dll

### **Syncfusion.Grouping.Base.dll**

- Syncfusion.Core.dll
- Syncfusion.Shared.Base.dll

**Syncfusion.Grouping.Windows.dll**

- Syncfusion.Core.dll
- Syncfusion.Grouping.Base.dll
- Syncfusion.Shared.Base.dll
- Syncfusion.Shared.Windows.

## 3 Getting Started

---

This section comprises the following topics:

### 3.1 Tutorials

#### What You Will Learn

This tutorial will show you how easy it is to get started with Essential Grid. It will give you a basic introduction to the concepts you need to know before getting started with the product, and some tips and ideas on how to implement the Grid into your projects to improve customization and increase efficiency. The lessons in this tutorial are meant to introduce you to Essential Grid with simple step-by-step procedures.

- [Grid Grouping Control Designer](#)

In Lesson 1, you will become familiar with the **Grid Grouping control**, learning how to bind it to an MDB file, and creating a data source from an MDB file. You will also receive information related to the designer and its appearance properties. You will learn about the **ShowGroupDropArea**, **TableOptions**, **Appearance** **TopLevelGroupOptions**, **ChildGroupOptions**, **TableDescriptors** properties as well as saving and restoring look and feel properties. There is also a section on adding summary rows, expression fields, and row filters.

- [Grid Control Designer](#)

In Lesson 2, you'll use our cell-oriented **Grid control** in design mode to set properties on individual cells and ranges of cells.

- [Grid Data Bound Grid Designer](#)

In Lesson 3, you'll receive a background in the basics of **Grid Data Bound Grid** and will also learn how to apply special column formats.

- [Virtual Grid](#)

Lesson 4 will give you information on the importance of the **Virtual Grid**. You will learn about creating the project and external data source, adding and initializing Essential grid, event handlers, saving edited values, type conversions and additional virtual grid properties.

- [Excel Export](#)

This section will step you through the conversion of Grid content to an Excel file, and also discusses the various converter options.

- [Accessibility Information for Grid Windows Forms](#)

This section will detail the criteria for accessibility for Grid Windows Forms

### 3.1.1 Before You Begin

Some of the projects in this tutorial are mainly VS .NET steps, but we have included them to give you complete experience and information.

During the installation of Essential Grid, these controls will be installed in your toolbox within Visual Studio: Grid control, Grid Data Bound Grid, Grid List control, Grid Record Navigation control and Grid Grouping control. The latter control will only be available if you have purchased the Enterprise version of Essential Grid. If these controls are not available in your toolbox, you must add them before proceeding with this tutorial.

Lesson 1 assumes the sample data file, ACC.mdb, that we ship as part of our samples, is on your disk. It is usually found in this folder: **C:\Syncfusion\EssentialStudio\[Version Number]\Windows\Data**

Lesson 2 assumes that you have previously installed the QuickStart samples that are part of the .NET framework installation. In particular, we will use the NorthWind MSDE database that is part of those samples.



Depending on the specifications of your computer (resolution, color, size of monitor), the illustrations and code in this tutorial may vary slightly from what will appear on your screen. Contact technical support if you have any questions.

### 3.1.2 Lesson 1: Grid Grouping Control Designer

Grid Grouping control has strong designer support. You can control all aspects of the grid's appearance through the designer. Additional commands (verbs) will let you save layouts and restore them. You can also use a preview feature that will let you load data into your control, and then further set the Grid Grouping control properties that can be persisted as design time properties.

This section has two major tasks. The first task is to place the Grid Grouping control on the form. The second task is to use the designer to set up data binding to an Access data file. This is strictly a Windows Forms process and really has nothing to do with our Grid Grouping control. You just need to set up a Data Adapter to access the data that is needed for the grid. The data for this is located in the My Documents\Syncfusion\EssentialStudio\VersionNumber\Windows\Data\NWind.mdb (depending upon your installation folder). You must use ADO.NET OLE DB support to access this data.

In this lesson, you will learn about:

### 3.1.2.1 Binding to an MDB File By Using VS 2003

The steps in this lesson are for use with Visual Studio 2003.

1. Open the **Data** section of your toolbox and drag an **OleDbDataAdapter** onto your form. This will open the **Data Adapter Configuration Wizard**.



*Figure 9: Data Adapter Configuration Wizard*

2. You can use this wizard to create a connection to the NWIND.mdb file. Click **Next** to continue.



*Figure 10: Setting up the Data Connection*

3. Click New Connection. The Data Link Properties dialog box will be displayed. In the Provider tab, select Microsoft Jet 4.0 OLE DB Provider option as shown in the following screen shot.

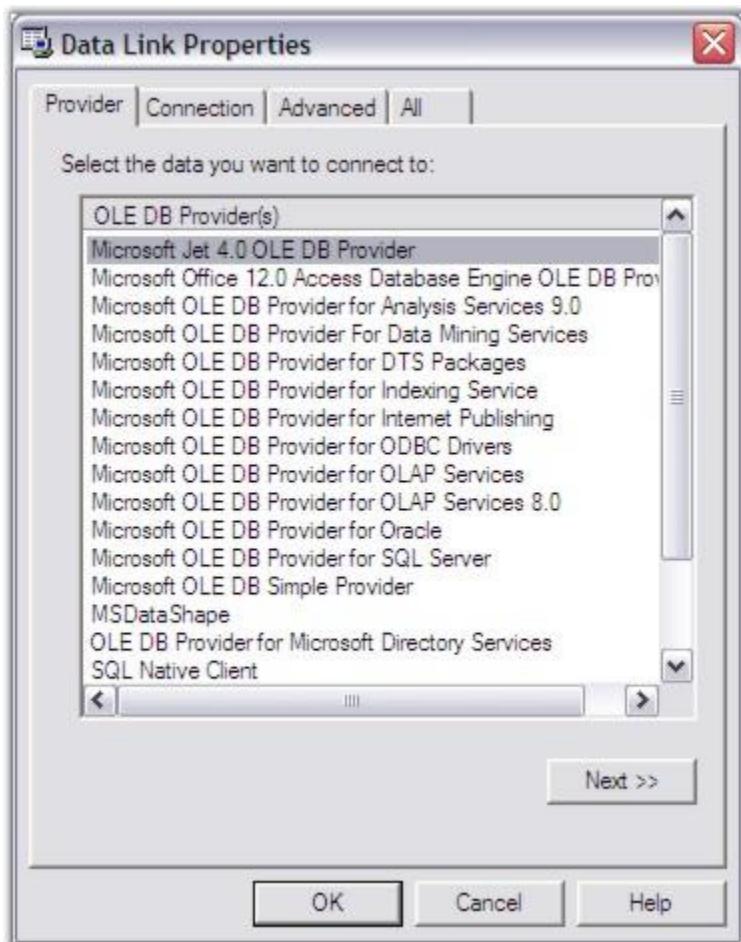


Figure 11: Choose the Access Data Provider, Jet 4.0

4. Click **Next**. The **Connection** tab will be displayed as follows.



Figure 12: Click the Browse button to select the File

5. Click the **Browse** button to browse and locate an mdb file. For example, 'NWIND.mdb' is selected. This file is found in the following path: **C:\Syncfusion\EssentialStudio\Version Number\Windows\Data** (this path will vary according to your installation location). Then click **OK**.

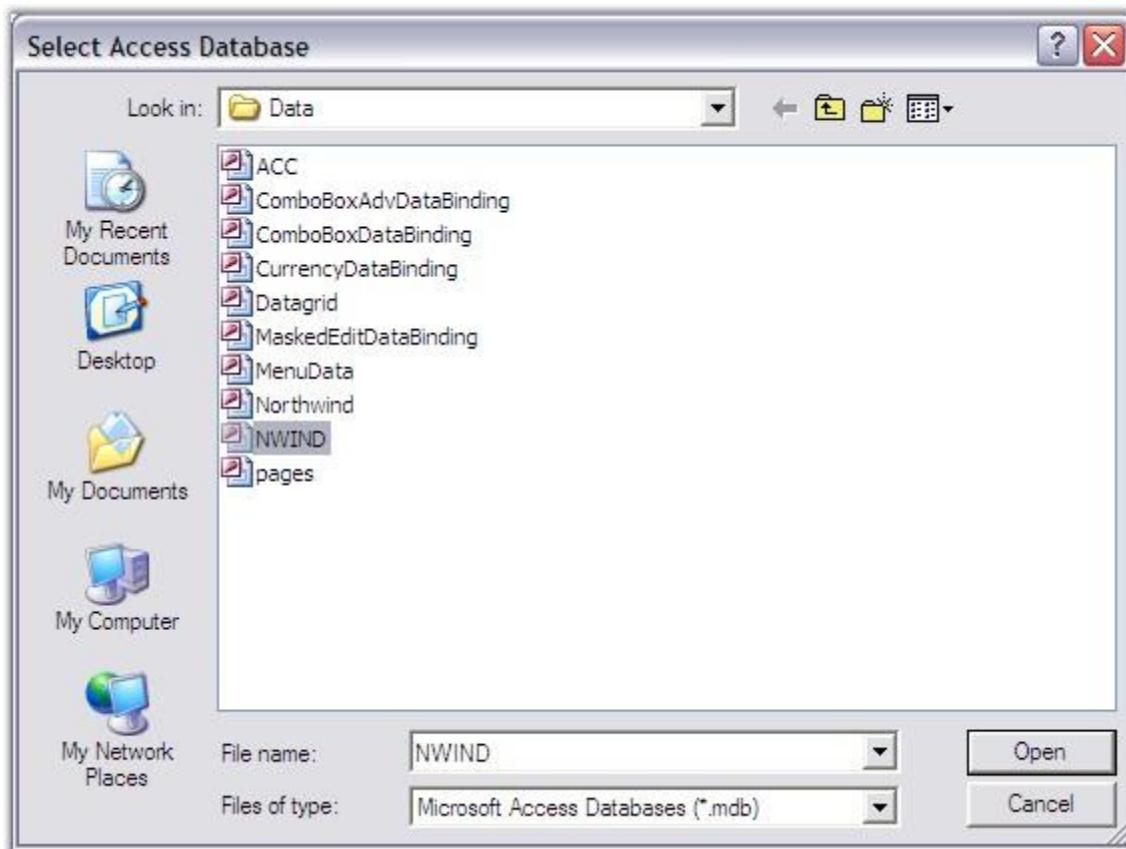


Figure 13: Locating the Access file, NWIND.MDB

6. Click **Next**.

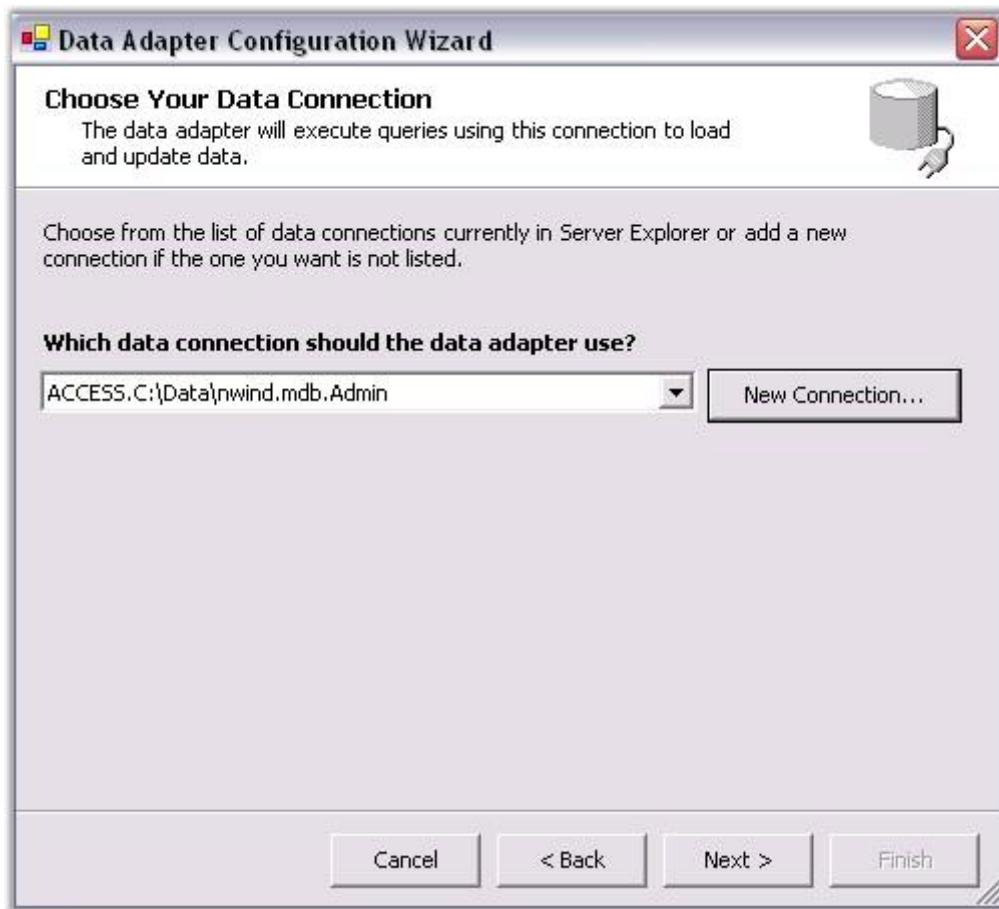
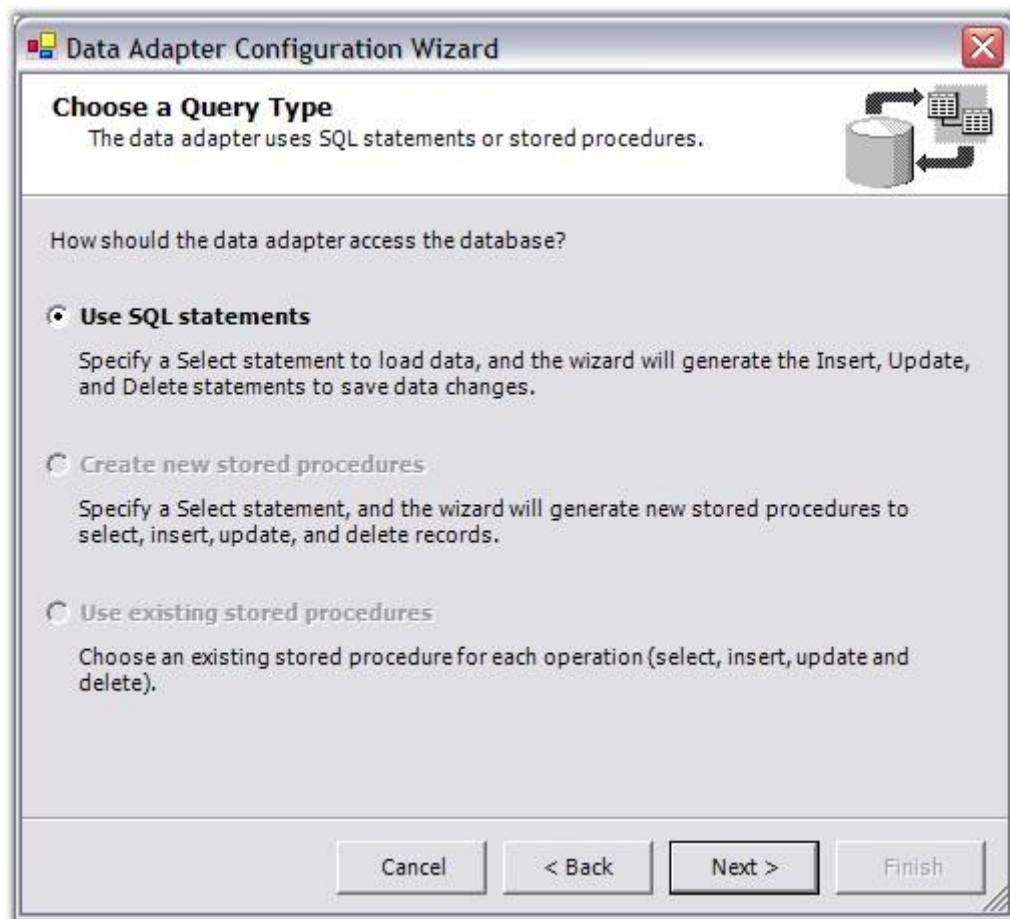


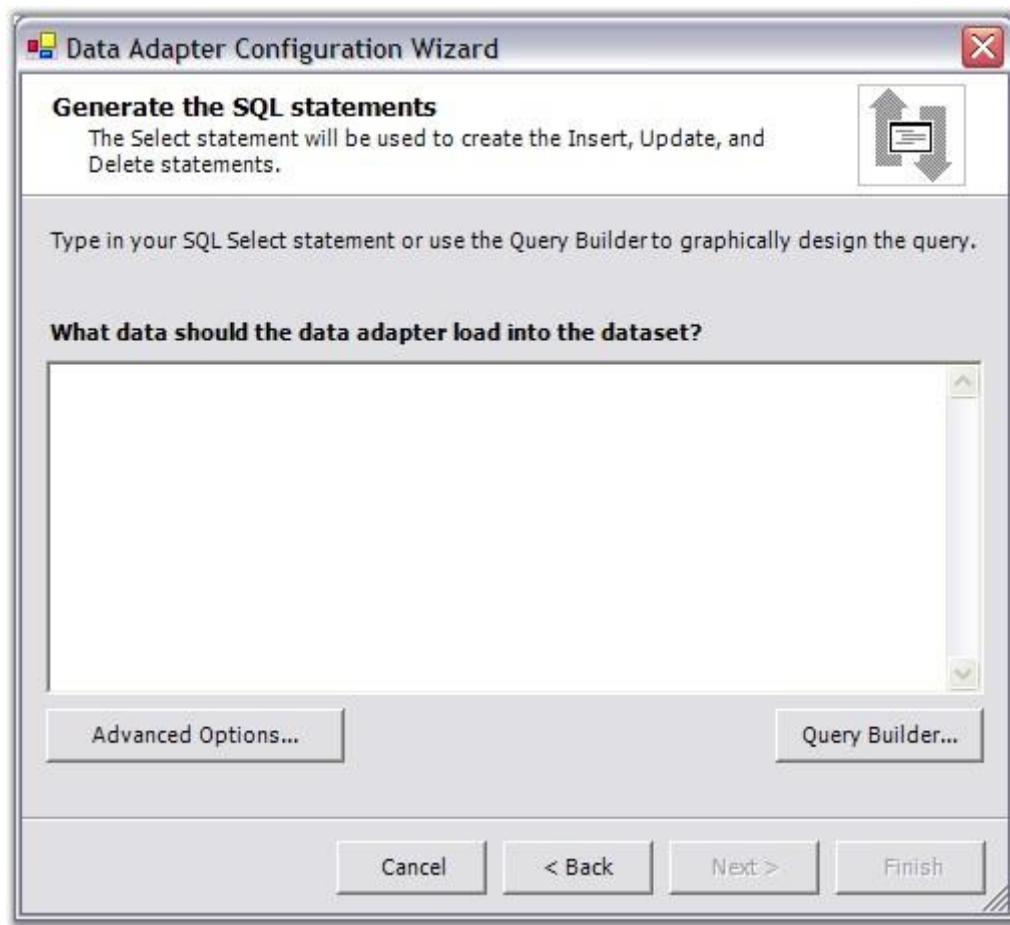
Figure 14: Setting the MDB file as the Data Connection

7. Select **Use SQL statements** as your query type, and then click **Next**.



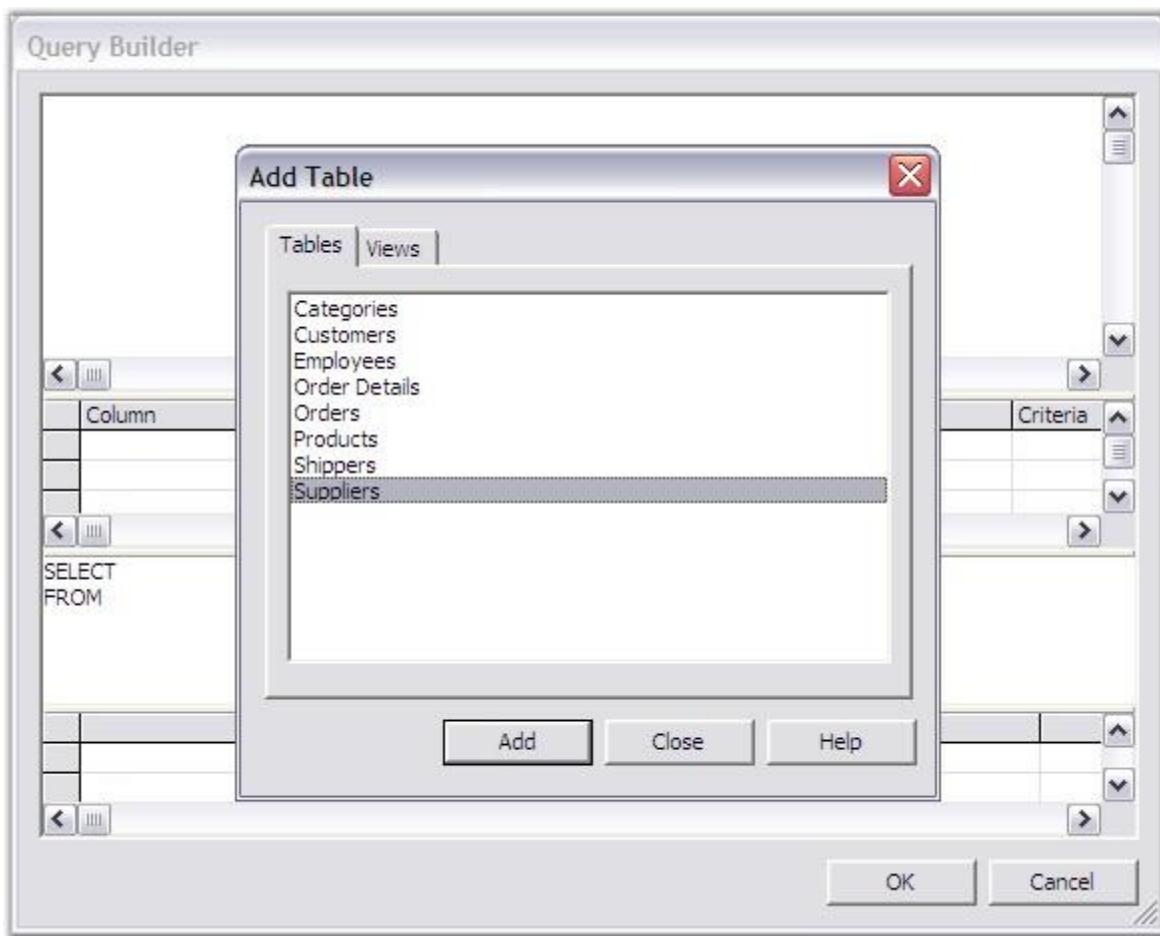
*Figure 15: Select to Use SQL Statements*

8. To generate the SQL statement, click **Query Builder**.



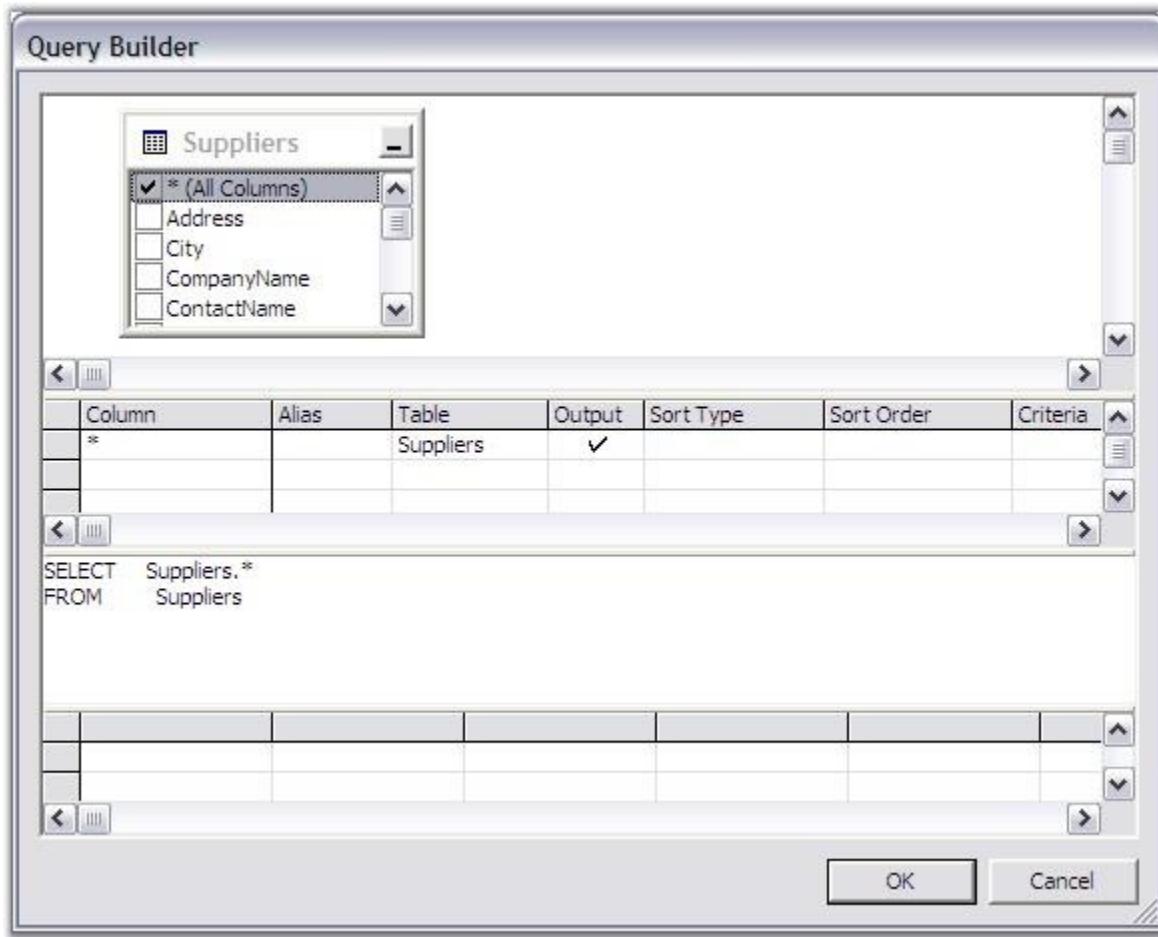
*Figure 16: Select the Query Builder*

9. In the **Add Table** dialog box, select the 'Suppliers' table, click **Add**, and then click **Close**.



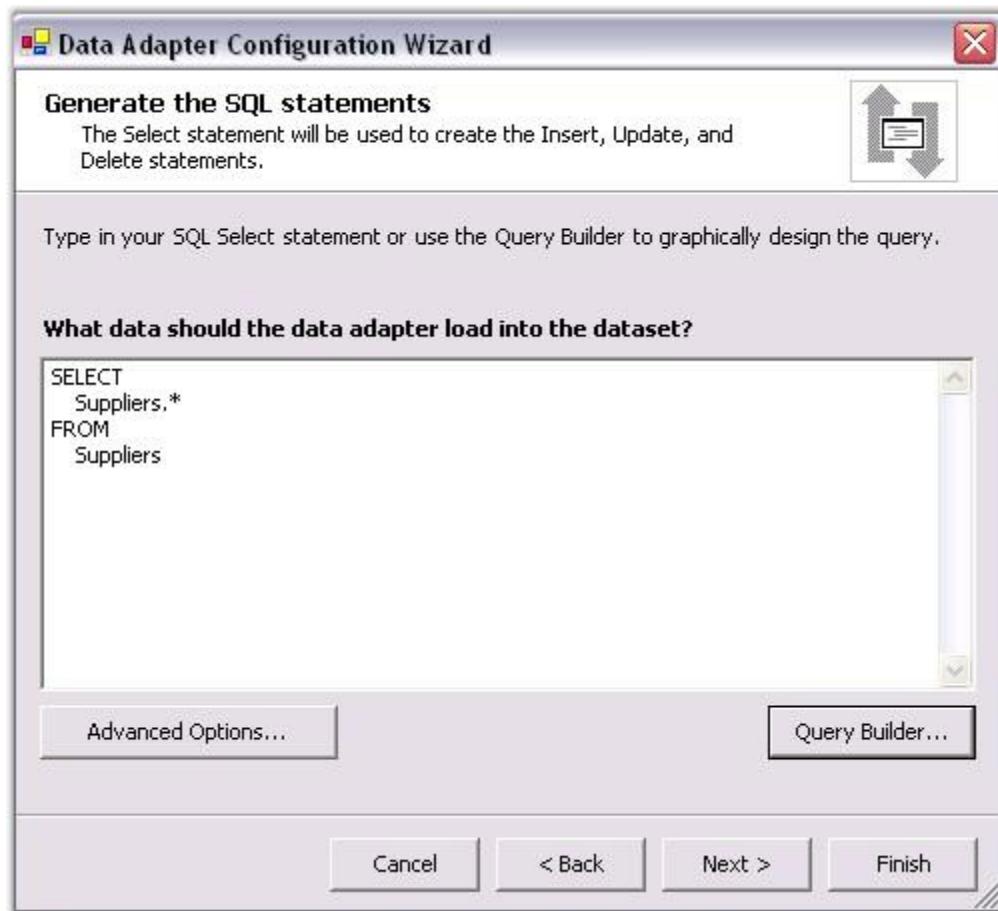
*Figure 17: Select the Suppliers Table*

10. In this Query Builder window, select **All Columns**. Then click **OK**.



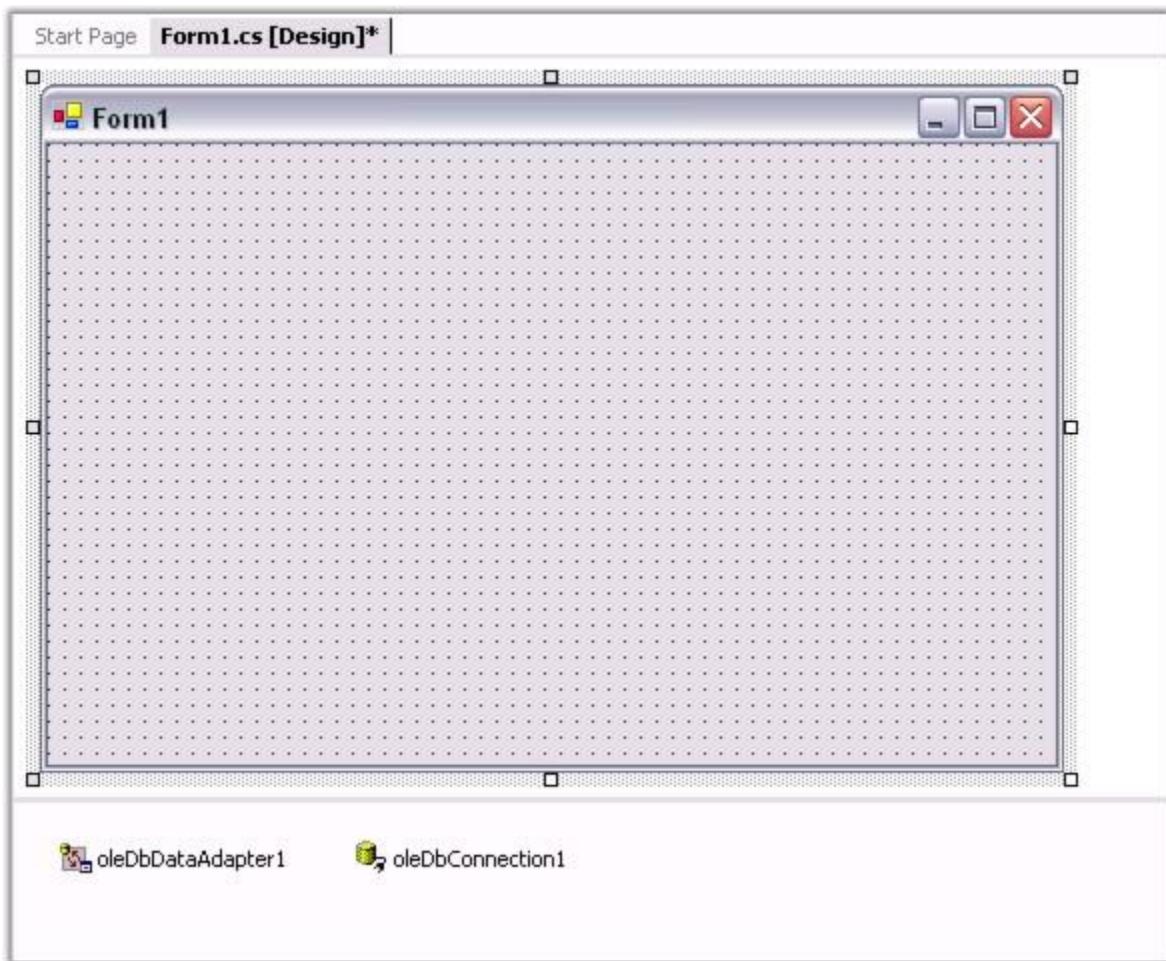
*Figure 18: Select the Fields for your Query*

11. Click Next to confirm the Query that you have selected.



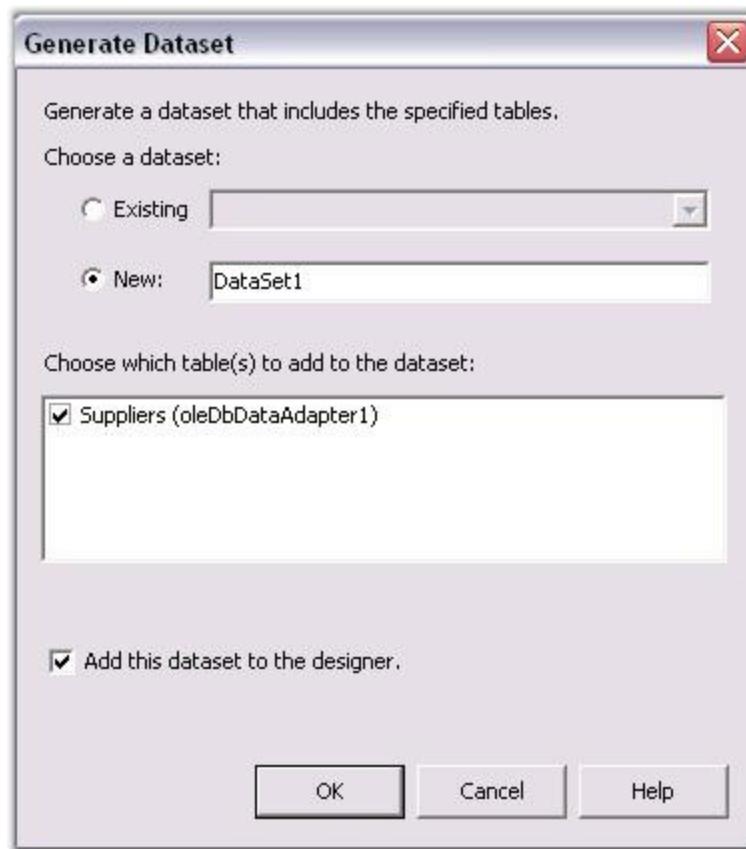
*Figure 19: Confirm the Query*

12. Click **Finish**. Your design surface will look similar to this.



*Figure 20: Adapter and Connection*

13. Next you will need to generate a data set from the OleDbDataAdapter. Right-click the **oleDbDataAdapter1** under the design surface and select **Generate DataSet**. The **Generate Dataset** dialog box will be displayed.



*Figure 21: Generating a Dataset*

14. Click **OK** to add a **DataSet11** object next to the **oleDbConnection1** under the design surface.
15. From the toolbox, drag the **Grid Grouping control** to your form. Size and position it. Also, set the **DataSource** property to **dataSet11.Suppliers Data Table** as shown in the following screen shot.

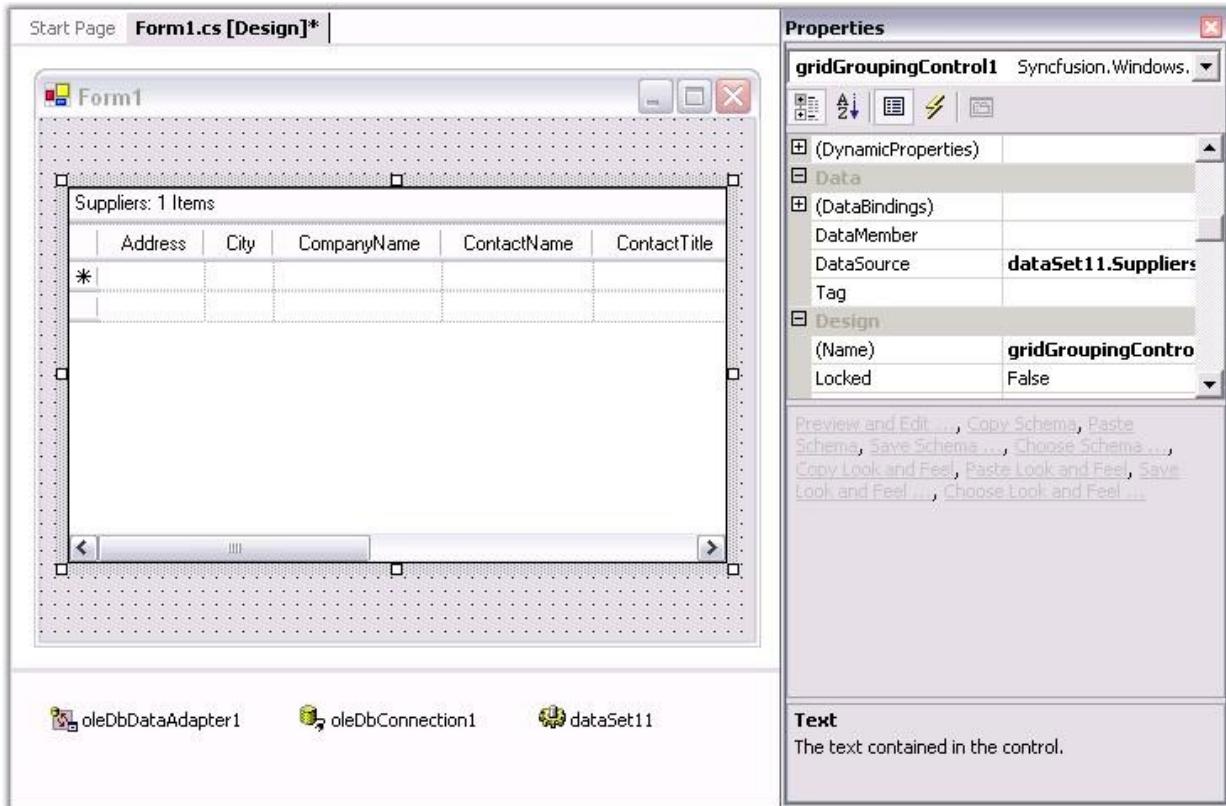


Figure 22: Designer with Grid Grouping control and setting the *DataSource* Property

16. Double-click the form on the design surface to add a Load event handler. In this handler, add the following code.

[C#]

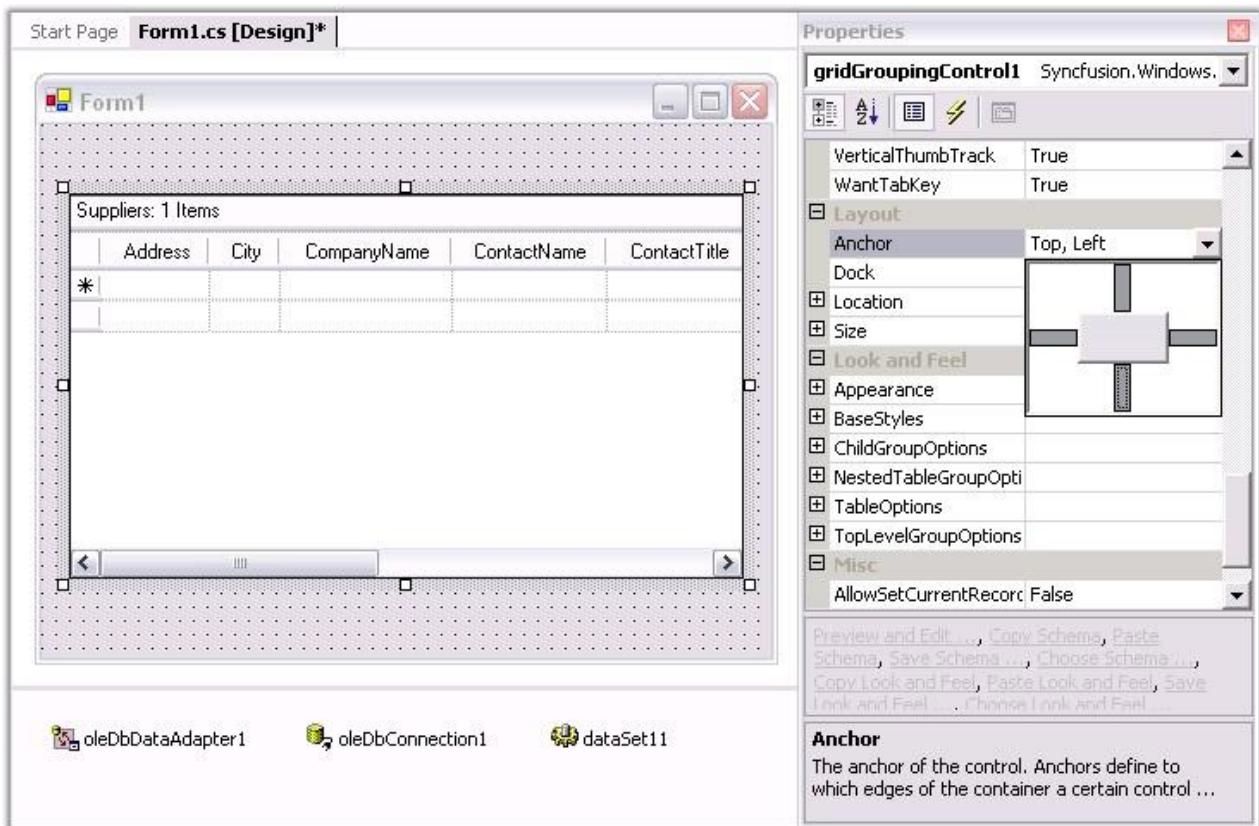
```
this.oleDbTypeAdapter.Fill(this.dataSet11);
```

[VB .NET]

```
Me.oleDbTypeAdapter.Fill(Me.dataSet11)
```

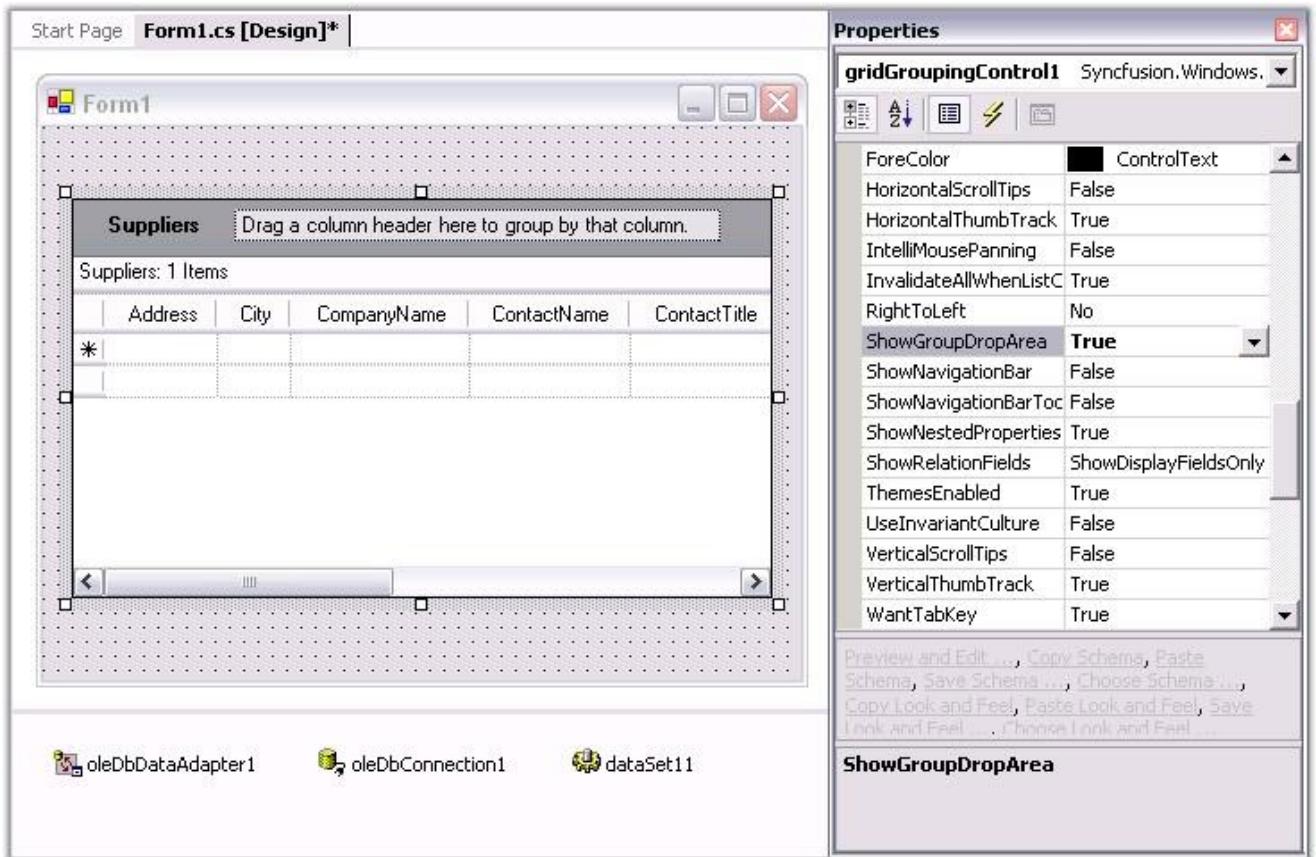
The preceding code is used to load the data from the Data Table. Without this code, you will see an empty Grid Grouping control at run time.

17. Finally, set the **Anchor** property of the Grid Grouping control to *All*, so that the Grid Grouping control can be easily sized with the form. This is depicted in the following screen shot.



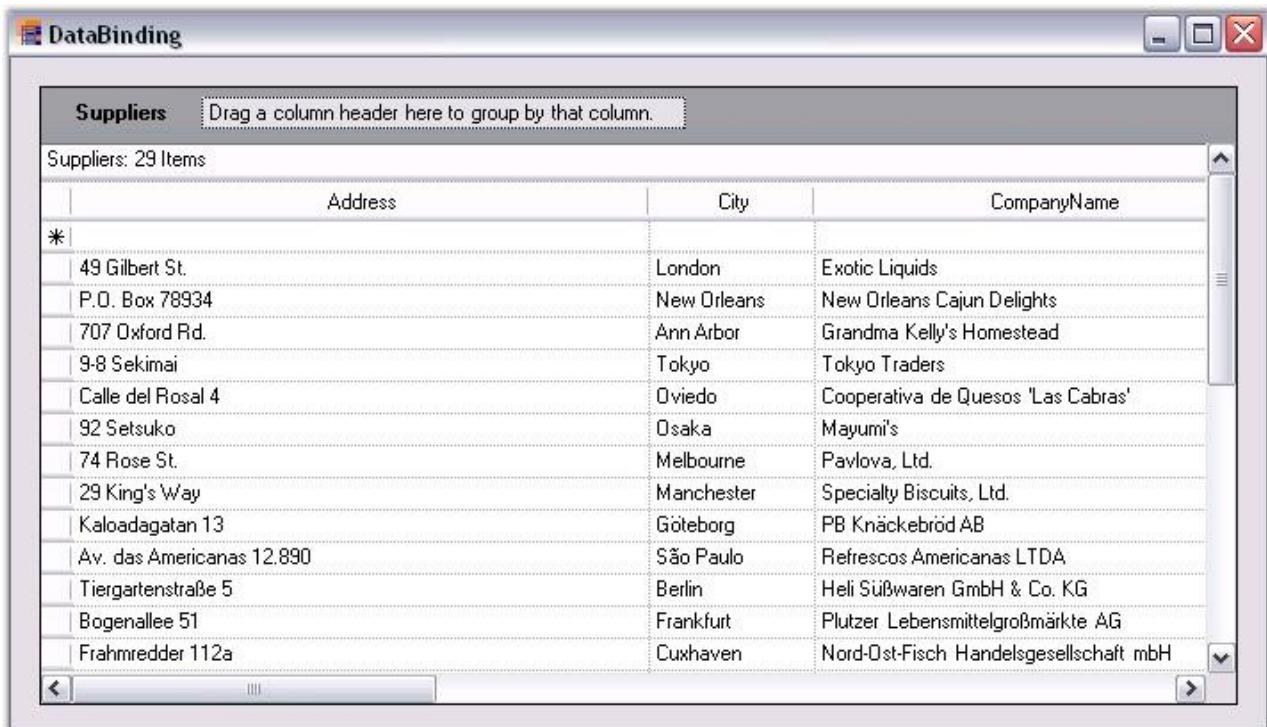
*Figure 23: Setting the Anchor Property*

18. To allow grouping at run time, the Grid Grouping control displays a drop panel onto which the user can drag columns to be grouped. To display this drop panel, you need to set the **ShowGroupDropArea** property to *true* as shown in the following screen shot.



*Figure 24: Adding the Group Drop Area*

Run the project. You will see the basic Grid Grouping control with the data as follows.



*Figure 25: Application showing Grid Grouping control with Data*

19. To group by CompanyName, click on the CompanyName column header and drag it to the drop area as illustrated in the following screen shot.

The screenshot shows a Windows application window titled "DataBinding". Inside, there's a grid control with the following columns: "SupplierID", "CompanyName", "Address", "City", and "Country". The data is grouped by "CompanyName". A tooltip above the grid says "Drag a column header up by that column." The first group, "Exotic Liquids", contains 4 items. The second group, "New Orleans Cajun Delights", contains 3 items. The third group, "Grandma Kelly's Homestead", contains 1 item. The fourth group, "Tokyo Traders", contains 2 items. The fifth group, "Cooperativa de Quesos 'Las Cabras'", contains 1 item. The sixth group, "Mayumi's", contains 1 item. The seventh group, "Pavlova, Ltd.", contains 1 item. The eighth group, "Specialty Biscuits, Ltd.", contains 1 item. The ninth group, "PB Knäckebröd AB", contains 1 item. The tenth group, "Refrescos Americanas LTDA", contains 1 item. The eleventh group, "Heli Süßwaren GmbH & Co. KG", contains 1 item. The twelfth group, "Plutzer Lebensmittelgroßmärkte AG", contains 1 item. The thirteenth group, "Nord-Ost-Fisch Handelsgesellschaft mbH", contains 1 item.

SupplierID	CompanyName	Address	City	Country
1	Exotic Liquids	49 Gilbert St.	London	UK
2	New Orleans Cajun Delights	P.O. Box 78934	New Orleans	USA
3	Grandma Kelly's Homestead	707 Oxford Rd.	Ann Arbor	USA
4	Tokyo Traders	9-8 Sekimai	Tokyo	Japan
5	Cooperativa de Quesos 'Las Cabras'	Calle del Rosal 4	Oviedo	Spain
6	Mayumi's	92 Setsuko	Osaka	Japan
7	Pavlova, Ltd.	74 Rose St.	Melbourne	Australia
8	Specialty Biscuits, Ltd.	29 King's Way	Manchester	UK
9	PB Knäckebröd AB	Kaloadagatan 13	Göteborg	Sweden
10	Refrescos Americanas LTDA	Av. das Americanas 12.890	São Paulo	Brazil
11	Heli Süßwaren GmbH & Co. KG	Tiergartenstraße 5	Berlin	Germany
12	Plutzer Lebensmittelgroßmärkte AG	Bogenallee 51	Frankfurt	Germany
13	Nord-Ost-Fisch Handelsgesellschaft mbH	Frahmredder 112a	Cuxhaven	Germany

Figure 26: Grouping by the CompanyName Column

**Note:** Each set of grouped values has its own "Caption" row and its own "AddNew" row (\*). Each group has its own PlusMinus cell that will let you expand/collapse the group.

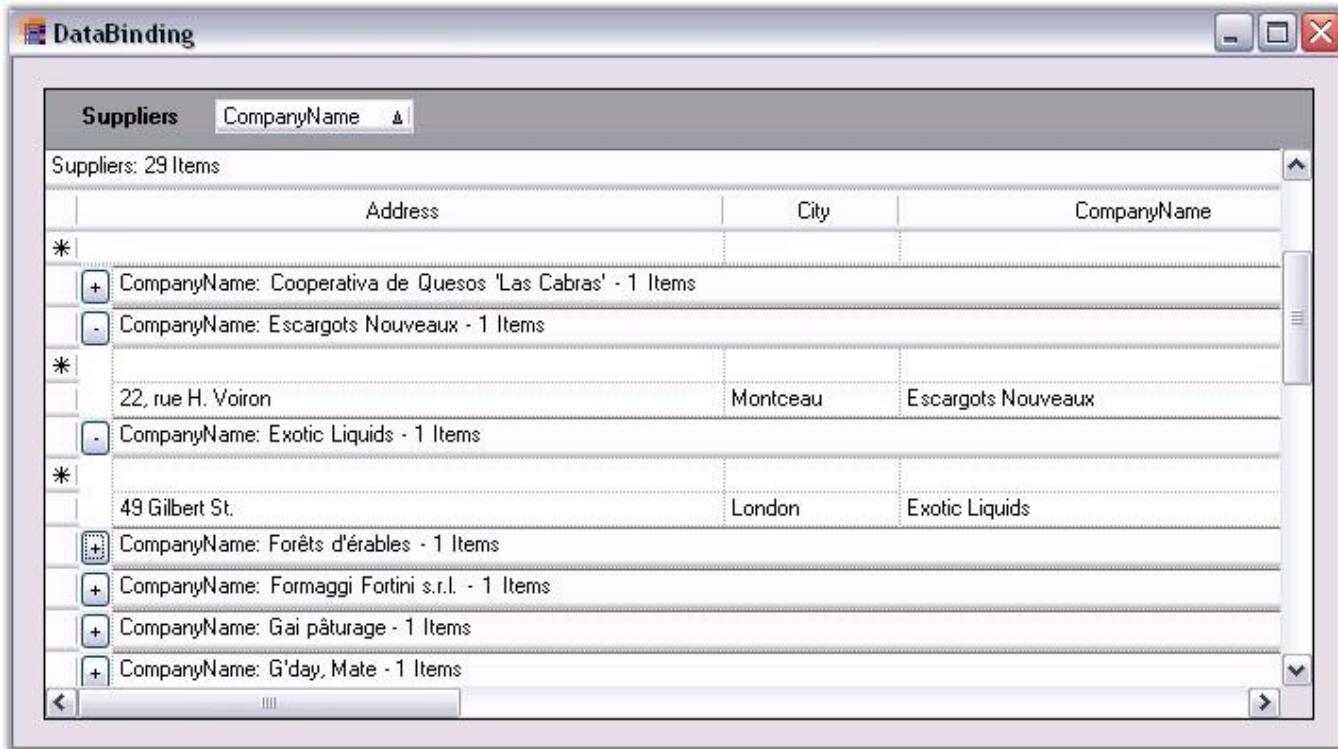
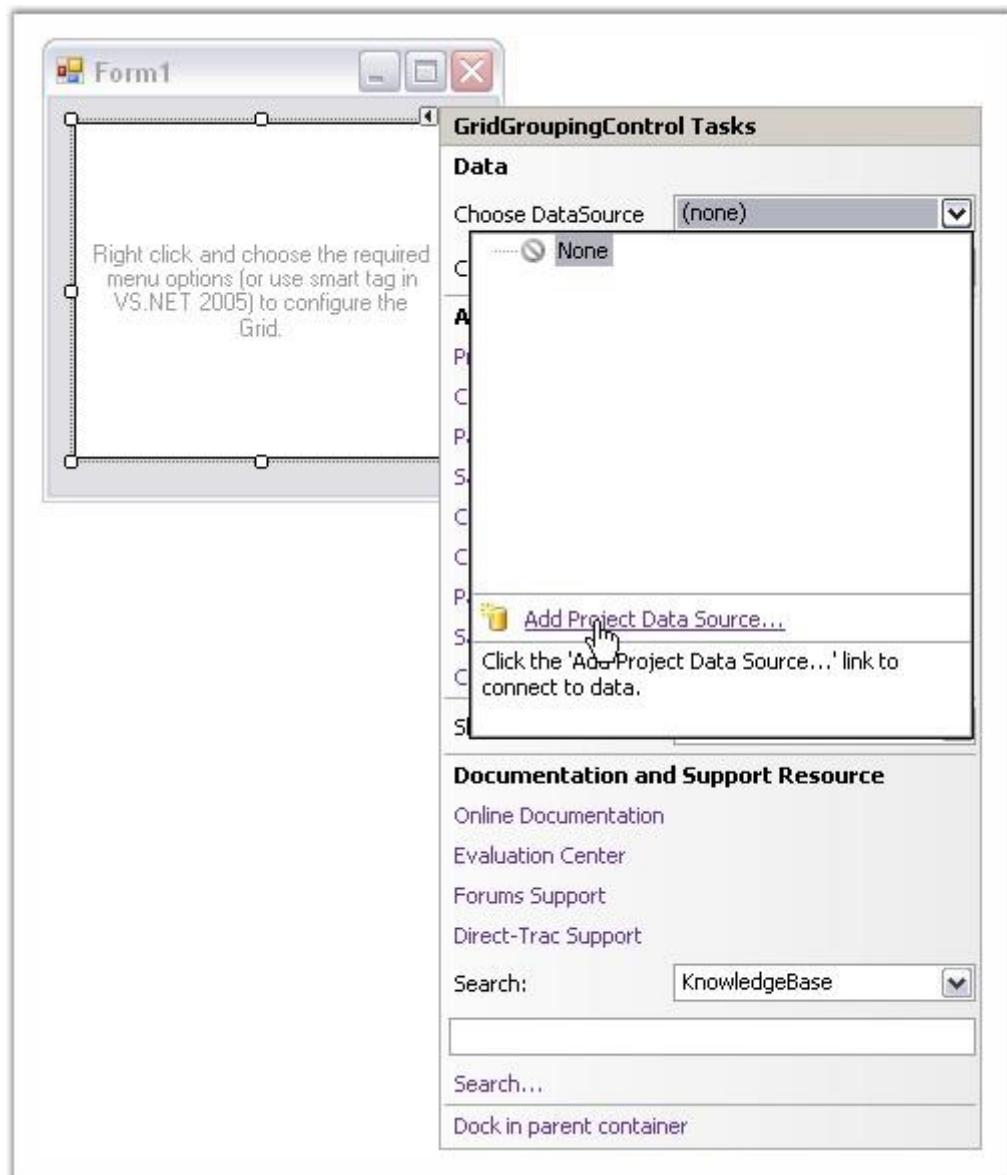


Figure 27: Grouped Grid

### 3.1.2.2 Binding to an MDB File By Using VS 2005

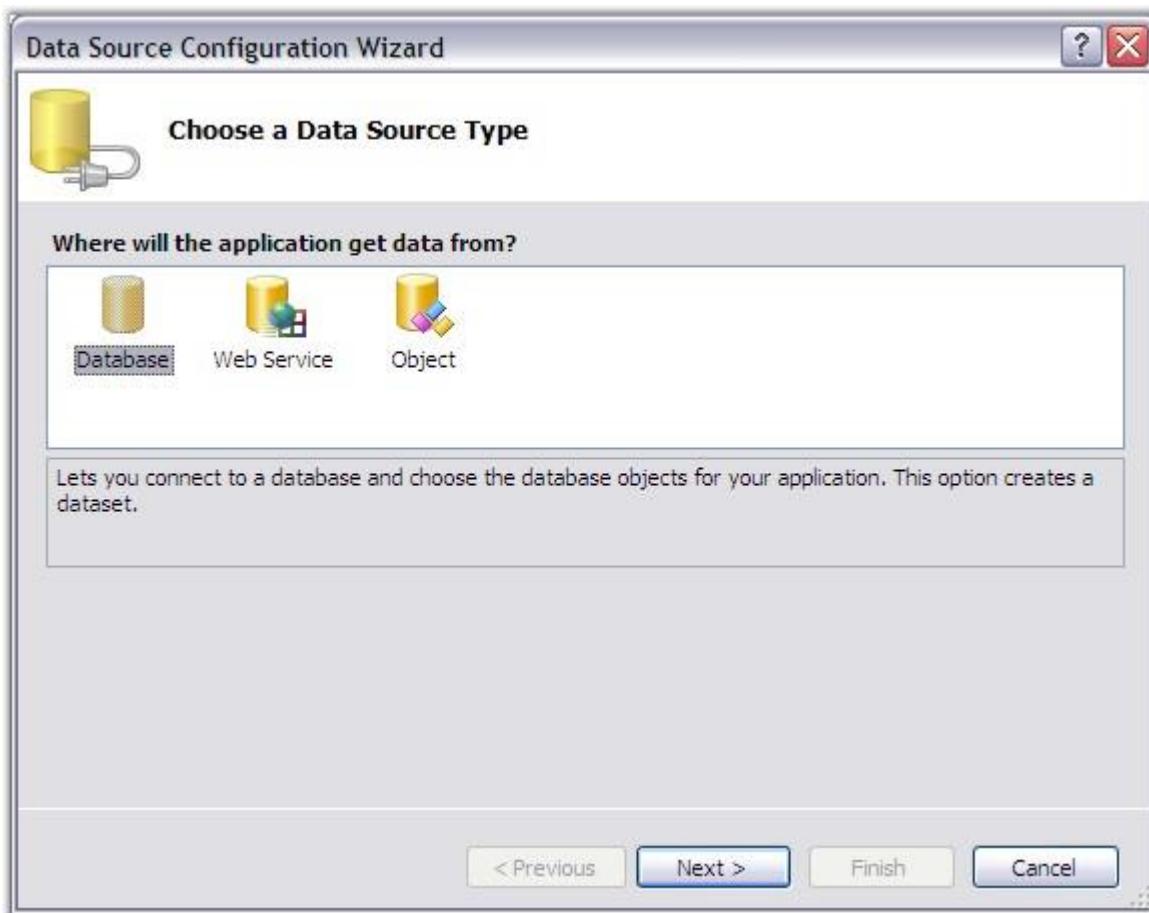
The steps in this lesson are for use with Visual Studio 2005 and .NET 2.0. You can use **Smart Tags** that are available in the .NET 2.0 Designer to hook into your MDB file. This tutorial is strictly a designer tutorial. You do not have to write even a single line of code.

1. From the **Syncfusion** tab in the toolbox, drag a **Grid Grouping control** onto your form.
2. In the Grid Grouping control smart tag, click the **Choose Data Source** drop down. Then click the **Add Project Data Source** link in the drop down.



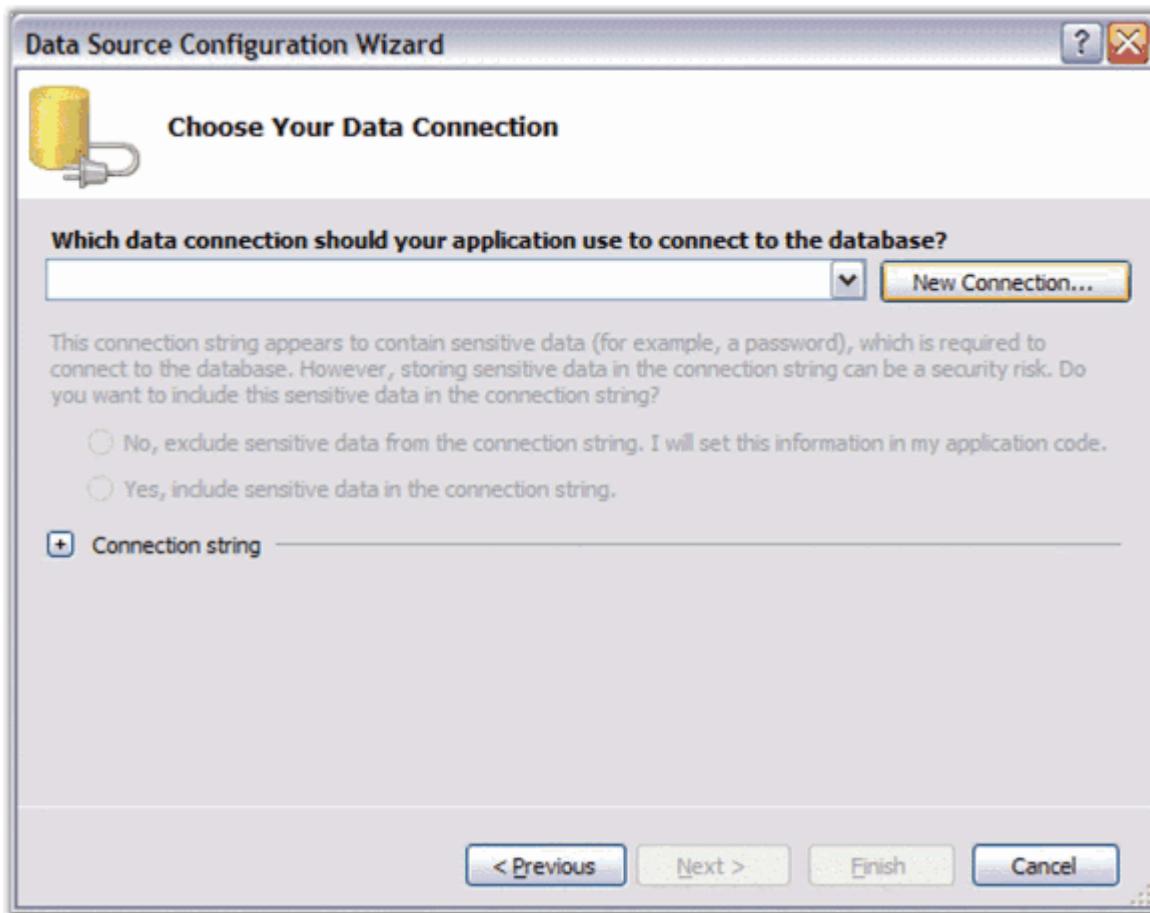
*Figure 28: Choosing a Data Source from the Smart Tag*

3. In the Data Source Configuration Wizard that appears, select DataBase and click Next.



*Figure 29: Choosing a Data Source Type*

4. Click **New Connection**. The **Add Connection** dialog box will be displayed.



*Figure 30: Choosing the Data Connection*

5. In the Add Connection dialog box, click **Change** button. This opens the **Change Data Source** dialog box.



Figure 31: Add Connection Dialog Box

6. In the Change Source dialog box, select the **Microsoft Access DataBase File** option, and then click **OK**.

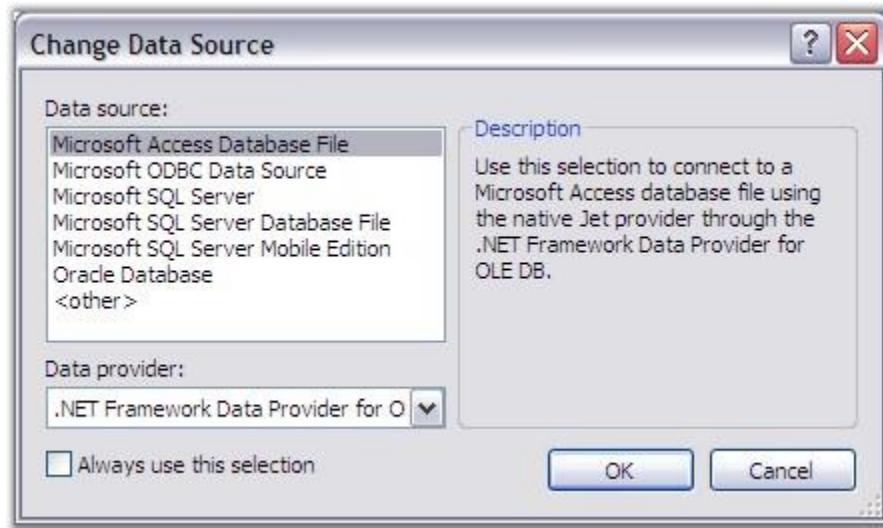
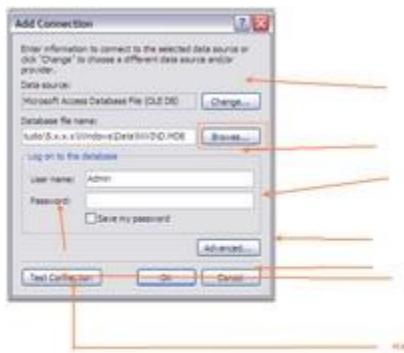


Figure 32: Choosing the Data Source

7. The **Add Connection** dialog box will be opened without the DataBase file name entry set. Click **Browse** button and browse to the following path: **C:\Syncfusion\EssentialStudio[Version Number]\Windows\Data\NWIND.mdb** (this path will vary according to your installation location). Click **OK**.



*Figure 33: Add Connection Dialog Box*

8. Click **New Connection** to choose your data connection. Click **Next**.

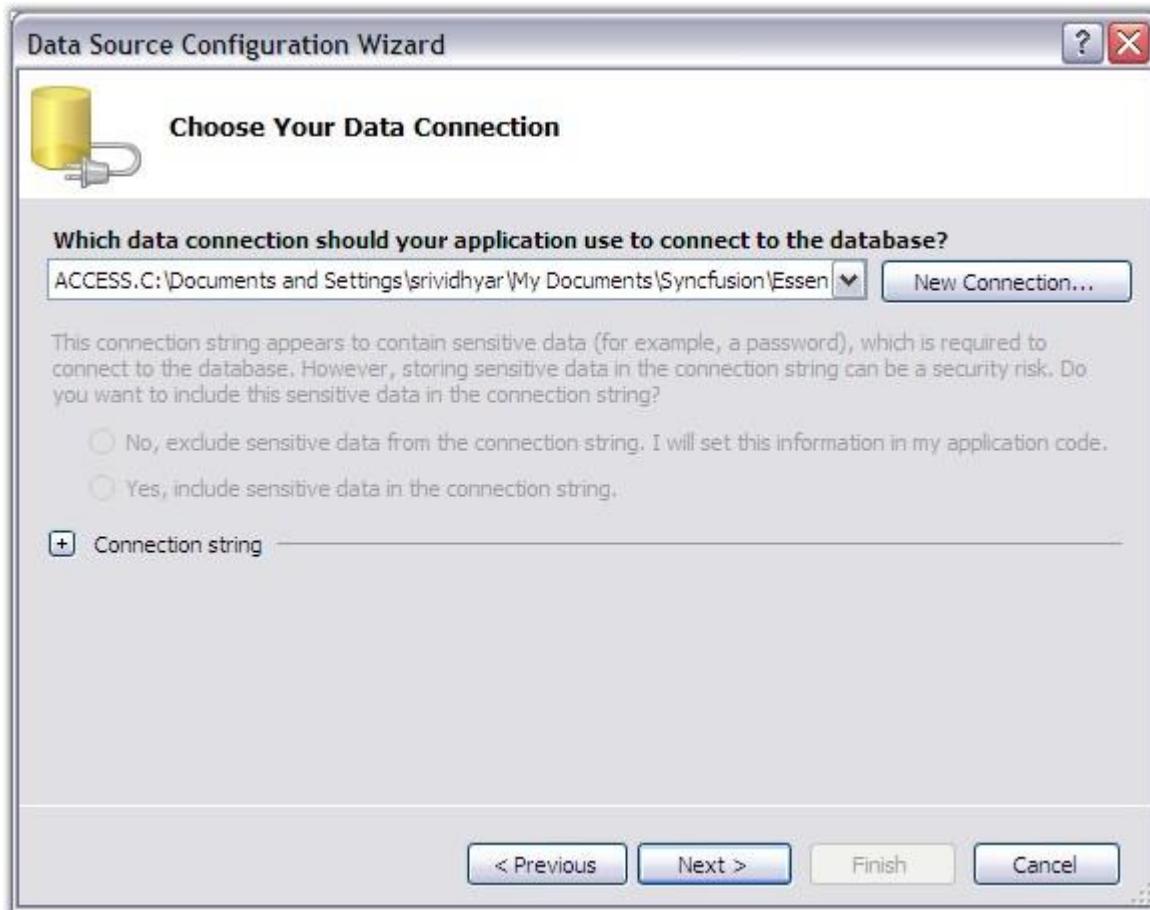


Figure 34: Database Connection Window

9. Click **No** to indicate that you do not want to save the MDB in the project.

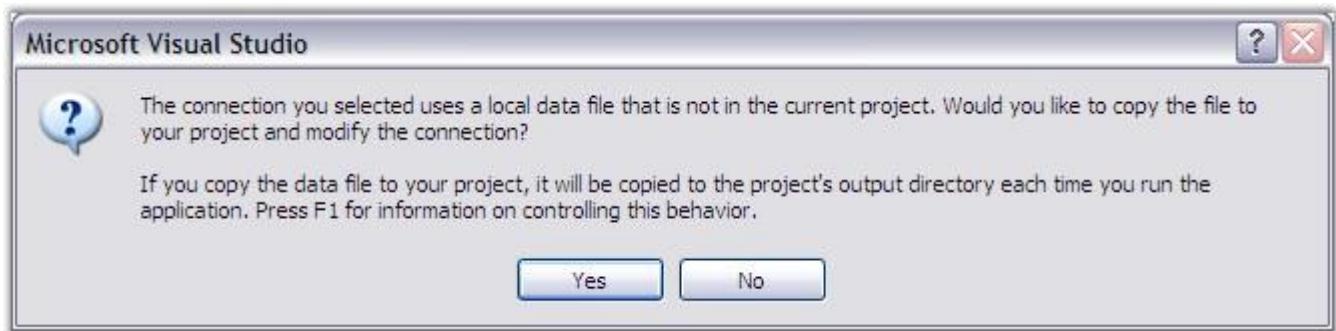
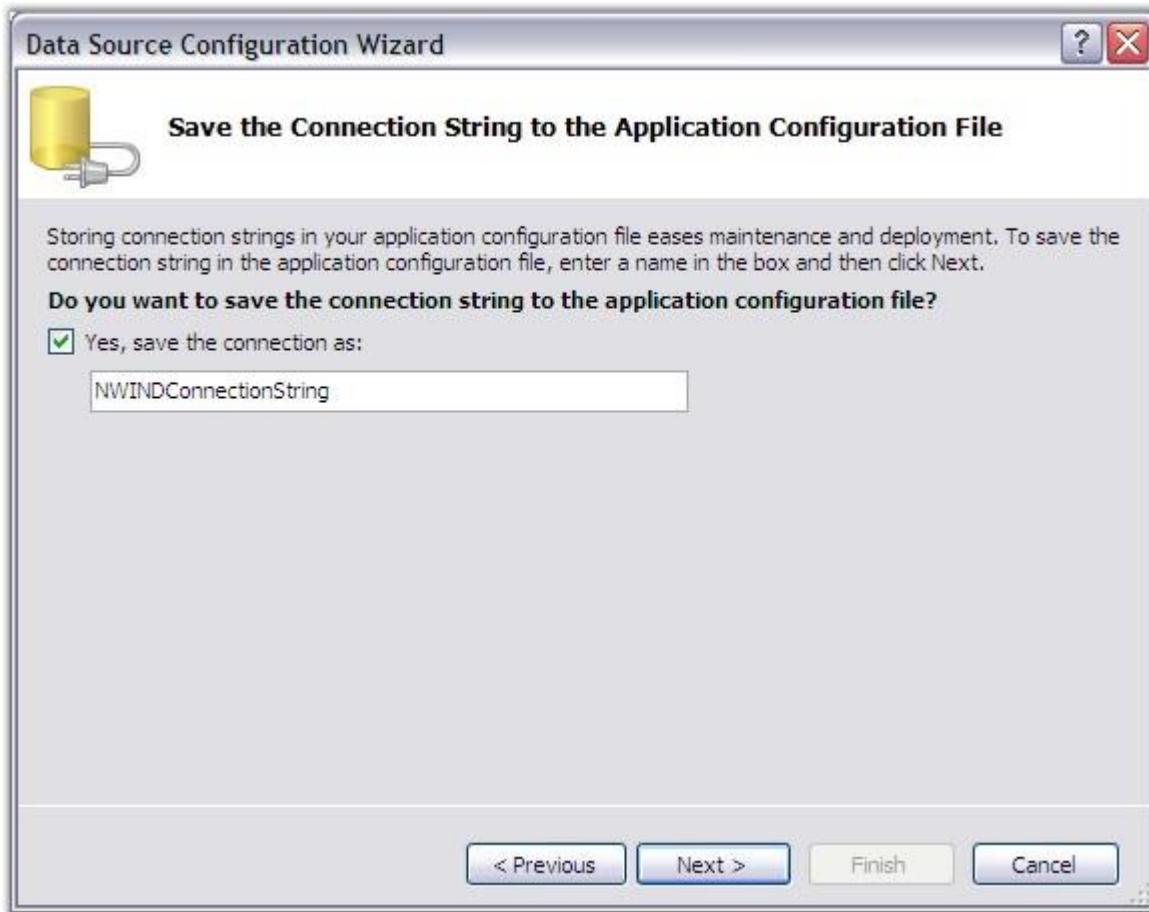


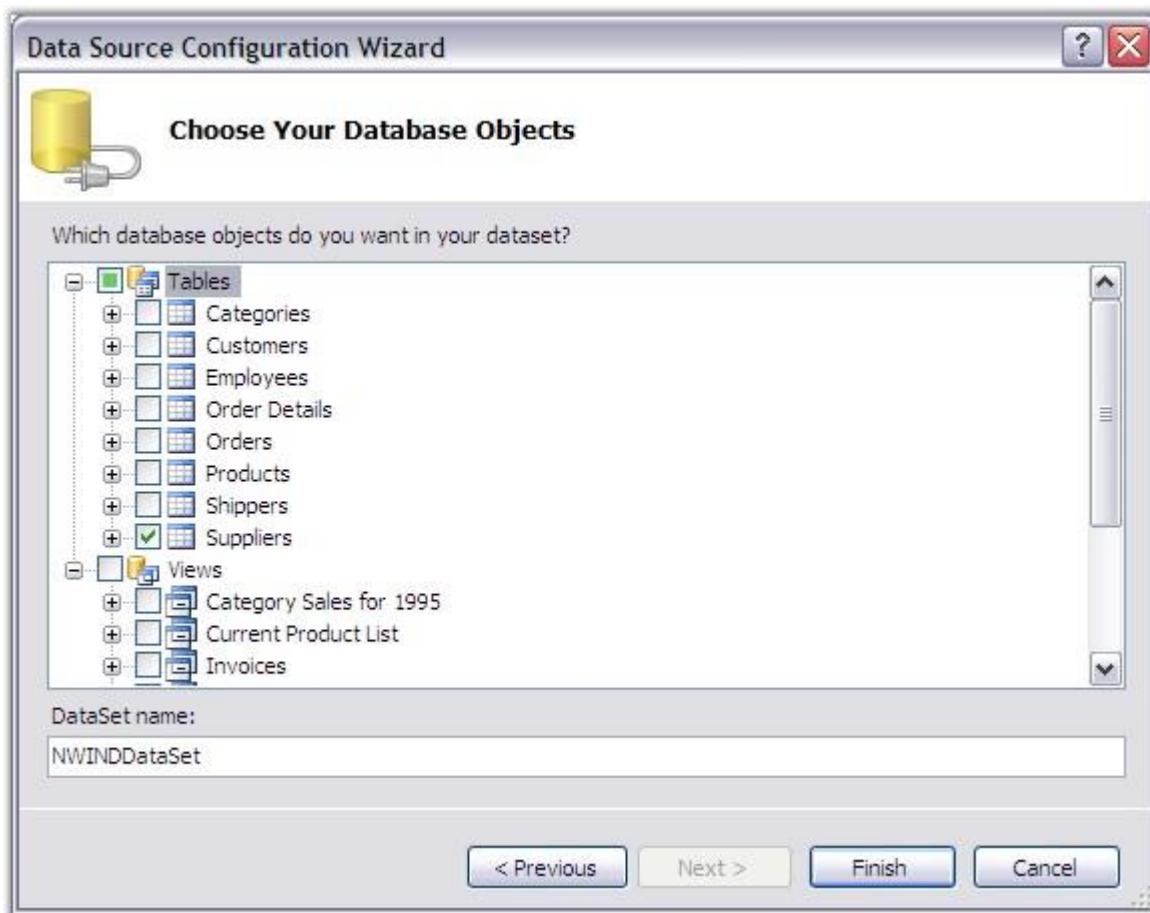
Figure 35: Pop-up Window displayed on clicking the Next Button

The following screen appears.



*Figure 36: Connection String Dialog*

10. Click **Next** to choose your Database Objects. Select the Tables that you want. Click **Finish**.



*Figure 37: New Access File Connection*

The columns in the Grid Grouping control will now get populated as depicted in the following screen shot.

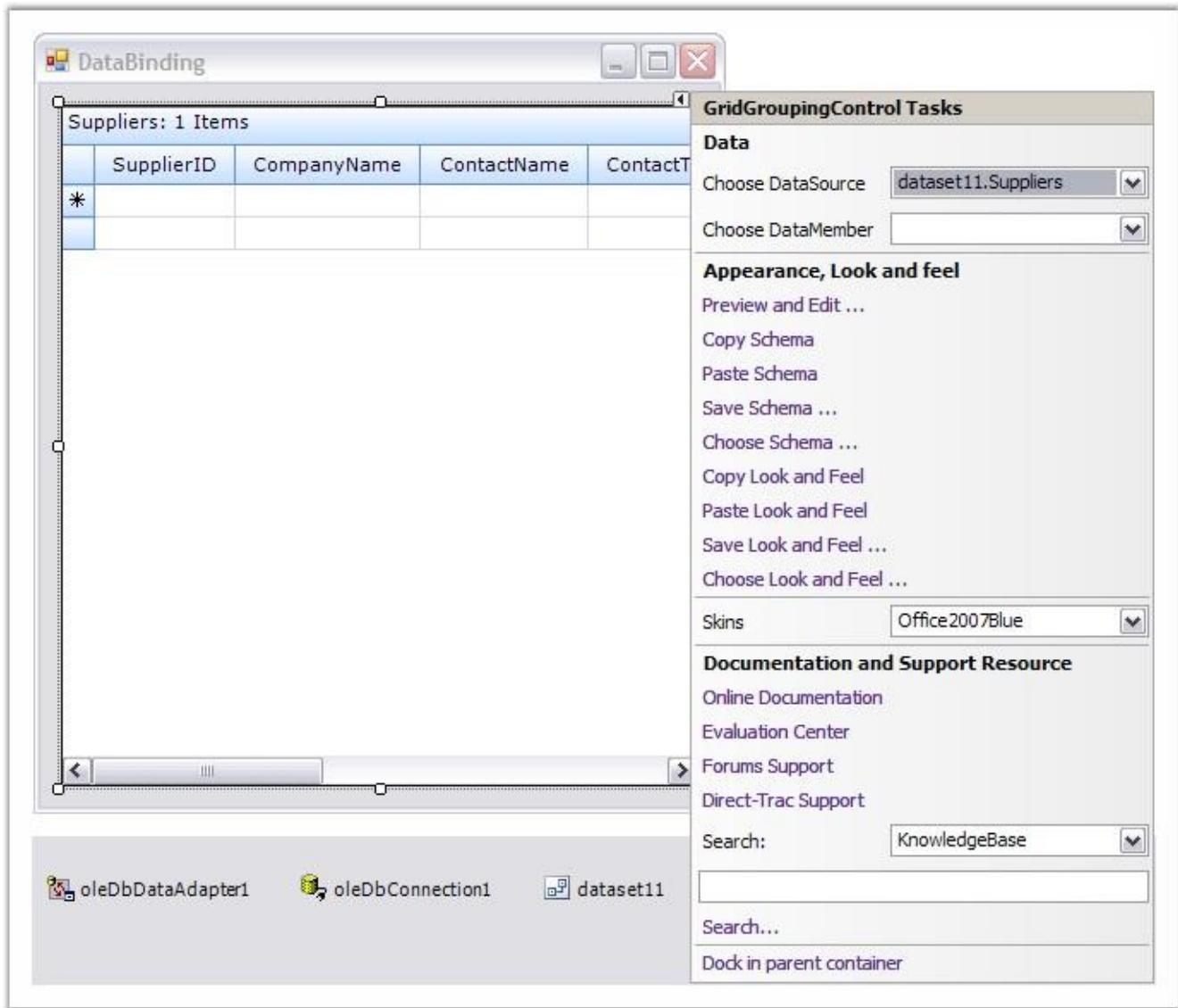


Figure 38: After setting the Data Source

11. Finally, set the **Anchor** property of the Grid Grouping control to *All*, so that the Grid Grouping control can be easily sized with the form. This is depicted in the following screen shot.

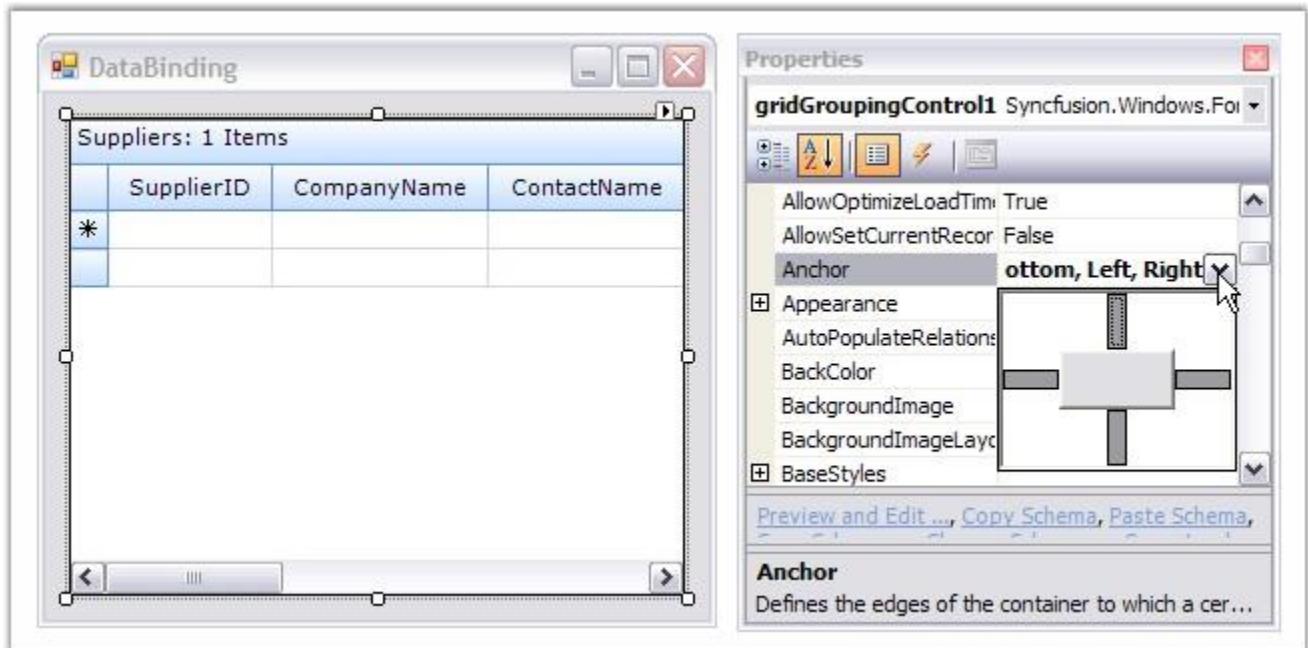


Figure 39: Setting the Anchor Property

12. To allow grouping at run time, the Grid Grouping control displays a drop panel onto which the user can drag columns to be grouped. To display this drop panel, you need to set the **ShowGroupDropArea** property to *true* as shown in the following screen shot.

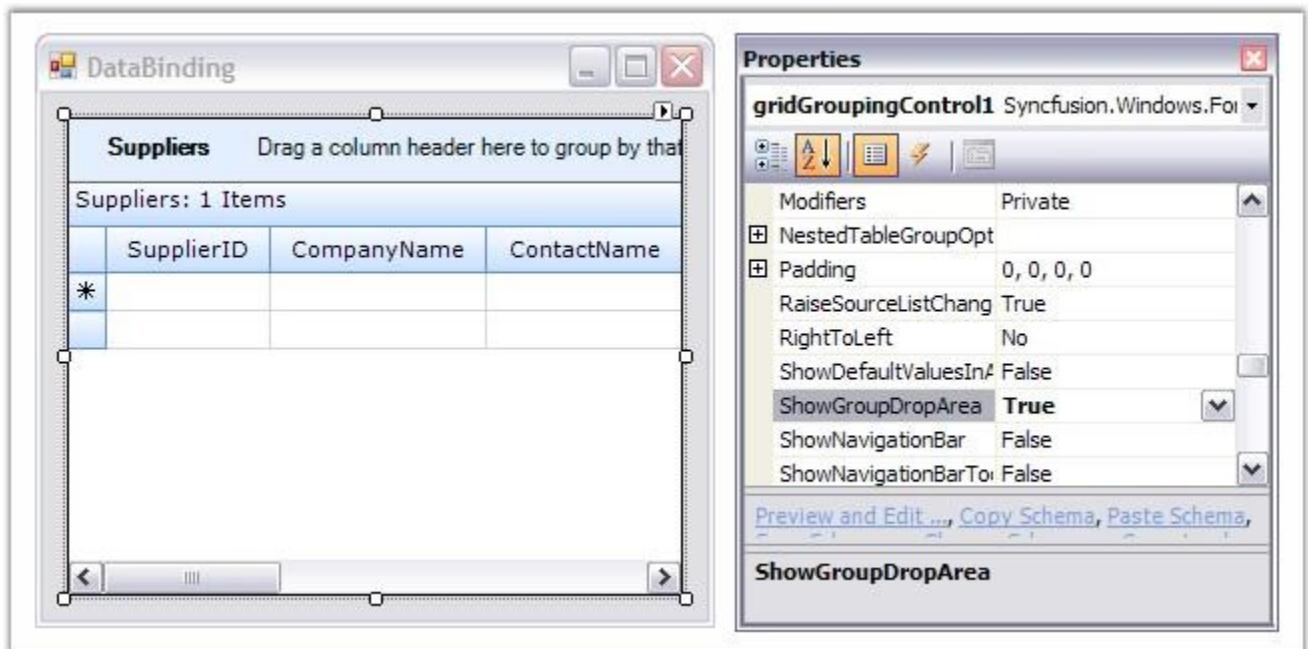


Figure 40: Adding the Group Drop Area

13. Run the application to see the Grid Grouping control display the data from the MDB file (without having written a single line of code). Your form should look similar to the one in the following screen shot.

Suppliers		
SupplierID	CompanyName	ContactName
1	Exotic Liquids	Charlotte Cooper
2	New Orleans Cajun Delights	Shelley Burke
3	Grandma Kelly's Homestead	Regina Murphy
4	Tokyo Traders	Yoshi Nagase
5	Cooperativa de Quesos 'Las Cabras'	Antonio del Valle Saavedra
6	Mayumi's	Mayumi Ohno
7	Pavlova, Ltd.	Ian Devling
8	Specialty Biscuits, Ltd.	Peter Wilson
9	PB Knäckebröd AB	Lars Peterson
10	Refrescos Americanas LTDA	Carlos Diaz
11	Heli Süßwaren GmbH & Co. KG	Petra Winkler
12	Plutzer Lebensmittelgroßmärkte AG	Martin Bein
13	Nord-Ost-Fisch Handelsgesellschaft mbH	Sven Petersen

*Figure 41: Application showing Grid Grouping control with Data*

14. To group by CompanyName, click on the CompanyName column header and drag it to the drop area as illustrated in the following screen shot.

The screenshot shows a Windows application window titled "DataBinding". Inside, a DataGridView displays data from a "Suppliers" table. The columns are "SupplierID", "CompanyName", and "ContactName". A tooltip above the "CompanyName" column says "Drag a column header here to group by that column." The data is grouped by "CompanyName", with 13 distinct groups. Each group has a caption row (e.g., "Exotic Liquids") and an "AddNew" row (indicated by an asterisk). A PlusMinus cell is present in each group's first row. The "CompanyName" column is highlighted with a blue header.

SupplierID	CompanyName	ContactName
1	Exotic Liquids	Charlotte Cooper
2	New Orleans Cajun Delights	Shelley Burke
3	Grandma Kelly's Homestead	Regina Murphy
4	Tokyo Traders	Yoshi Nagase
5	Cooperativa de Quesos 'Las Cabras'	Antonio del Valle Saavedra
6	Mayumi's	Mayumi Ohno
7	Pavlova, Ltd.	Ian Devling
8	Specialty Biscuits, Ltd.	Peter Wilson
9	PB Knäckebröd AB	Lars Peterson
10	Refrescos Americanas LTDA	Carlos Diaz
11	Heli Süßwaren GmbH & Co. KG	Petra Winkler
12	Plutzer Lebensmittelgroßmärkte AG	Martin Bein
13	Nord-Ost-Fisch Handelsgesellschaft mbH	Sven Petersen

Figure 42: Grouping by the CompanyName Column

Notice that each set of grouped values has its own "Caption" row and its own "AddNew" row (\*). Each group has its own PlusMinus cell that will let you expand/collapse the group.

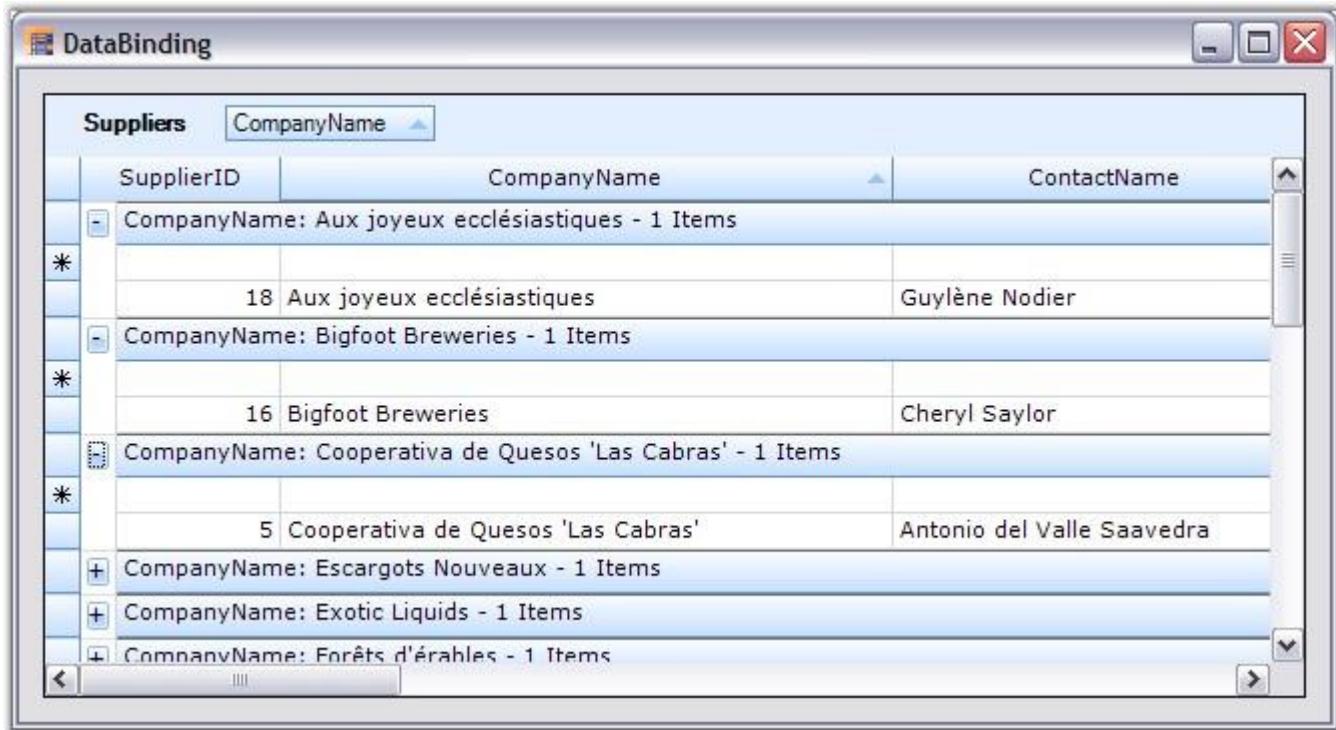


Figure 43: Grouped Grid



**Note:** For more details, refer the following browser sample:

C:\Syncfusion\EssentialStudio\Version  
Number]\Windows\Grid.Grouping.Windows\Samples\2.0\GettingStarted\Data Binding VS 2005 Demo

### 3.1.3 Lesson 2: Grid Control Designer

Starting with Version 4.1, the design time support for the Grid control will be much more user friendly. To make the task of designing the Grid control easier on a cell level, a new Designer Editor has been added. With the editor, the grid can be modified and saved (and loaded) to XML formatted files, or Soap formatted templates. There is also no longer a Toggle Interactive Mode design verb that was present in versions prior to 4.1.

In this lesson, you will learn about a basic [Grid Control](#).

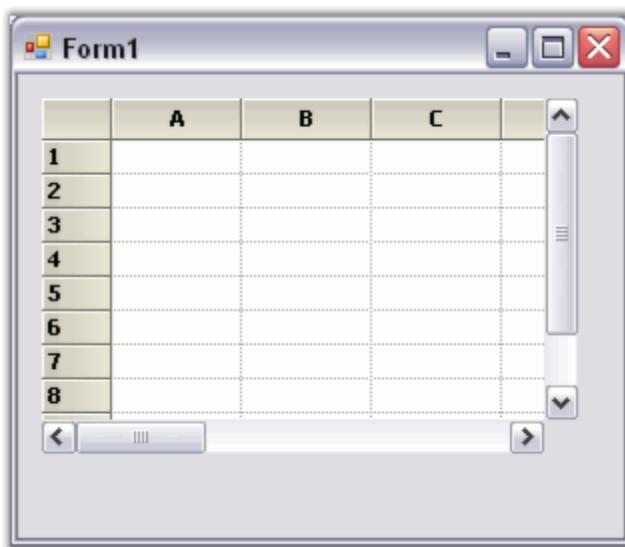
### 3.1.3.1 Basic Grid Control

This section covers information on how to add a grid control to your Windows application.

#### Adding a Grid Control to your Application

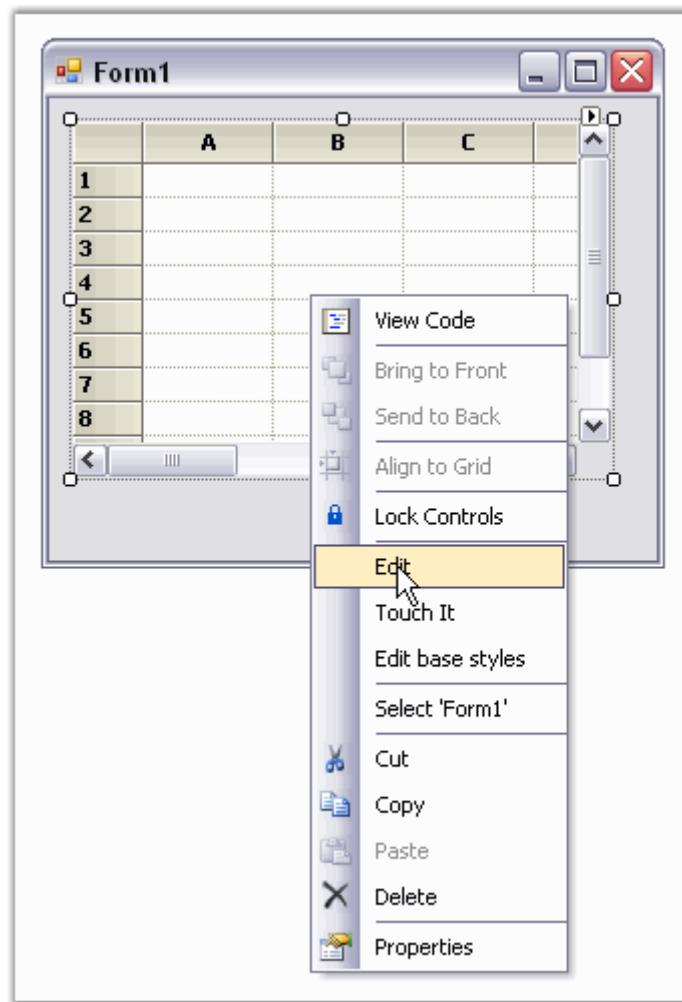
Following step-by-step procedure illustrates how to add a simple grid control to your application.

1. Drag the **GridControl** component from the toolbox onto the form.



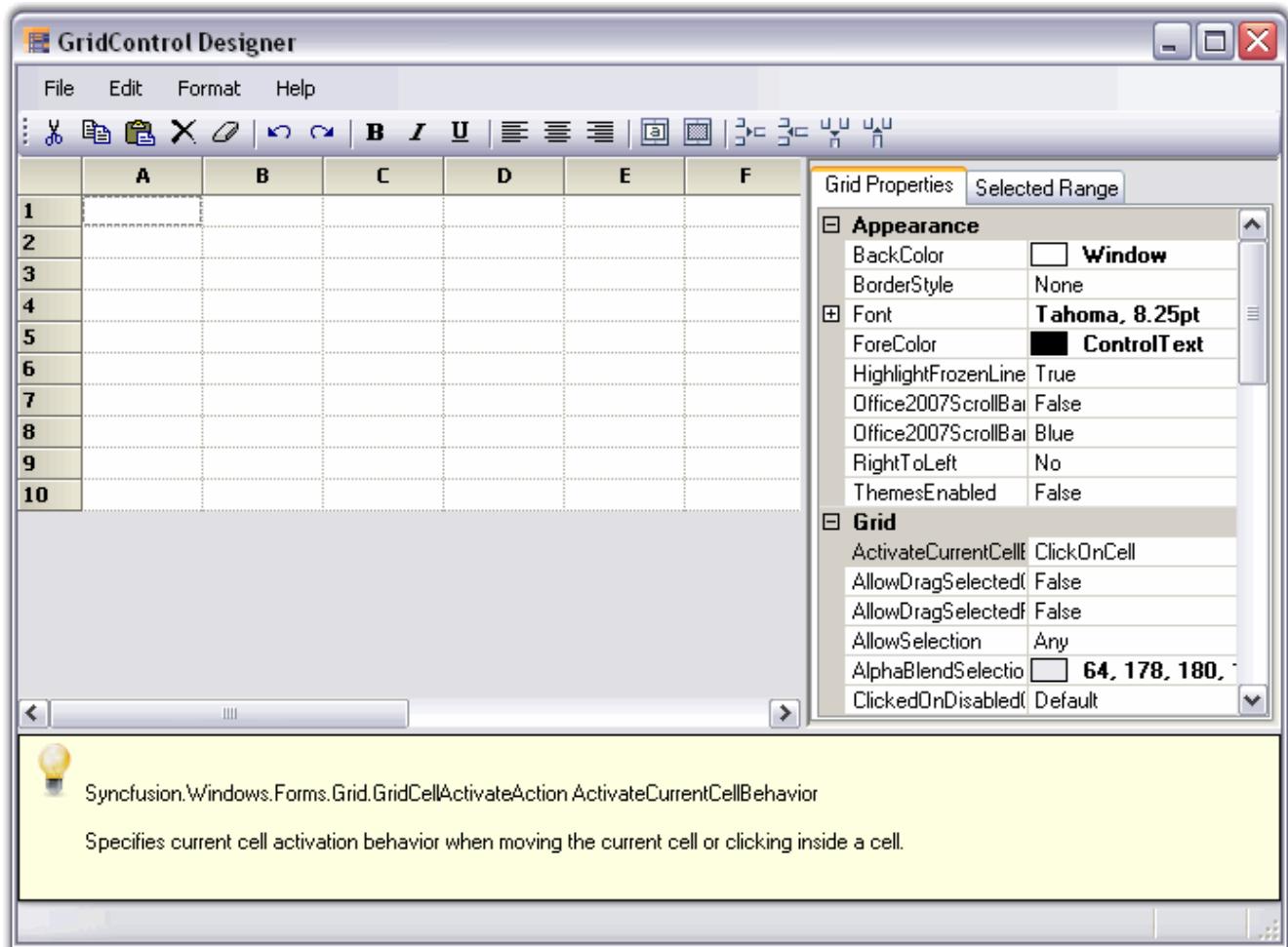
*Figure 44: Grid Control*

2. To edit the cell level properties of the grid (and also general Grid control properties), right-click on the Grid control and select **Edit** from the context menu.



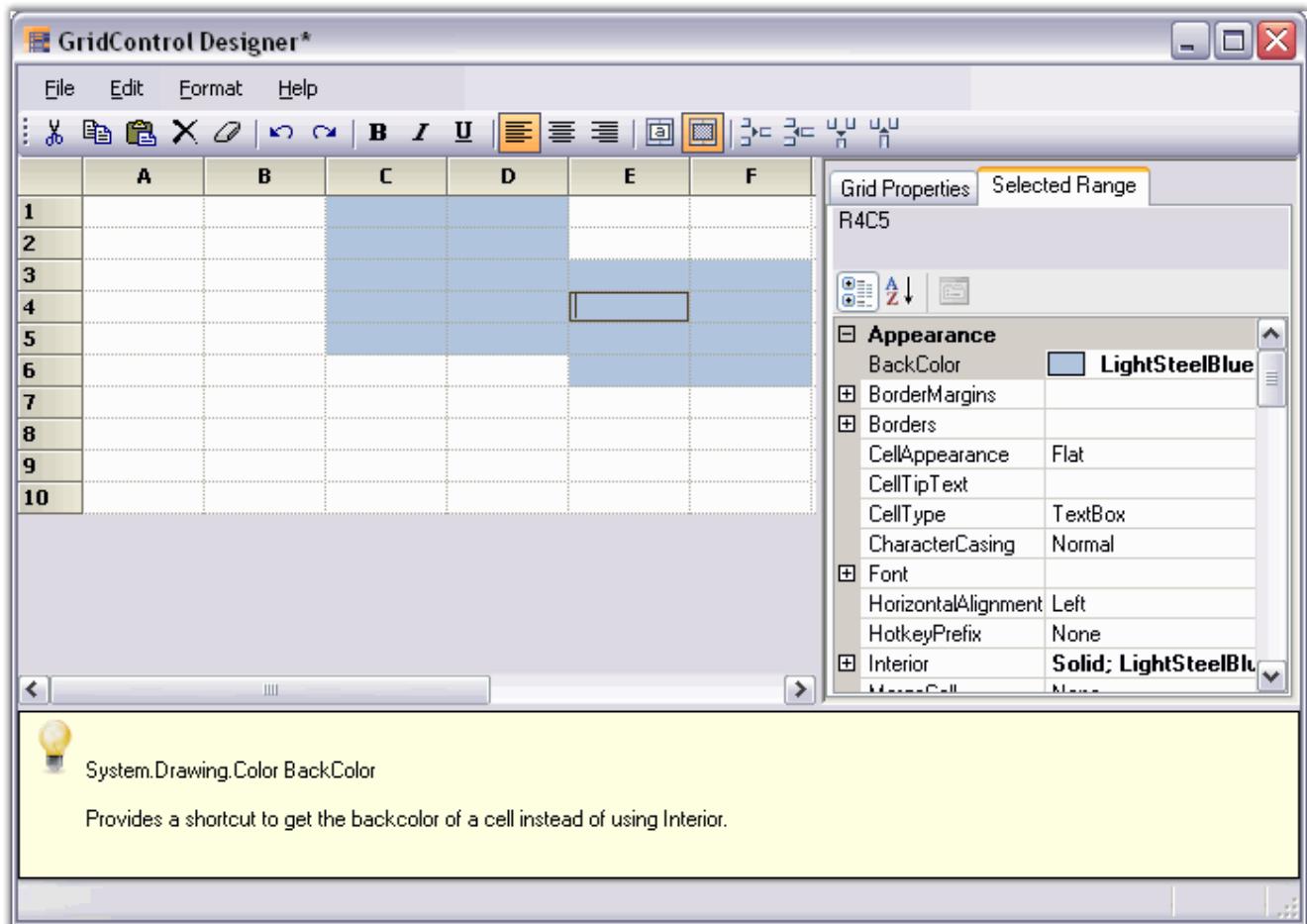
*Figure 45: Grid Control Context Menu*

The **GridControl Designer** window is displayed. By using the GridControl Designer, the cell contents/styles, as well as general grid properties can be modified.



*Figure 46: GridControl Designer*

3. Single cells can be modified along with a selection of ranges. To do this, select a range of cells, and switch to the **Selected Range** tab to view the property grid for the selection.



*Figure 47: Property Grid for Selected Range*

GridControl Designer also lets you to save/load xml formatted files, and Soap templates.

4. When the changes are complete, simply exit out of the designer. If changes have been made, you will be prompted to save the changes to the Grid control in the designer.

### **3.1.4 Lesson 3: Grid Data Bound Grid Designer**

In this lesson, you will learn how to use the Forms Designer to create an Essential Grid Data Bound Grid. You can rely on the designer to generate all the code necessary except for two lines: one that fills the ADO.NET adapter and one that updates the database when you are done.

In this lesson, you will learn about the following topics.

### 3.1.4.1 Basic Grid Data Bound Grid

In this part, you will learn how to use the designer to place a Grid Data Bound Grid on a form.

1. In Visual Studio .NET, use the File -> Menu option to create a new Windows Application project, naming it DBGridTutorial.

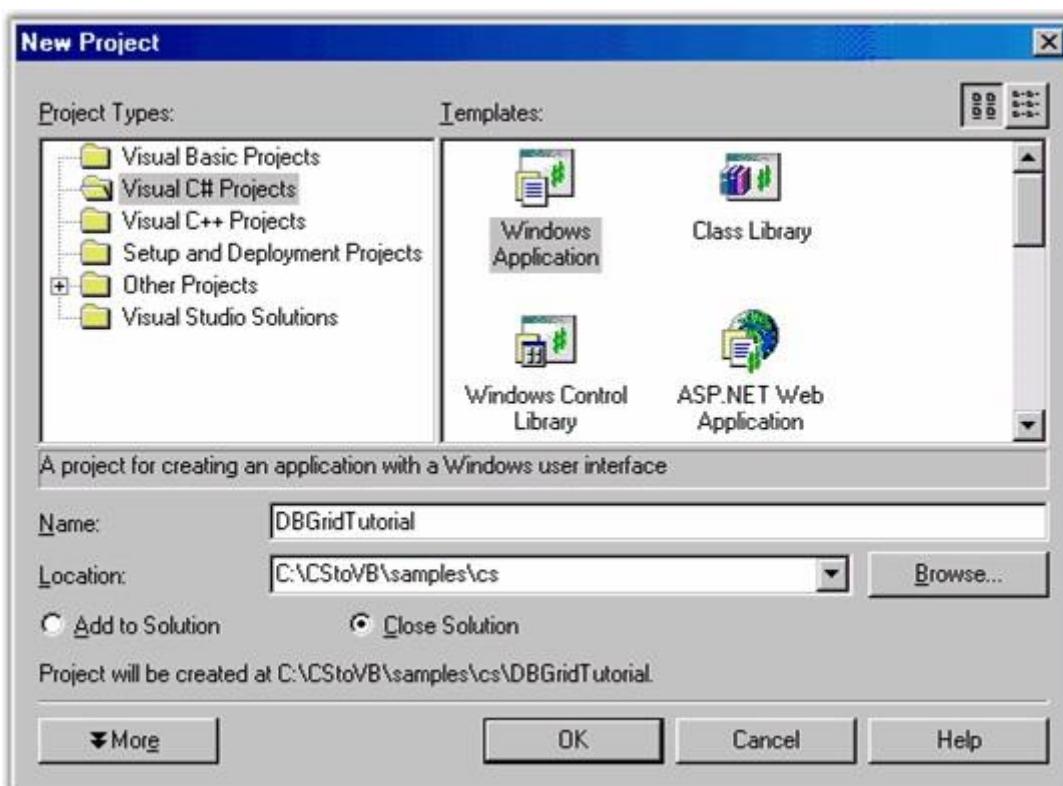
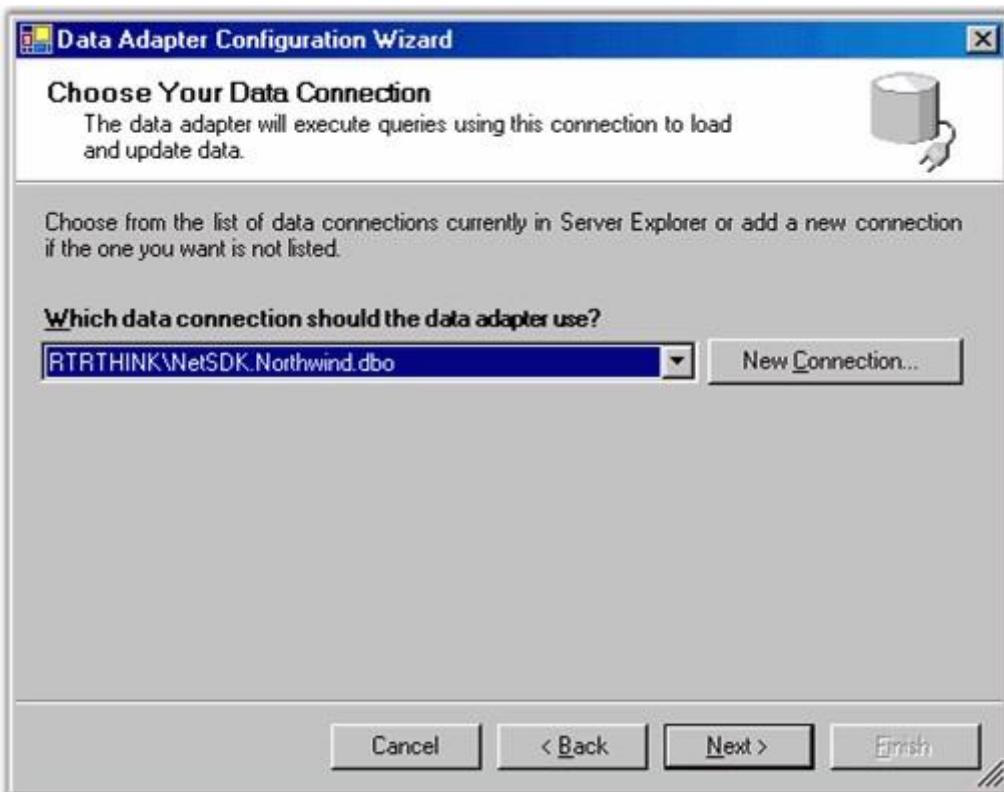


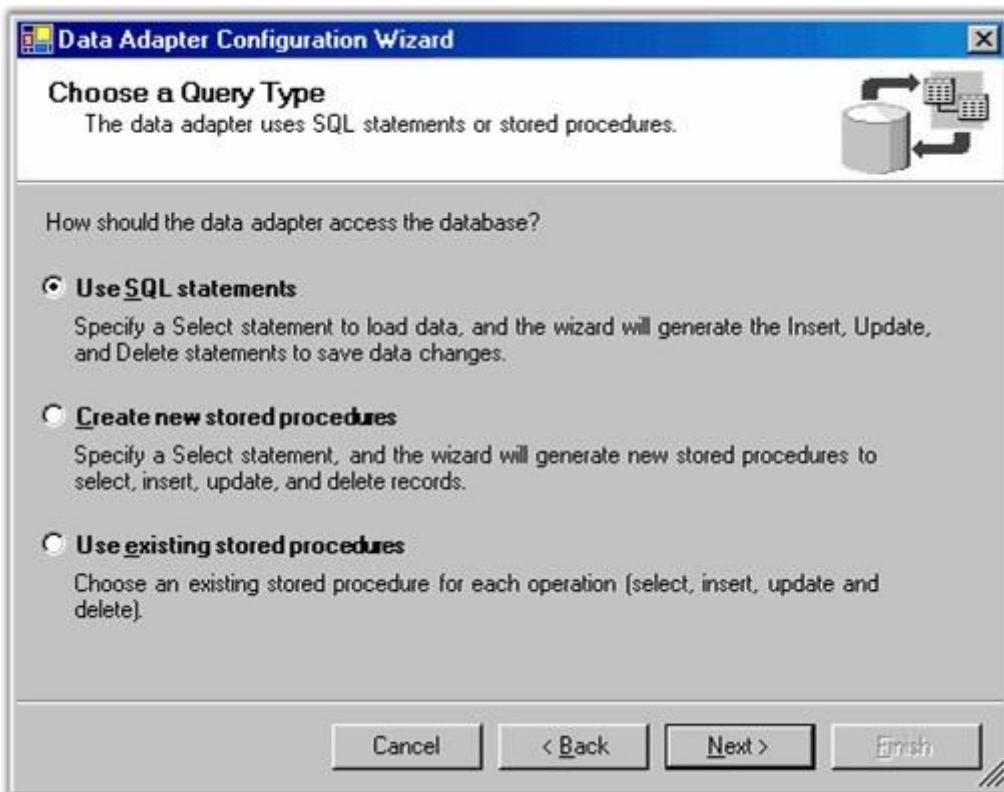
Figure 48: Creating a New Windows Application

2. You now have an empty form on the design surface. Open the Data section of your toolbox and drag a SQLDataAdapter onto your form. This will open a Data Adapter Wizard.
3. Use the wizard to create a connection to the NorthWind database. This DataBase is installed as part of the .NET Framework ADO.NET samples.



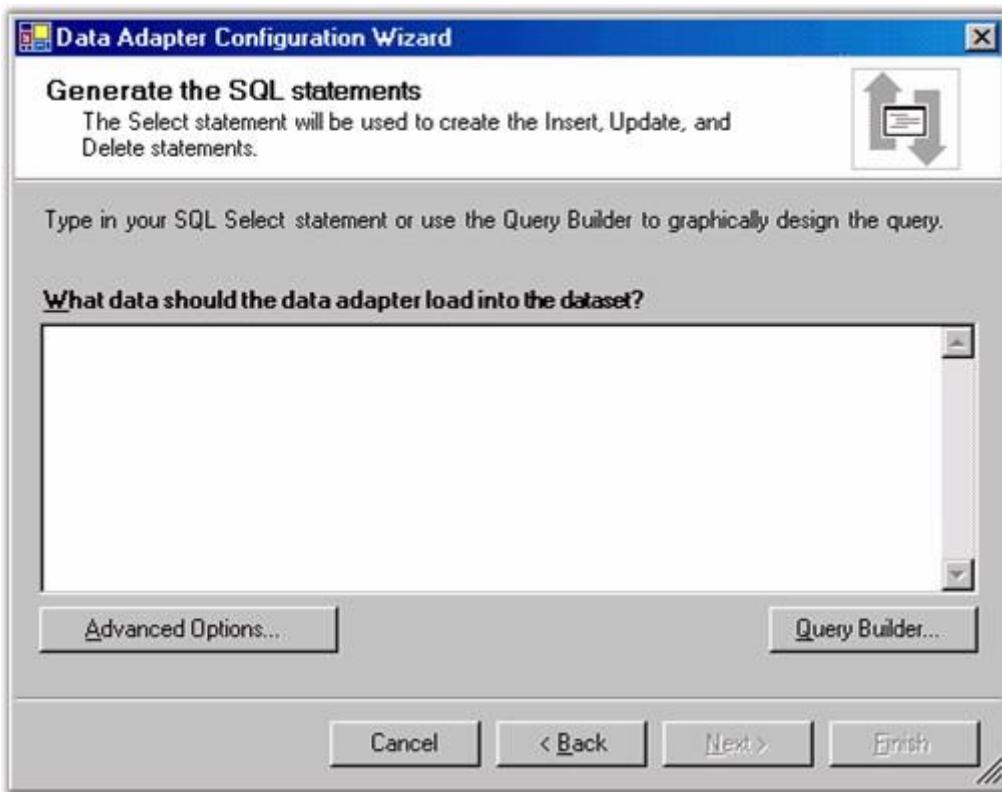
*Figure 49: Select the NorthWind Database*

4. Select to use SQL statements.



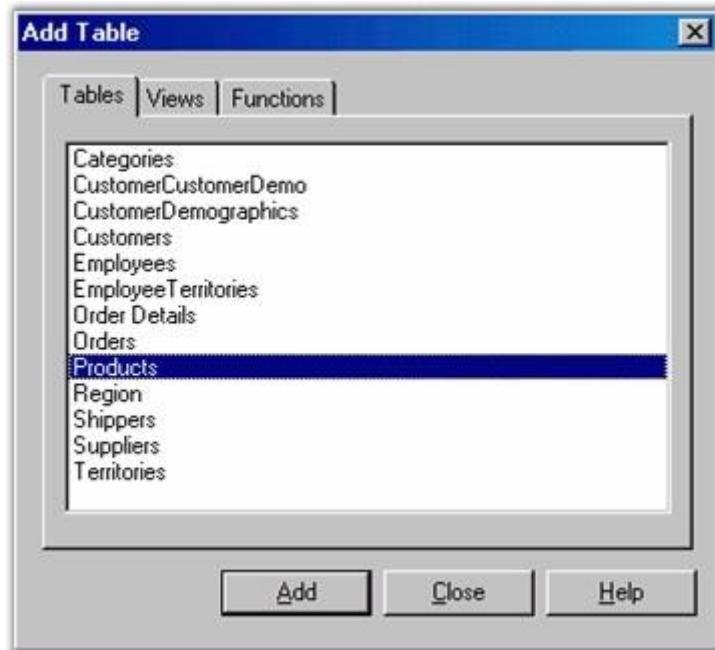
*Figure 50: Select to Use SQL Statements*

5. To generate the SQL statement, click the Query Builder button.



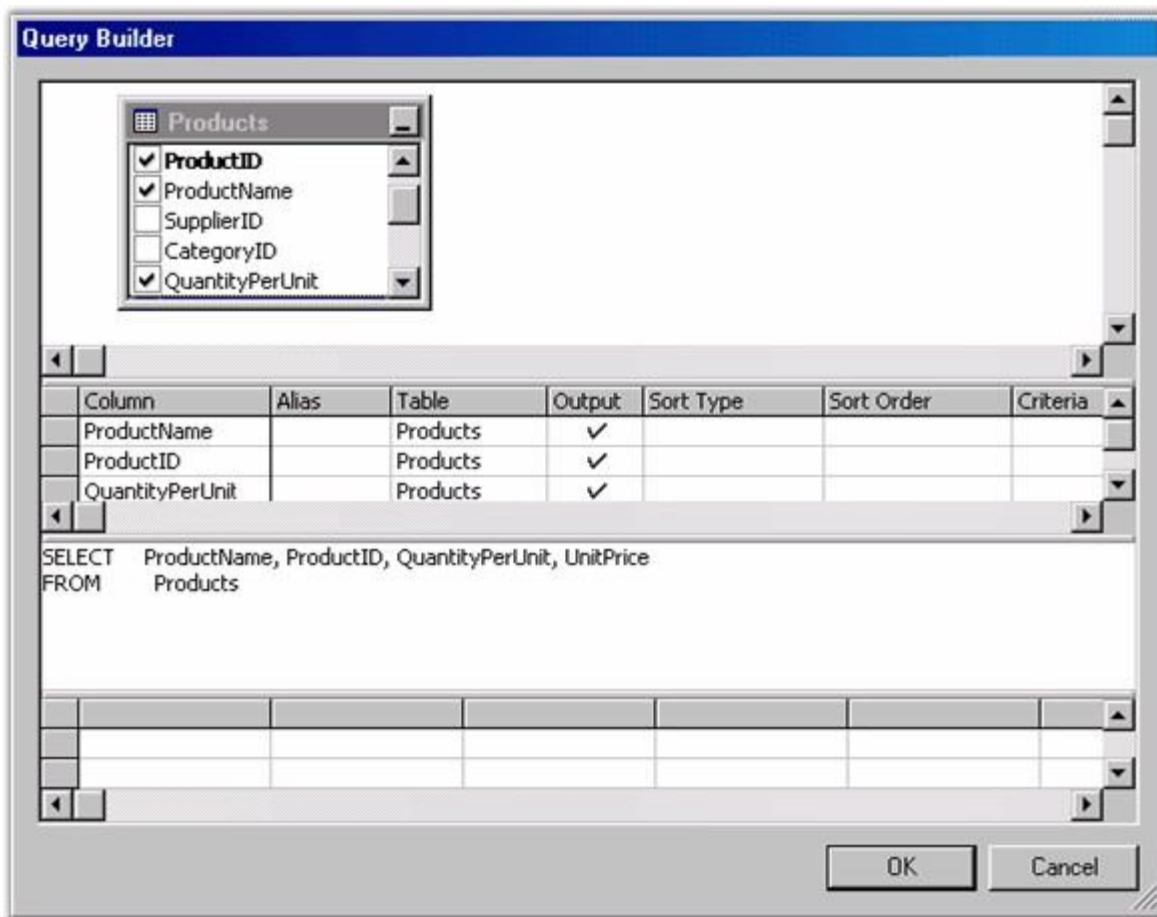
*Figure 51: Select the Query Builder*

6. In the Add Table dialog, select the Products table and click Add, then Close.



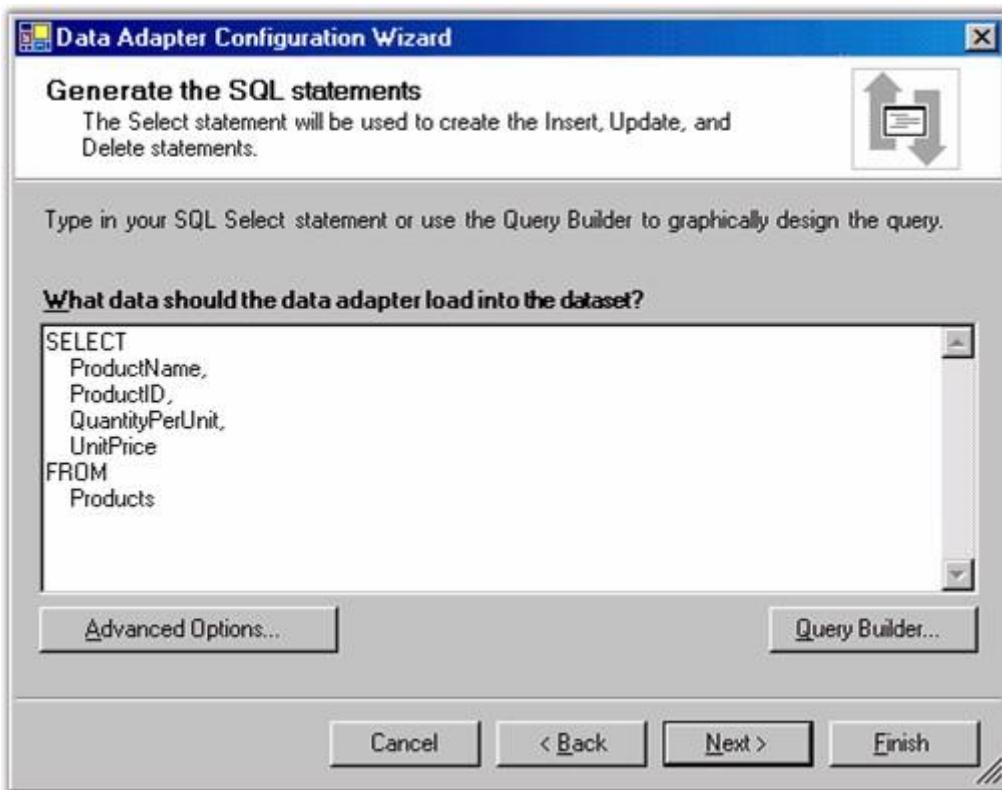
*Figure 52: Select the Products Table*

7. In this Query Build window, select the ProductName, ProductID, QuantityPerUnit and UnitPrice. Then press OK.



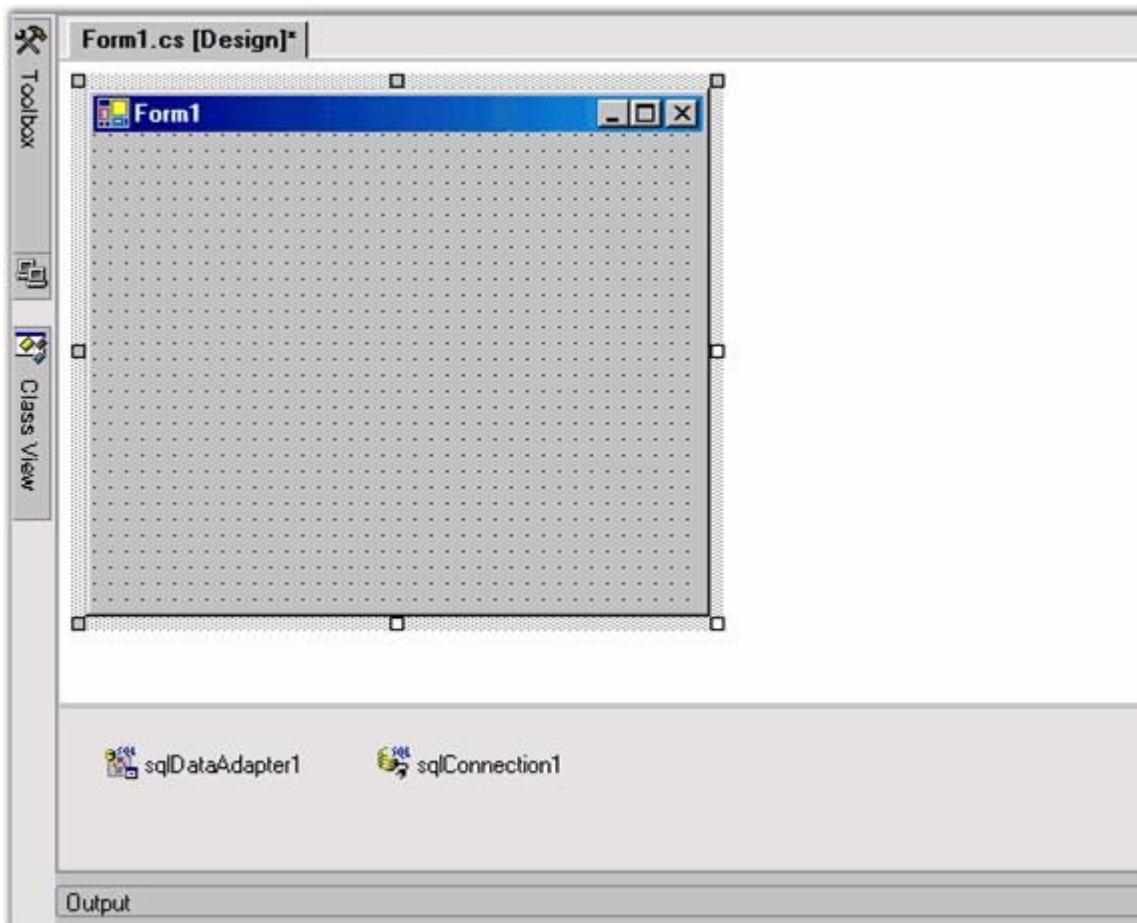
*Figure 53: Select the Fields for your Query*

8. Click Next to confirm the Query you selected.



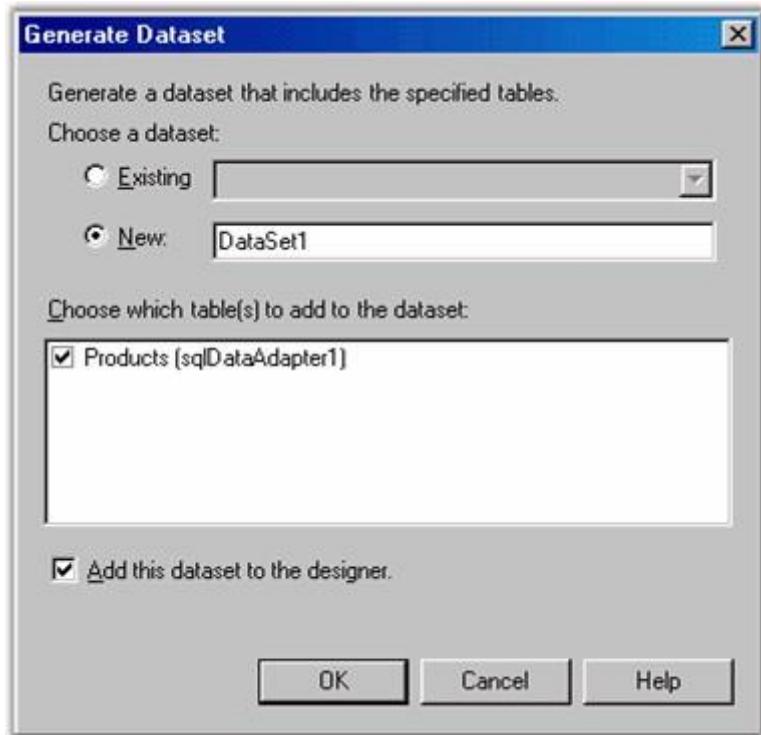
*Figure 54: Confirm the Query*

9. Click Finish. Your design surface will look similar to this.



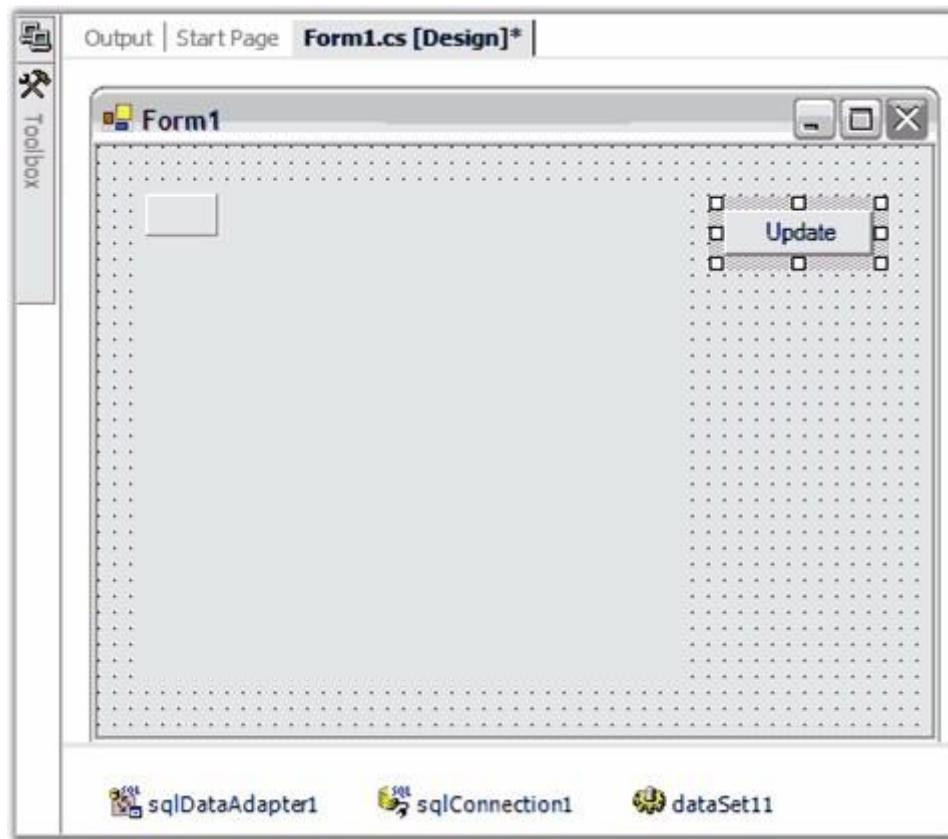
*Figure 55: Adapter and Connection*

10. Next you will need to generate a dataset from the SQLDataAdapter. Right-click the sqlDataAdapter1 under the design surface and select Generate **DataSet**. You will then see this window.



*Figure 56: Generating a Dataset*

11. Press OK to add a DataSet1 object next to the SqlConnection1 under the design surface.
12. From the toolbox, drag the Grid Data Bound Grid control to your form. Size and position it and add a button labeled Update to your form.



*Figure 57: Designer with Grid Data Bound Grid and Button*

13. Click the Grid Data Bound Grid to display its properties in the **PropertyGrid**. Set these properties.

DataSource	DataSet11
DisplayMember	Products

14. Double click the form on the design surface (not one of the controls, but the form itself) to add a load event handler. In this handler, add this single statement which, is given below.

[C#]

```
// Load dataset with records.  
this.sqlDataAdapter1.Fill(this.dataSet11);
```

**[VB.NET]**

```
' Load dataset with records.  
Me.sqlDataAdapter1.Fill(Me.dataSet11)
```

15. To support updation of the data in your database, you will need to call the **Update** command on the SQLDataAdapter. Double click the Update button on the design surface to add a Click Handler. Then add this single line of code to the handler.

**[C#]**

```
// Save Changes (if any) back to the database.  
this.sqlDataAdapter1.Update(this.dataSet11);
```

**[VB.NET]**

```
' Save Changes (if any) back to the database.  
Me.sqlDataAdapter1.Update(Me.dataSet11);
```

Now when you click the Update button it will post any changes that you have made back to your database.

### 3.1.4.2 Applying Special Column Formats

The **GridBoundColumn** collection property of the Grid Data Bound Grid is used to set column properties. This collection will let you control the columns that are displayed as well as their order. For each column that you want displayed, add a Grid Bound Column. In this Grid Bound Column, you must set the **MappingName** property; the other properties such as **HeaderText** and the **Style** are optional. Under the Style property, you will have access to the normal **GridStyleInfo** properties that you can apply to this column such as **BackColor**, **CellType** and **Font**.

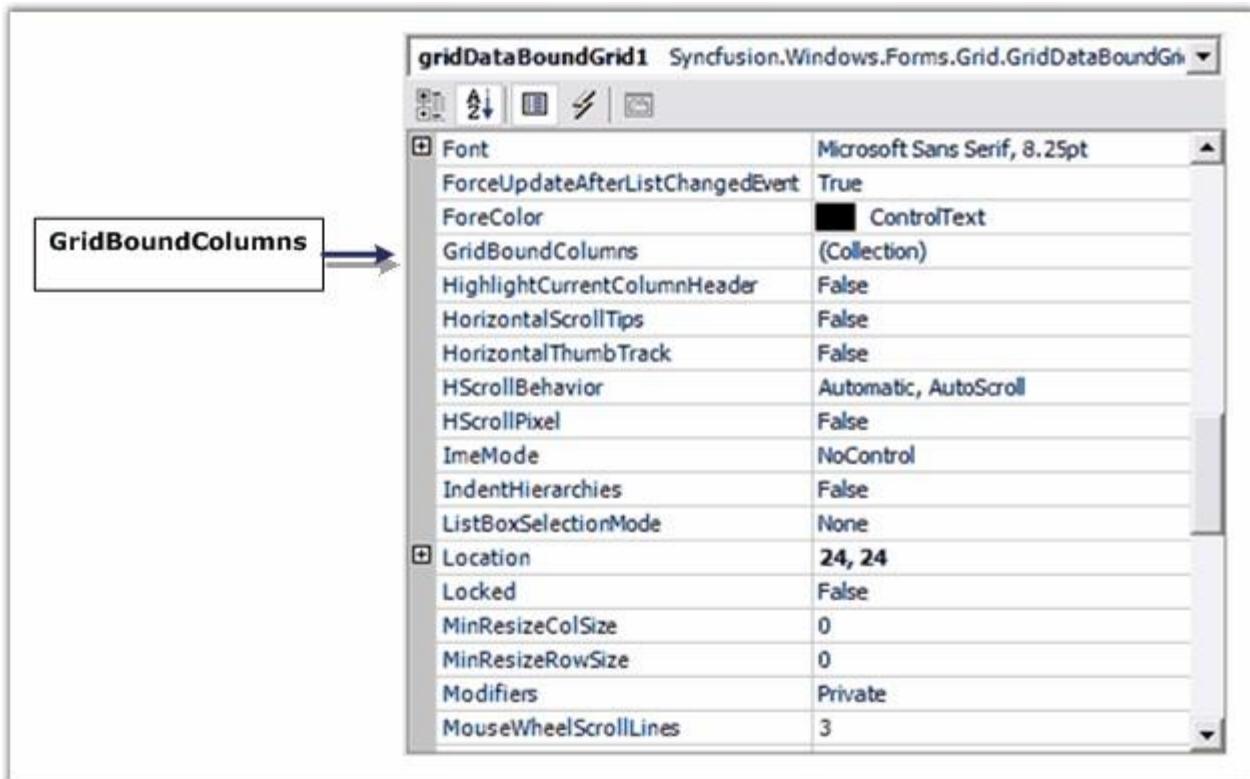


Figure 58: GridBoundColumns Collection in the Grid Data Bound Grid's PropertyGrid

1. Open the **GridBoundColumns** collection editor by using the property grid.
2. Click the **Add** button to add a grid bound column, and then set **MappingName** property of that grid bound column to *ProductName*.

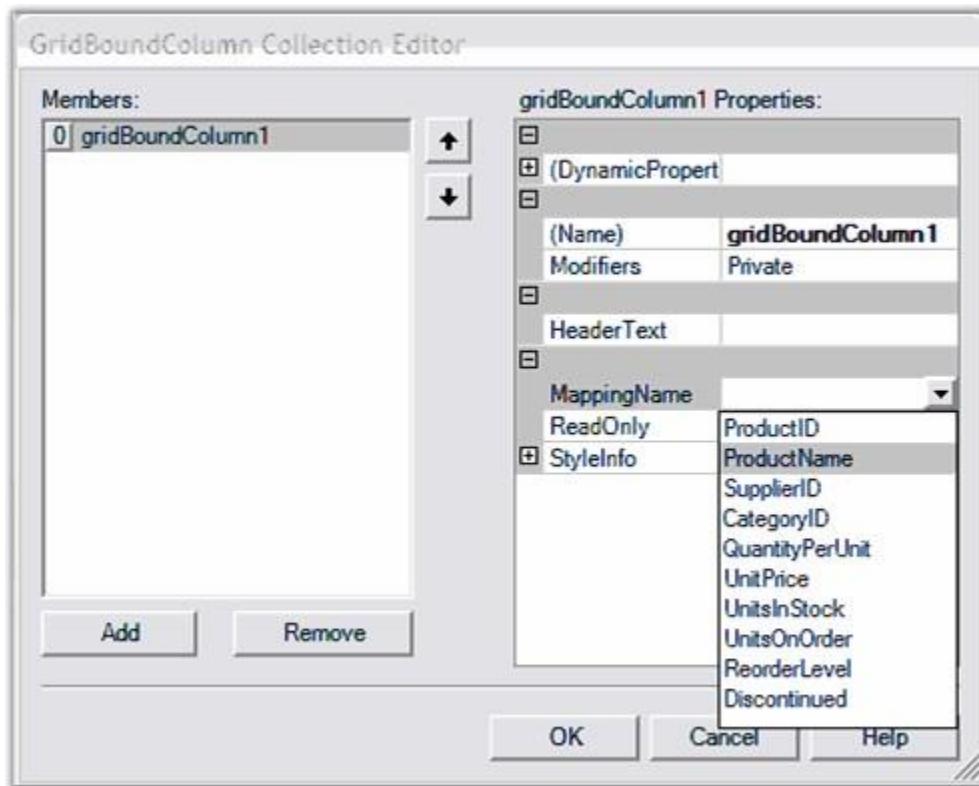


Figure 59: Adding a GridBoundColumn to hold the ProductName Column

3. Select **StyleInfo** property and set the back color for the column as shown in the following screen shot.

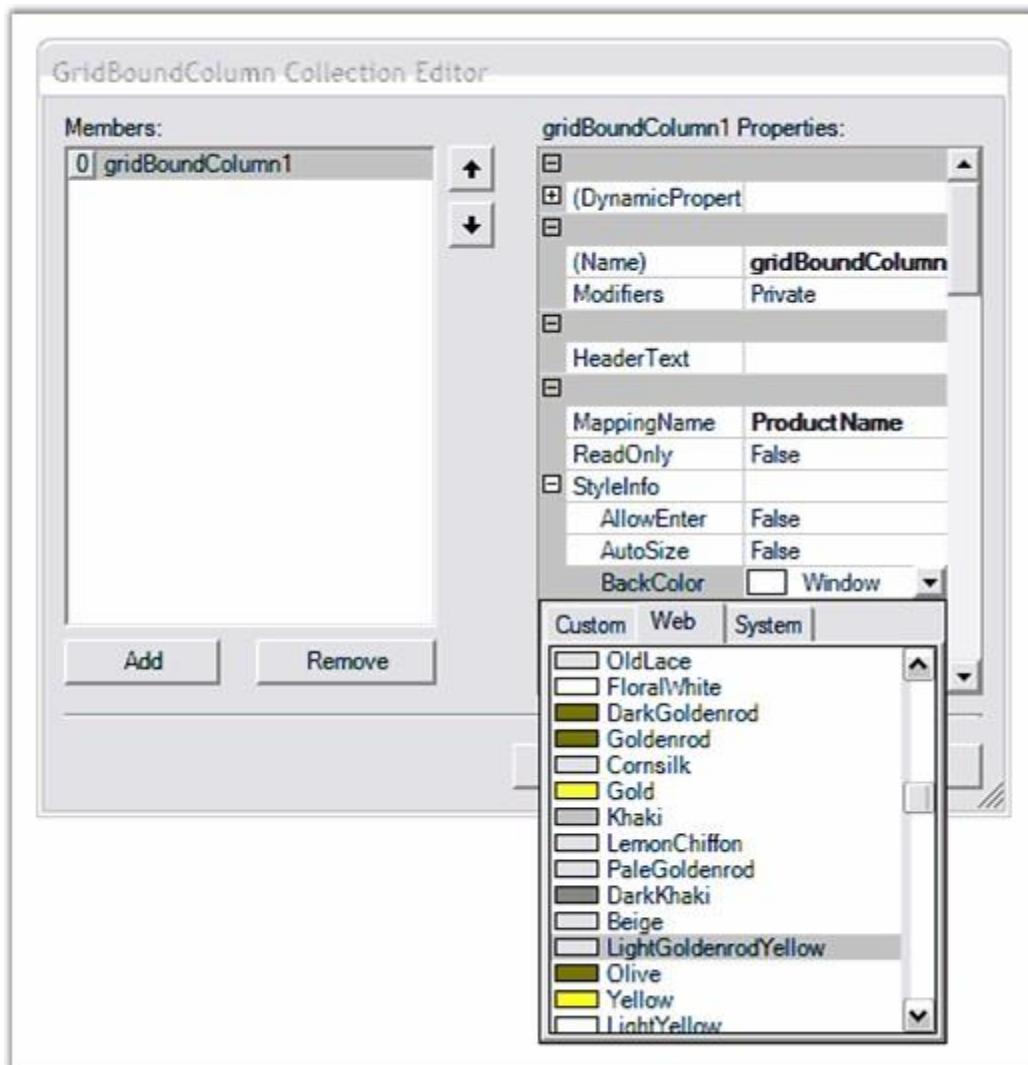
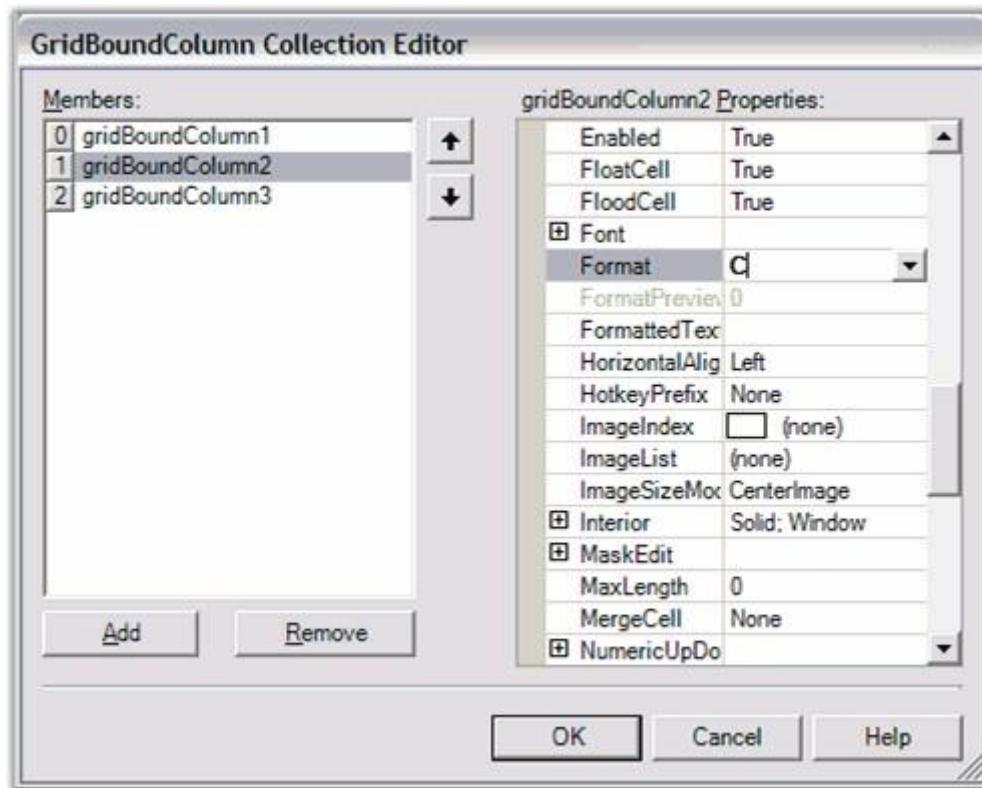


Figure 60: Setting `StyleInfo.BackColor` for the Column

4. Repeat the above steps to add Grid Bound Columns for 'UnitPrice' and 'UnitsInStock'. For the 'UnitPrice' Grid Bound Column, set **StyleInfo.Format** to C.



*Figure 61: Setting the StyleInfo.Format Property on Column 2*

5. Compile and run the project to see the formatted Grid Data Bound Grid. In the following screen shot, you will be able to see the grid with the columns that you specified and in the order that you specified them. Notice that the 'UnitPrice' column shows the price in the specified currency format.



Figure 62: Formatted Grid Data Bound Grid

### 3.1.5 Lesson 4: Virtual Grid Tutorial

A virtual grid is one where the grid does not hold any data. All the data that is displayed by the grid is provided on demand from some external data source to the grid when it needs it. Virtual grids are ideal for displaying large amounts of data which are already stored in some manner. This data is not moved from its original location or stored in **GridStyleInfo** objects. Instead, **GridInfoStyle** objects are created on the fly, to temporarily hold only the necessary data and are discarded when they are no longer needed. There is no data stored in the grid.

Implementing a virtual grid is straight forward. Depending upon the functionality that you need, you can implement a virtual grid with as few as three events. To implement a virtual grid, you must tell the grid how many rows and columns your data source has, and provide the grid the data from your data source. You must do these things in real time, only when the grid requests these data elements. When the grid needs to know the number of rows in the grid, it will fire the **QueryRowCount** event. When it needs to know the number of columns in the grid, it will fire the **QueryColCount** event. When it needs to know a **GridStyleInfo** object for a particular cell, it will fire the **QueryCellInfo** event. By handling these events and setting appropriate members of the **EventArgs**, you are providing the information that the grid needs at the time when it needs it.

In this section, you will learn how to set up an external data source, and then display it by using a virtual grid. The first iteration will allow the display of the external data source; a second iteration will add code that will allow you to edit the displayed data in the virtual grid, pushing the changes back to your data source.

In this lesson, you will learn about the following topics.

### 3.1.5.1 Creating the Project and Data Source

1. In Visual Studio .NET, open **File** menu and select **New Project**. Then using either VB.NET or C#, select the Windows Application project template to create a new Windows Forms project, naming it VirtualGridTutorial.

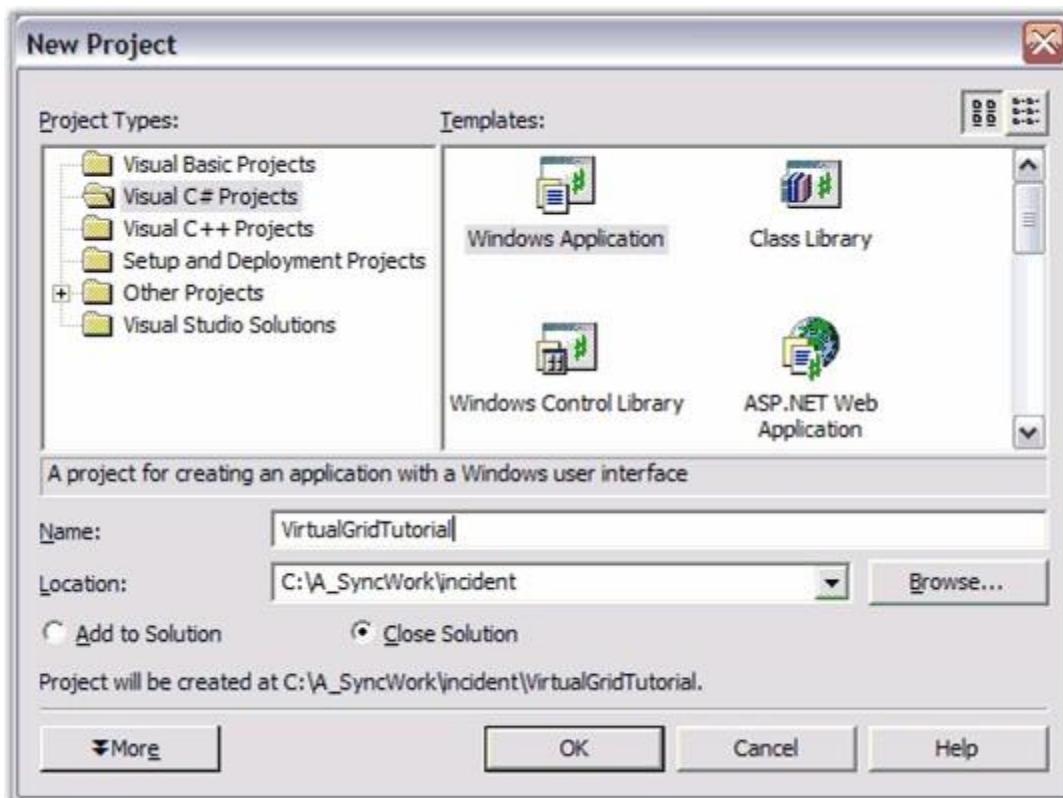


Figure 63: Creating New Project

#### 2. Create the data source

You can set any external data source that knows how to return a value based on the row-column parameters that are passed to it. In addition, the external data source should have

knowledge of the number of rows and columns. This last requirement can be relaxed, but if the data source knows the number of rows and columns, it simplifies things. Our tutorial will assume that this is the case.

The external data source will be a class with two public properties (RowCount and ColCount) and a two-parameter class indexer that will return the integer value that is associated with the two integers (row and column) that are passed in as the indexes. The exact mechanics of this class are not material to this tutorial. This class simply provides the data on demand. How it stores it or how it gets it, plays no role in a virtual grid.

3. To add the **DataSource** class, right-click the project in the **Solution Explorer**, point to **Add**, click **Class**, and add a class named 'ExternalData'.
4. Copy the following code into this file. Notice that the constructor accepts a row and column count, and then populates an integer array. You can modify this class in any way you like, so long as you define the class indexer and the **RowCount** and **ColCount** properties, so that the virtual grid can access the data when needed.

```
[C#]

public class ExternalData
{
    private int _rowCount;
    private int _colCount;
    private int[,] _data;

    public ExternalData(int rows, int cols)
    {
        // Set number of rows and number of columns.
        _rowCount = rows;
        _colCount = cols;

        // Allocate memory to store data values.
        _data = new int[_rowCount, _colCount];

        // Just set the data.
        for(int i = 0; i < RowCount; ++i)
            for(int j = 0; j < ColCount; ++j)
                _data[i,j] = 100 * i + j;
    }

    // Set Properties.
    public virtual int this[int row, int col]
    {
        get{ return _data[row, col]; }
        set{ _data[row, col] = value; }
    }
}
```

```
public virtual int RowCount
{
    get{ return _rowCount; }
}

public virtual int ColCount
{
    get{ return _colCount; }
}
```

**[VB.NET]**

```
Public Class ExternalData
    Private _rowCount As Integer
    Private _colCount As Integer
    Private _data(,) As Int32

    Public Sub New(ByVal rows As Integer, ByVal cols As Integer)
        MyBase.New()

        ' Set number of rows and number of columns.
        _rowCount = rows
        _colCount = cols

        ' Allocate memory to store data values.
        _data = New Integer(_rowCount - 1, _colCount - 1) {}

        ' Just set data.
        Dim i As Integer
        i = 0
        Do While (i < RowCount)
            Dim j As Integer
            j = 0
            Do While (j < ColCount)
                _data(i, j) = ((100 * i) + j)
                j = (j + 1)
            Loop
            i = (i + 1)
        Loop
    End Sub

    ' Set Properties.
    Public Overridable ReadOnly Property RowCount() As Integer
        Get
```

```
    Return _rowCount
End Get
End Property

Public Overridable ReadOnly Property ColCount() As Integer
Get
    Return _colCount
End Get
End Property

Default Public Overridable Property Item(ByVal row As Integer, ByVal
-
                           col As Integer) As Integer
Get
    Return _data(row, col)
End Get

Set(ByVal Value As Integer)
    _data(row, col) = Value
End Set
End Property
End Class
```

### 3.1.5.2 Adding Virtual Grid

To add a Virtual Grid to your application

1. Select the form, open the toolbox and drag a **Grid control** onto your form.



**Note:** Do not change the values of the RowCount or ColCount properties for the grid. Let the default values remain as it is. These values will be provided dynamically as part of the virtual grid implementation.

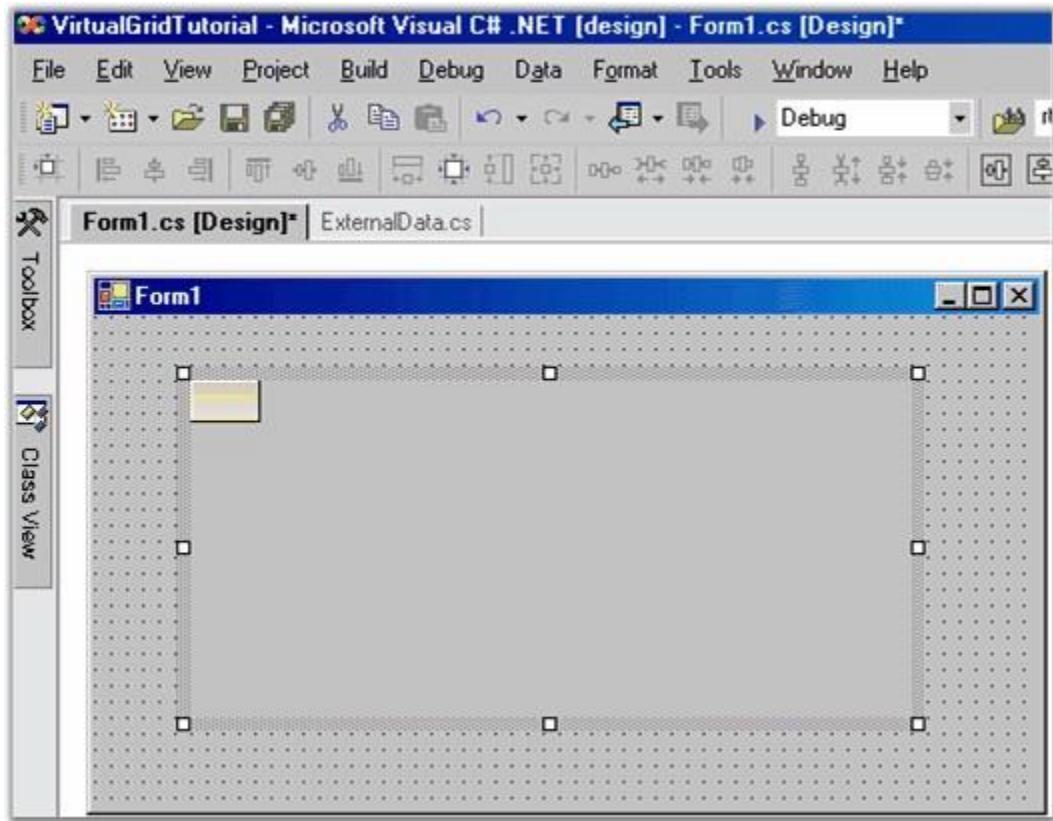


Figure 64: Virtual Grid on Form

2. Customize the other properties such as **BorderStyle**, and so on.

### 3.1.5.3 Initializing the Virtual Grid

To initialize the Virtual Grid added to your application

1. Double-click the form's background so that a handler for the form's load event is added to your code.
2. Add an ExternalData member to your form with the code given below.

[C#]

```
// Add an external data member.  
private ExternalData _extData;
```

[VB .NET]

```
' Add an external data member.  
Private _extData As ExternalData
```

3. Then in your **Form1\_Load** handler, add the following code to initialize the external data source and hook up the Grid control events, so that the grid can use the external data source to get the data which is in demand. The events of interest are **GridControl.QueryRowCount**, **GridControl.QueryColCount** and **GridControl.QueryCellInfo**. The call to **ResetVolatileData** tells the grid that it needs to reset properties like the **RowCount** and **ColCount** the next time they are needed. This will allow the event handlers to set these values.

**[C#]**

```
private void Form1_Load(object sender, System.EventArgs e)  
{  
    // Create a new external data source with 100 rows and 20 columns.  
    this._extData = new ExternalData(100, 20);  
  
    // Prepare the grid for virtual data.  
    gridControl1.ResetVolatileData();  
  
    // Hook up the events needed for the virtual grid.  
    gridControl1.QueryCellInfo += new  
        GridQueryCellInfoEventHandler(GridQueryCellInfo);  
    gridControl1.QueryRowCount += new  
        GridRowColCountEventHandler(GridQueryRowCount);  
    gridControl1.QueryColCount += new  
        GridRowColCountEventHandler(GridQueryColCount);  
}
```

**[VB .NET]**

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs)  
  
    ' Create a new external data source with 100 rows and 20 columns.  
    Me._extData = New ExternalData(100, 20)  
  
    ' Prepare the grid for virtual data.  
    gridControl1.ResetVolatileData()  
  
    ' Hook up the events needed for the virtual grid.  
    ' While only the QueryCellInfo is absolutely required, it would be  
    ' unusual not to handle at least one of the count events.  
    AddHandler gridControl1.QueryCellInfo, New
```

```
_GridQueryCellInfoEventHandler(AddressOf GridQueryCellInfo)
AddHandler gridControl1.QueryRowCount, New
_GridRowColCountEventHandler(AddressOf GridQueryRowCount)
AddHandler gridControl1.QueryColCount, New
_GridRowColCountEventHandler(AddressOf GridQueryColCount)
End Sub
```

### 3.1.5.4 Style Properties

In your **GridQueryCellInfo** handler, it is possible to set the style properties other than the **CellValue**. For example, the code that follows will color any value that is divisible by three.

To set properties other than the **CellValue**, change your **QueryCellInfo** event handler as shown below.

#### [C#]

```
void GridQueryCellInfo(object sender, GridQueryCellInfoEventArgs e)
{
    if (e.RowIndex > 0 && e.ColIndex > 0)
    {
        // Set Cell Value.
        e.Style.CellValue = this._extData[e.RowIndex - 1, e.ColIndex - 1];

        // Apply conditional formatting.
        if(this._extData[e.RowIndex - 1, e.ColIndex - 1] % 3 == 0)
            e.Style.BackColor = Color.LightPink;
        e.Handled = true;
    }
}
```

#### [VB.NET]

```
Private Sub GridQueryCellInfo(ByVal sender As Object, ByVal e As GridQueryCellInfoEventArgs)
    If ((e.RowIndex > 0) AndAlso (e.ColIndex > 0)) Then

        ' Set Cell Value.
        e.Style.CellValue = Me._extData(e.RowIndex - 1, e.ColIndex - 1)

        ' Apply conditional formatting.
```

```

If ((Me._extData(e.RowIndex - 1, e.ColumnIndex - 1) Mod 3) = 0) Then
    e.Style.BackColor = Color.LightPink
End If
e.Handled = True
End If
End Sub

```

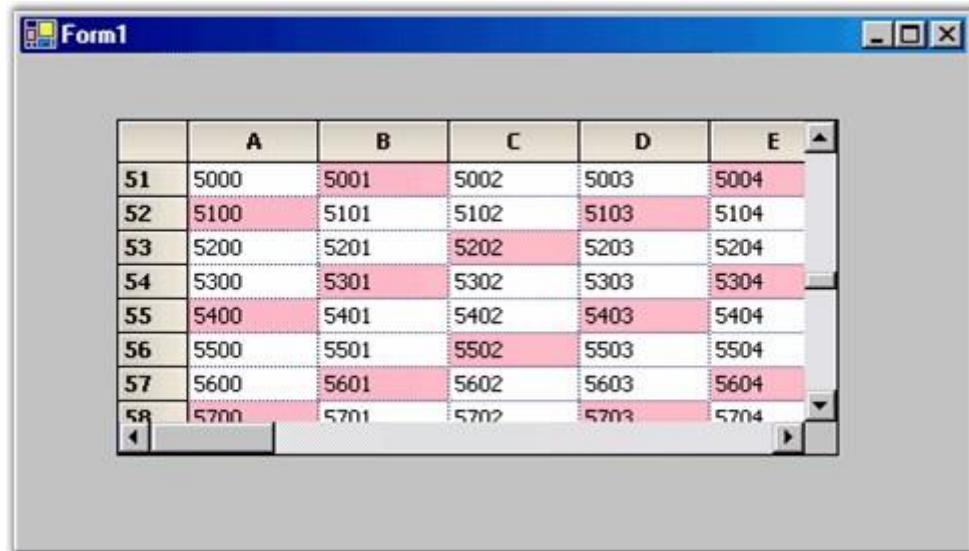


Figure 65: Virtual Grid with Back Color set for Cells

### 3.1.5.5 Handling Events to Retrieve Data for Virtual Grid

To retrieve data for the Virtual Grid from the external data source

1. Add event handlers for the **QueryRowCount** and **QueryColCount** events. Use the following code in your event handlers.

The **GridQueryRowCount** and **GridQueryColCount** provide the numbers of rows and columns from the external data source. Thus, the implementation code will access the public properties of our external data object to get these values.

[C#]

```

// Set number of rows from external data source.
void GridQueryRowCount(object sender, GridRowColCountEventArgs e)
{
    e.Count = this._extData.RowCount;
}

```

```
        e.Handled = true;
    }

// Set number of columns from external data source.
void GridQueryColCount(object sender, GridRowColEventArgs e)
{
    e.Count = this._extData.ColCount;
    e.Handled = true;
}
```

**[VB.NET]**

```
' Set number of rows from external data source.
Private Sub GridQueryRowCount(ByVal sender As Object, ByVal e _As
GridRowColEventArgs)
    e.Count = Me._extData.RowCount
    e.Handled = True
End Sub

' Set number of columns from external data source.
Private Sub GridQueryColCount(ByVal sender As Object, ByVal e _As
GridRowColEventArgs)
    e.Count = Me._extData.ColCount
    e.Handled = True
End Sub
```

2. Add a **QueryCellInfo** event handler.

The **GridQueryCellInfo** is the event handler where the grid expects the external data source to provide the cell values that are in demand. Here is how it can be implemented with the external data source. Recall that row 0 and column 0 are usually the header columns in a grid. In the GridQueryCellInfo, do not provide these values and use the default headers. If you need to provide special header values, you can do so.

**[C#]**

```
void GridQueryCellInfo(object sender, GridQueryCellInfoEventArgs e)
{
    if (e.RowIndex > 0 && e.ColumnIndex > 0)
    {
        e.Style.CellValue = this._extData[e.RowIndex - 1, e.ColumnIndex - 1];
        e.Handled = true;
    }
}
```

**[VB.NET]**

```
Private Sub GridQueryCellInfo(ByVal sender As Object, ByVal e As GridQueryCellInfoEventArgs)
    If ((e.RowIndex > 0) AndAlso (e.ColumnIndex > 0)) Then
        e.Style.CellValue = Me._extData(e.RowIndex - 1, e.ColumnIndex - 1)
        e.Handled = True
    End If
End Sub
```

Notice that all three handlers set the **Handled** property on the **EventArgs** when a value is provided. This informs the grid that no further processing is needed. Don't forget this or you'll lose some of the benefits of using a virtual grid.

3. Compile and run the project. You will see something similar to the screen shot given below. The point is that the grid itself does not hold any data at all. All the data information is being provided on demand through the three events that you have added.

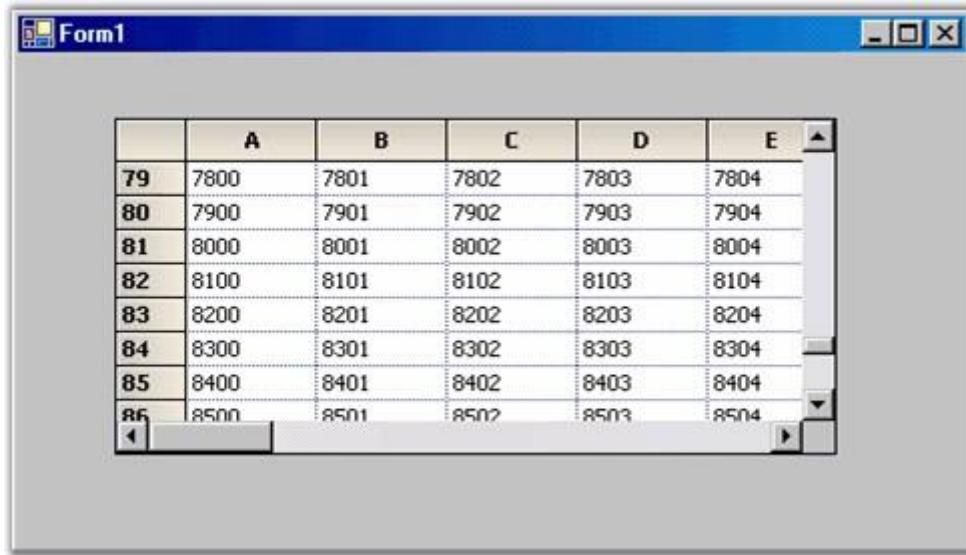


Figure 66: Virtual Grid Holds No Internal Data

### 3.1.5.6 Saving Edited Values

While working on your Virtual Grid, you will see that the changes do not stay around after you leave the current cell, i.e., if you over type a cell entry, when you move off the cell, the old value is restored. The reason is that currently, there is no way for the changed value to be moved back to the external data source. So, as you move off the cell, when the grid redraws the old cell, it will query the external data source (through `GridQueryCellInfo`), get the original value and display it. The value which you have typed will be lost.

In order to edit and save values in the Virtual Grid, you must be able to get the changed value back to the external data source. This is accomplished through the `GridControl.SaveCellInfo` event. You must add a handler for this event and in this handler, you must save the changed value back to the external data source.

**[C#]**

```
private void Form1_Load(object sender, System.EventArgs e)
{
    // Create a new external data source with 100 rows and 20 columns.
    this._extData = new ExternalData(100, 20);

    // Hook up the events needed for the virtual grid.
    gridControl1.ResetVolatileData();
    gridControl1.QueryCellInfo += new GridQueryCellInfoEventHandler(
        GridQueryCellInfo);
    gridControl1.QueryRowCount += new GridRowColCountEventHandler(
        GridQueryRowCount);
    gridControl1.QueryColCount += new GridRowColCountEventHandler(
        GridQueryColCount);

    // Handle saving data back to the data source.
    gridControl1.SaveCellInfo += new GridSaveCellInfoEventHandler(
        GridSaveCellInfo);
}

void GridSaveCellInfo(object sender, GridSaveCellInfoEventArgs e)
{
    try
    {
        // Move the changes back to the external data object.
        if( e.ColumnIndex > 0 && e.RowIndex > 0)
        {
            this._extData[e.RowIndex - 1, e.ColumnIndex - 1] =
                int.Parse(e.Style.CellValue.ToString());
        }
    }
    catch{}
    e.Handled = true;
}
```

[VB.NET]

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs)
    ' Create a new external data source with 100 rows and 20 columns.
    Me._extData = New ExternalData(100, 20)

    ' Prepare the grid for virtual data.
    gridControl1.ResetVolatileData()

    ' Hook up the events needed for the virtual grid.
    ' While only the QueryCellInfo is absolutely required, it would be
    unusual not to handle at least one of the count events.
    AddHandler gridControl1.QueryCellInfo, New
        _GridQueryCellInfoEventHandler(AddressOf GridQueryCellInfo)
    AddHandler gridControl1.QueryRowCount, New
        _GridRowColCountEventHandler(AddressOf GridQueryRowCount)
    AddHandler gridControl1.QueryColCount, New
        _GridRowColCountEventHandler(AddressOf GridQueryColCount)

    ' Handle saving data back to the data source.
    AddHandler gridControl1.SaveCellInfo, New
        _GridSaveCellInfoEventHandler(AddressOf GridSaveCellInfo)
End Sub

Private Sub GridSaveCellInfo(ByVal sender As Object, ByVal e As
    GridSaveCellInfoEventArgs)
    Try

        ' Move the changes back to the external data object.
        If ((e.ColumnIndex > 0) AndAlso (e.RowIndex > 0)) Then
            Me._extData((e.RowIndex - 1), (e.ColumnIndex - 1)) =
                _System.Int32.Parse(e.Style.CellValue.ToString)
        End If
    Catch ex As System.Exception
    End Try
    e.Handled = True
End Sub
```

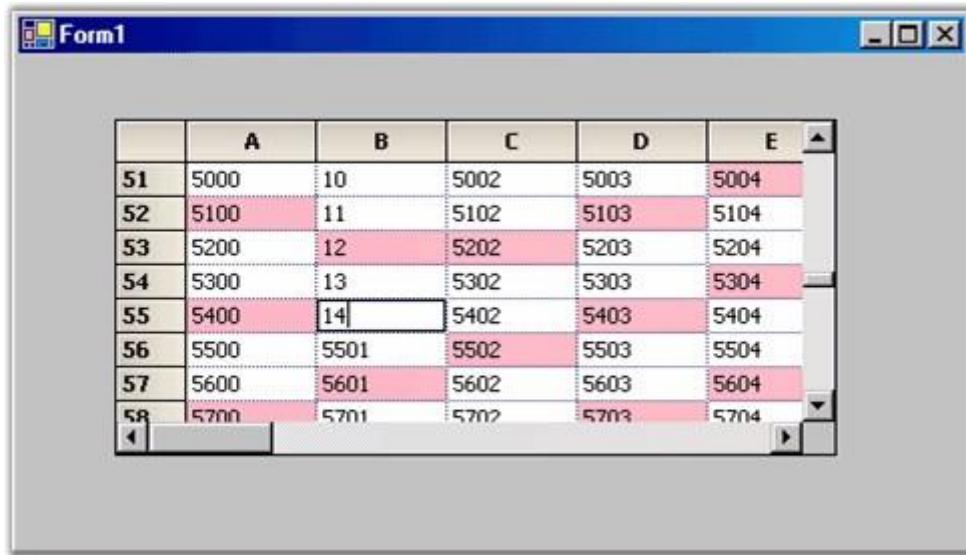


Figure 67: Editable Virtual Grid

### 3.1.5.7 Setting Properties in a Virtual Grid

So far you have seen that you can provide any **GridStyleInfo** property in a virtual manner by merely handling the Grid control's **QueryCellInfo** event. You can also provide the row count and column count in a virtual manner as well.

Here is a list of the other events that will allow virtual access to an Essential Grid.

- **QueryRowHeight**-This event allows you to dynamically provide row heights.
- **QueryColWidth**-This event allows you to dynamically provide column widths.
- **QueryCoveredRange**-This event allows you to dynamically provide covered cell ranges.

[C#]

```
// Provide the row heights on demand - optional...
void GridQueryRowHeight(object sender, GridRowColSizeEventArgs e)
{
    if( e.Index % 2 == 0 )
    {
        e.Size = 20;
        e.Handled = true;
    }
}
```

```
// Provide the col widths on demand - optional...
void GridQueryColWidth(object sender, GridRowColSizeEventArgs e)
{
    if( e.Index % 3 == 0)
    {
        e.Size = 40;
        e.Handled = true;
    }
}

// Provide covered range on demand - optional...
void GridQueryCoveredRange(object sender, GridQueryCellRangeEventArgs e)
{
    // Cover odd rows, columns 1 through 3.
    if (e.RowIndex % 2 == 1 && e.ColumnIndex >= 1 && e.ColumnIndex <= 3)
    {
        e.Range = GridRangeInfo.Cells(e.RowIndex, 1, e.RowIndex, 3);
        e.Handled = true;
    }

    // Cover column 6 with odd-even row pairs.
    if (e.RowIndex > 0 && e.ColumnIndex == 6)
    {
        int row = (e.RowIndex-1) /2 * 2 + 1;
        int col = e.ColumnIndex;
        e.Range = GridRangeInfo.Cells(row, col, row+1, col);
        e.Handled = true;
    }
}
```

**[VB.NET]**

```
Private Sub GridQueryRowHeight(ByVal sender As Object, ByVal e As
_GridRowColSizeEventArgs)
    If ((e.Index Mod 2) = 0) Then
        e.Size = 20
        e.Handled = True
    End If
End Sub

Private Sub GridQueryColWidth(ByVal sender As Object, ByVal e As
_GridRowColSizeEventArgs)
    If ((e.Index Mod 3) = 0) Then
        e.Size = 40
    End If
End Sub
```

```
    e.Handled = True
End If
End Sub

Private Sub GridQueryCoveredRange(ByVal sender As Object, ByVal e As
_GridQueryCellRangeEventArgs)

    ' Cover odd rows, columns 1 through 3.
    ' Cover column 6 with odd-even row pairs.
    If (((e.RowIndex Mod 2) = 1) AndAlso (e.ColumnIndex >= 1)) _AndAlso
(e.ColumnIndex <= 3)) Then
        e.Range = GridRangeInfo.Cells(e.RowIndex, 1, e.RowIndex, 3)
        e.Handled = True
    End If
    If ((e.RowIndex > 0) AndAlso (e.ColumnIndex = 6)) Then
        Dim row As Integer
        row = (((e.RowIndex - 1) / 2) * 2) + 1
        Dim col As Integer
        col = e.ColumnIndex
        e.Range = GridRangeInfo.Cells(row, col, (row + 1), col)
        e.Handled = True
    End If
End Sub
```

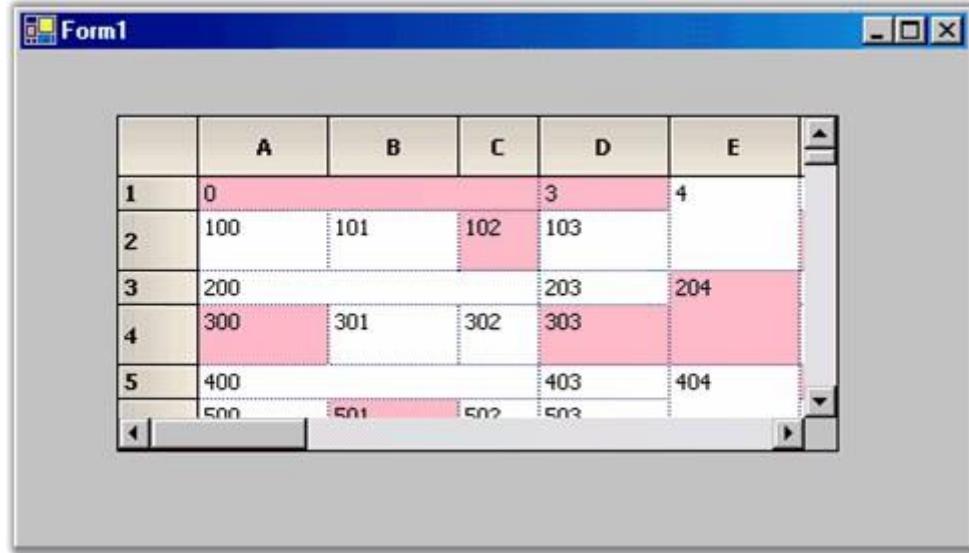


Figure 68: Grid with Column Widths, Row Heights and Covered Cells Provided Dynamically

### 3.1.5.8 Type Conversions

You will notice that in the **GridSaveCellInfo** method, you had made use of the **int.Parse** method to convert the string value in the **GridStyleInfo** object to the integer you needed for the external data source. You can instead make use of the more general **Convert** class provided by Essential Grid, to handle conversions between various data types. This class, for example, can convert the value in a **CellValue** property to a **DateTime** object, or to a **Color** object, depending upon the need. The following code example illustrates how to use this Convert class.

**[C#]**

```
// Convert a value in CellValue property to 'int' (as required by our
// data source) by using Convert class.
this._extData[e.RowIndex - 1, e.ColumnIndex - 1] =
(int)GridCellValueConvert.ChangeType(e.Style.CellValue, typeof(int),
null);
```

**[VB .NET]**

```
' Convert a value in CellValue property to 'int' (as required by our
// data source) by using Convert class.
Me._extData(e.RowIndex - 1, e.ColumnIndex - 1) =
CInt(GridCellValueConvert.ChangeType(e.Style.CellValue,
GetType(Integer), Nothing))
```



**Note:** This conversion problem may occur when the value that is stored in the style object is a string. This happens when the **CellValueType** property is not explicitly set on the style object in your **GridQueryCellInfo** method. But when this is set to "int", then you can cast the **CellValue** in **SaveCellInfo** to an int, and do not have to worry about conversions.

### 3.1.6 Lesson 5: Excel Export

Export to Excel is one of the most common functionalities that are required in the .NET world. The Essential Grid control has built-in support for Excel Export. You can download the data from the Grid control or Grid Data Bound Grid or Grouping Grid control into an Excel spreadsheet for offline verification and/or computation. This can be achieved by making use of the **GridExcelConverter** and **GroupingGridExcelConverter** classes. This section will step you through the conversion of the contents of the grid to an Excel file, as well as discuss the various converter options.

In this lesson, you will learn about the following topics.

### 3.1.6.1 Exporting the Grid Control or Grid Data Bound Grid To Excel

The **GridExcelConverter** class provides support for exporting data from a Grid control or Grid Data Bound Grid into an Excel spreadsheet for verification and/or computation. This control automatically copies the Grid's styles and formats to Excel. The **GridExcelConverter** control is derived from the **GridExcelConverterBase**. The XlsIO libraries support the conversion of Grid content to Excel.

To make use of the GridExcelConverter class, the following assemblies should be added along with the default assemblies present in the **References** folder of your application:  
**Syncfusion.GridConverter.Base** and **Syncfusion.XlsIO.Base**.

The **GridToExcel** method is used to export the grid to an Excel sheet. The following code example illustrates how to convert the Grid content to Excel.

#### [C#]

```
Syncfusion.GridExcelConverter.GridExcelConverterControl gecc = new  
Syncfusion.GridExcelConverter.GridExcelConverterControl();  
gecc.GridToExcel(this.gridControl1.Model, @"C:\MyGC.xls");
```

#### [VB .NET]

```
Dim gecc As Syncfusion.GridExcelConverter.GridExcelConverterControl =  
New Syncfusion.GridExcelConverter.GridExcelConverterControl()  
gecc.GridToExcel(Me.gridControl1.Model, "C:\MyGC.xls")
```

The following code example illustrates how to convert the Grid Data Bound Grid content to Excel.

#### [C#]

```
Syncfusion.GridExcelConverter.GridExcelConverterControl gecc = new  
Syncfusion.GridExcelConverter.GridExcelConverterControl();  
gecc.GridToExcel(this.gridDataBoundGrid1.Model, @"C:\MyGC.xls");
```

#### [VB .NET]

```
Dim gecc As Syncfusion.GridExcelConverter.GridExcelConverterControl =  
New Syncfusion.GridExcelConverter.GridExcelConverterControl()  
gecc.GridToExcel(Me.gridDataBoundGrid1.Model, "C:\MyGC.xls")
```

### 3.1.6.1.1 GridControl to Excel Conversion Process

The conversion of Grid to Excel is done cell by cell. Each cell is converted with respect to its style, including its format and background color. This is done by using the **GridCellToExcel()** method of the **GridExcelConverterControl** class.

#### 3.1.6.1.1.1 Currency Cell Conversion

##### 3.1.6.1.1.1.1 If format specified

The cell is checked whether it is a Currency cell type by using the **CellType** property. If it is a Currency Cell type then the **CellValue** will be converted into Double and saved in Range's **Number** property. **NumberFormatInformation** is extracted from the **NumberFormatInfoObject** and the number is converted to string by using the extracted format. **FormatString** is created and saved in Range's **NumberFormat** property.

The following is a sample code to set the format of the Currency cell:

**[C#]**

```
this.gridControl1[row,col].Format = "C";
```

**[VB]**

```
Me.gridControl1(row,col).Format = "C"
```

##### 3.1.6.1.1.1.2 If format not specified

The cell is checked whether it is a Currency cell type by using the **CellType** property. If the **CellType** is not given then it switches to the default **CellType** and the **CellValue** is stored in Range's "Value2" property, where the value2 is converted into the given **CellValueType**(Except Date Time and Currency). Hence, in this case the **CellValue** will be converted to General format.

#### 3.1.6.1.1.2 Number Cell Conversion

##### If format specified

The cell is checked whether it is a Number cell type by using the **CellType** property. If it is a Number Cell type then the cell value is assigned to Range's **Value2** property. Since the format is specified, the **Value2**'s value is converted to its respective type and set to Number format.

The following is a sample code to set the format of Currency cell:

**[C#]**

```
this.gridControl1[row,col].Format = "N";
```

**[VB]**

```
Me.gridControl1(row,col).Format = "N"
```

#### **3.1.6.1.1.2.2 If format not specified**

The cell is checked whether it is a Number cell type by using the CellType property. If it is not a Number Cell type, then the format is skipped for all cell types and finally the CellValue is assigned to Range's Value2 property. Since the format is not specified, the Value2 value is converted to its respective type and set to General Format.

#### **3.1.6.1.1.3 Image Cell Conversion**

##### **If CellType specified**

The cell is checked whether it is an Image cell type by using the CellType property. If it is an Image Cell type then the CellValue is converted to a bitmap and this bitmap is inserted into the sheet specifying the row and column.

##### **3.1.6.1.1.3.2 If CellType not specified**

Not Applicable

#### **3.1.6.1.1.4 ComboBox and RichText Cell Conversion**

##### **If CellType specified**

The cell is checked whether it is a ComboBox or RichText cell type by using the CellType property. If it is a ComboBox or RichText cell type then the FormattedText is stored in the Range's Text property.

##### **3.1.6.1.1.4.2 If CellType not specified**

Not Applicable

#### **3.1.6.1.1.5 ProgressBar Cell Conversion**

##### **If CellType specified**

The cell is checked whether it is a ProgressBar cell type using CellType property. If it is a ProgressBar cell type then the ProgressBar text style is checked whether it should be in percentage style. If the style is in percentage then the percentage value is calculated and saved to Range's Number property. The Range's NumberFormat is saved as "0%". If there is no style specified, the ProgressValue is directly saved to Range's Number.

##### **3.1.6.1.1.5.2 If CellType not specified**

Not Applicable

#### **3.1.6.1.1.6 DateTime Cell Conversion**

##### **If format specified**

The cell is checked whether it is a DateTime cell type using the CellType property. If it is a DateTime cell type, the format is skipped for all cell types and finally the CellValue is assigned to Range's DateTime property. Since the format is specified, the DateTime value is converted to DateTime type and the format is set to the DateTime format in Range's Format.

##### **3.1.6.1.1.6.2 If format not specified**

The cell is checked whether it is a DateTime cell type using the CellType property. If it is not a DateTime cell type, the format is skipped for all cell types and finally the CellValue is assigned to Range's DateTime property. Since the format is not specified, the DateTime value is converted to DateTime type and set to General Format.

#### 3.1.6.1.1.7 Formula Cell Conversion

##### If Format specified

The cell is checked whether it is a Formula cell type using the **CellType** property. If it is a Formula cell type then the **GridCell**'s **Text** string is stored to Range's **Formula** property. The values in Excel cells can be formatted by specifying the format in Formula cell types.

##### 3.1.6.1.1.7.2 If Format not specified

If no formula is specified for the cell then the formula is set to Excel sheet's range **Formula** property with its default format.

## 3.1.6.2 Import an Excel Sheet into Essential Grid

An Excel sheet can also be imported to the Grid control or Grid Data Bound Grid. This can be done by using the **ExcelToGrid** method in the **GridExcelConverterControl** class.

The following code example illustrates how to transfer Excel content to the Grid control.

### [C#]

```
Syncfusion.GridExcelConverter.GridExcelConverterControl gecc = new  
Syncfusion.GridExcelConverter.GridExcelConverterControl();  
gecc.ExcelToGrid(@"C:\MyGC.xls", this.gridControl1.Model);
```

### [VB .NET]

```
Dim gecc As Syncfusion.GridExcelConverter.GridExcelConverterControl =  
New Syncfusion.GridExcelConverter.GridExcelConverterControl()  
gecc.ExcelToGrid("C:\MyGC.xls", Me.gridControl1.Model)
```

The following code example illustrates how to transfer Excel content to the Grid Data Bound Grid.

### [C#]

```
Syncfusion.GridExcelConverter.GridExcelConverterControl gecc = new  
Syncfusion.GridExcelConverter.GridExcelConverterControl();  
gecc.ExcelToGrid(@"C:\MyGC.xls", this.gridDataBoundGrid1.Model);
```

### [VB .NET]

```
Dim gecc As Syncfusion.GridExcelConverter.GridExcelConverterControl =  
New Syncfusion.GridExcelConverter.GridExcelConverterControl()  
gecc.ExcelToGrid("C:\MyGC.xls", Me.gridDataBoundGrid1.Model)
```

### 3.1.6.3 Excel Converter Options

The **GridExcelConverter** class enables you to export specific grid elements like column headers, row headers, and so on. By default, the **GridExcelConverterControl** exports all the elements in the grid.

The following code example illustrates how to include both row and column headers during the export.

[C#]

```
gecc.GridToExcel(this.grid.Model, @"C:\MyGGC.xls",
Syncfusion.GridExcelConverter.ConverterOptions.RowHeaders |
Syncfusion.GridExcelConverter.ConverterOptions.ColumnHeaders);
```

[VB .NET]

```
gecc.GridToExcel(Me.grid.Model, "C:\MyGGC.xls",
Syncfusion.GridExcelConverter.ConverterOptions.RowHeaders | Syncfusion.GridExcelConverter.ConverterOptions.ColumnHeaders)
```

### 3.1.6.4 Exporting Grid Grouping Data To Excel

The **GroupingGridExcelConverter** class provides support for exporting data from a Grouping Grid control into an Excel spreadsheet for verification and/or computation. This control automatically copies the Grid's styles, formats, groups, summary rows and expression fields to Excel. The **GroupingGridExcelConverter** control is derived from the **GridExcelConverterBase**. The XlsIO libraries support the conversion of Grid content to Excel.

To make use of the GroupingGridExcelConverter class, the following assemblies must be added along with the default assemblies present in the **References** folder of your application: **Syncfusion.XlsIO.Base** and **Syncfusion.GridConverter.Windows**.

The content of the Grid Grouping control can be transferred to Excel by using the **GroupingGridToExcel** method in the **GroupingGridExcelConverterControl** class. There are two export options provided by the Grid Grouping control: first option converts the entire content in the grid to Excel, and the second option converts only the visible content in the grid to Excel.

The following code example illustrates how to convert the entire Grid content to Excel.

**[C#]**

```
Syncfusion.GroupingGridExcelConverter.GroupingGridExcelConverterControl  
converter = new  
Syncfusion.GroupingGridExcelConverter.GroupingGridExcelConverterControl()  
();  
converter.GroupingGridToExcel(this.gridGroupingControl1,  
@"C:\MyGGC.xls",  
Syncfusion.GridExcelConverter.ConverterOptions.Default);
```

**[VB .NET]**

```
Dim converter As  
Syncfusion.GroupingGridExcelConverter.GroupingGridExcelConverterControl  
= New  
Syncfusion.GroupingGridExcelConverter.GroupingGridExcelConverterControl()  
converter.GroupingGridToExcel(Me.gridGroupingControl1, "C:\MyGGC.xls",  
Syncfusion.GridExcelConverter.ConverterOptions.Default);
```

You can export the visible, or expanded records or groups alone by using the following code.

**[C#]**

```
converter.GroupingGridToExcel(this.gridGroupingControl1,  
@"C:\MyGGC.xls",  
Syncfusion.GridExcelConverter.ConverterOptions.Visible);
```

**[VB .NET]**

```
converter.GroupingGridToExcel(this.gridGroupingControl1,  
"C:\MyGGC.xls",  
Syncfusion.GridExcelConverter.ConverterOptions.Visible);
```

### 3.1.6.5 Exporting Multiple Grids

It is possible to save multiple grids to a single XLS file as worksheets. The following code example illustrates how to do this.

**[C#]**

```
using Syncfusion.XlsIO;  
using Syncfusion.GridExcelConverter;
```

```
private void buttonExport_Click(object sender, System.EventArgs e)
{
    SaveFileDialog saveFileDialog = new SaveFileDialog();
    saveFileDialog.Filter = "Files (*.XLS) | *.XLS";
    saveFileDialog.AddExtension = true;
    saveFileDialog.DefaultExt = ".XLS";

    if(saveFileDialog.ShowDialog() == DialogResult.OK &&
    saveFileDialog.CheckPathExists)
    {
        GridExcelConverterControl gec = new
        GridExcelConverterControl();
        IWorkbook workBook = ExcelUtils.CreateWorkbook(new string[]
        {"Sheet1", "Sheet2"});
        gec.GridToExcel(this.gridControl1.Model,
        workBook.Worksheets[0]);
        gec.GridToExcel(this.gridControl2.Model,
        workBook.Worksheets[1]);
        workBook.SaveAs(saveFileDialog.FileName);
        workBook.Close();
        ExcelUtils.ThrowNotSavedOnDestroy = false;
    }
}
```

**[VB.NET]**

```
Imports Syncfusion.XlsIO
Imports Syncfusion.GridExcelConverter

Private Sub buttonExport_Click(ByVal sender As Object, ByVal e As
System.EventArgs)
    Dim saveFileDialog As SaveFileDialog = New SaveFileDialog()
    saveFileDialog.Filter = "Files (*.XLS) | *.XLS"
    saveFileDialog.AddExtension = True
    saveFileDialog.DefaultExt = ".XLS"
    If saveFileDialog.ShowDialog() = DialogResult.OK And Also
    saveFileDialog.CheckPathExists Then
        Dim gec As GridExcelConverterControl = New
        GridExcelConverterControl
        Dim workbook As IWorkbook = ExcelUtils.CreateWorkbook(New
        String() {"Sheet1", "Sheet2"})
        gec.GridToExcel(Me.gridControl1.Model, workBook.Worksheets(0))
        gec.GridToExcel(Me.gridDataBoundGrid1.Model,
        workbook.Worksheets(1))
        workbook.SaveAs(saveFileDialog.FileName)
        workBook.Close()
        ExcelUtils.ThrowNotSavedOnDestroy = False
    End If
End Sub
```

```
End If  
End Sub
```

### 3.1.6.6 Exporting Grid Data to Excel 2010

You can export Grid data to Excel 2010. To do this, you must explicitly set the Excel version by creating an XlsIO application, and then export the Grid data to the Excel worksheet.

You can set the **DefaultVersion** to *Excel2010* to export to this version, by default. Similarly, you can also export Grid data to other versions of Excel.

The following code example illustrates this.

```
[C#]  
  
ExcelEngine engine = new ExcelEngine();  
  
IApplication app = engine.Excel.Application;  
  
app.DefaultVersion = ExcelVersion.Excel2010;  
  
IWorkbook book = app.Workbooks.Create();  
  
GroupingGridExcelConverterControl gecc = new  
GroupingGridExcelConverterControl();  
  
SaveFileDialog saveFileDialog = new SaveFileDialog();  
  
saveFileDialog.Filter = "Files(*.xlsx)|*.xlsx";  
  
saveFileDialog.DefaultExt = ".xlsx";  
  
if (saveFileDialog.ShowDialog() ==  
System.Windows.Forms.DialogResult.OK)  
{  
  
    gecc.GroupingGridToExcel(this.gridGroupingControl1, book.Worksheets(0),  
    ConverterOptions.Visible);  
  
    book.SaveAs(saveFileDialog.FileName);  
  
    if (MessageBox.Show("Do you wish to open the xls file now?", "Export  
    to Excel", MessageBoxButtons.YesNo, MessageBoxIcon.Question) ==  
    System.Windows.Forms.DialogResult.Yes)  
    {  
  
        Process proc = new Process();  
  
        proc.StartInfo.FileName = saveFileDialog.FileName;
```

```
proc.Start();  
}  
}
```

**[VB.NET]**

```
Dim engine As New ExcelEngine()  
Dim app As IApplication = engine.Excel.Application  
app.DefaultVersion = ExcelVersion.Excel2010  
Dim book As IWorkbook = app.Workbooks.Create()  
Dim gecc As New GroupingGridExcelConverterControl()  
Dim saveFileDialog As New SaveFileDialog()  
saveFileDialog.Filter = "Files(*.xlsx)|*.xlsx"  
saveFileDialog.DefaultExt = ".xlsx"  
If saveFileDialog.ShowDialog() = System.Windows.Forms.DialogResult.OK  
Then  
    gecc.GroupingGridToExcel(Me.gridGroupingControll, book.Worksheets(0),  
    ConverterOptions.Visible)  
    book.SaveAs(saveFileDialog.FileName)  
    If MessageBox.Show("Do you wish to open the xls file now?", "Export  
    to Excel", MessageBoxButtons.YesNo, MessageBoxIcon.Question) =  
    System.Windows.Forms.DialogResult.Yes Then  
        Dim proc As New Process()  
        proc.StartInfo.FileName = saveFileDialog.FileName  
        proc.Start()  
        End If  
    End If
```



**Note:** This is applicable to all Grid controls in Essential Grid.

### 3.1.7 Lesson 6: Accessibility Information for Grid Windows Forms

#### Technical Standards

Software Applications and Operating Systems – Detail Voluntary Product Accessibility Template

Criteria	Supporting Features	Remarks	Explanations
(a) When software is designed to run on a system that has a keyboard, product functions shall be executable from a keyboard where, the function itself or the result of performing a function can be discerned textually.	Fully Supported	Essential Grid supports keyboard navigation and the editing of text.	Events can be handled in order to have all the keys to be suppressed and simulated. The events could be made to be dependent on the cell or over the grid.
(b) Applications shall not disrupt or disable activated features of other products that are identified as accessibility features, where those features are developed and documented according to industry standards. Applications also shall not disrupt or disable activated features of any operating system that are identified as accessibility features where the application programming interface for those accessibility features has been documented by the manufacturer of the operating system and is available to the product developer.	Fully Supported	Essential Grid component can be placed independent to other control, so that no other products or items on the operating system could be disrupted or disabled.	Unless the grid is bound to the shared data source with external control, the grid does not affect the attributes over the other. Refreshing the form can also be specified within the bounds of the grid object.

Criteria	Supporting Features	Remarks	Explanations
(c) A well-defined on-screen indication of the current focus shall be provided that moves among interactive interface elements as the input focus changes. The focus shall be programmatically exposed so that Assistive Technology can track focus and focus changes.	Fully Supported	The focusing indication is applied to the individual cells.	The border of the current cell can be highlighted if the grid has the focus on the specific cell.
(d) Sufficient information about a user interface element including the identity, operation and state of the element shall be available to Assistive Technology. When an image represents a program element, the information conveyed by the image must also be available in text.	Not Applicable	It is application-oriented, where the information to be provided on an individual component depends on the event of the grid.	
(e) When bitmap images are used to identify controls, status indicators, or other programmatic elements, the meaning assigned to those images shall be	Fully Supported	Supports on design time to identify the control.	The dependent property has been provided, where the property window will identify the control and specify its own attributes.

Criteria	Supporting Features	Remarks	Explanations
consistent throughout an application's performance.			
(f) Textual information shall be provided through operating system functions for displaying text. The minimum information that shall be made available is text content, text input caret location, and text attributes.	Fully Supported	The runtime support is also provided in order to change the text and its style. The format could be modified with the settings.	The validation on the text with the runtime is fully supported. The Format dialog can be wired to the cell in order to change the settings. From the caret position, the nearest character can be obtained from the underlying text.
(g) Applications shall not override user selected contrast and color selections and other individual display attributes.	Fully Supported	The settings handled in the grid will not be affected by the application.	Supports with the DPI setting, also the contrast does not affect with the variations.
(h) When animation is displayed, the information shall be displayable in at least one non-animated presentation mode at the option of the user.	Not Applicable	This behavior is application oriented, where the animation provided in the cells will be in specific bounds and the rest of the area can be used to provide non-animated content.	
(i) Color coding shall not be used as the only means of conveying information, indicating an action, prompting a response, or	Fully Supported	Essential Grid controls provide complete functionality that conforms to the	Essential Grid abide by the default validation rules and constraints such that, while entering invalid values in cells, along with red color being highlighted, it prompts the respective system error message, which requires

Criteria	Supporting Features	Remarks	Explanations
distinguishing a visual element.		criteria.	the user to rectify the invalid value to proceed further.
(j) When a product permits a user to adjust color and contrast settings, a variety of color selections capable of producing a range of contrast levels shall be provided.	Fully Supported	Fully supported for Essential Grid controls internally.	Depends on the component base of elements with default setting with the operating system. The window control acts to the specification, and provides a solid display with various contrasts.
(k) Software shall not use flashing or blinking text, objects, or other elements having a flash or blink frequency greater than 2 Hz and lower than 55 Hz.	Not Applicable	The blinking support is provided in the Essential Grid. But the frequency on data update has to be provided by the user at their end based on the need.	
(l) When electronic forms are used, the form shall allow people using Assistive Technology to access the information, field elements, and functionality required for completion and submission of the form, including all directions and cues.	Not Applicable	This criterion does not apply to Essential Grid controls.	
(m) At least one mode of operation and information retrieval that does not require user vision shall be provided, or support for	Fully Supported	-	Narrator support Essential Grid satisfies this compliance in all scenarios except when we navigate the cells using the arrow keys in the keyboard. The cell information

Criteria	Supporting Features	Remarks	Explanations
Assistive Technology used by people who are blind or visually impaired shall be provided.			cannot be detected through the 'Narrator'. However, we can rectify this by exposing the information through handling the tooltip for the cells in the grid.
(n) Where audio information is important for the use of a product, at least one mode of operation and information retrieval shall be provided in an enhanced auditory fashion, or support for assistive hearing devices shall be provided.	Not applicable	Not applicable	
(o) At least one mode of operation and information retrieval that does not require user speech shall be provided, or support for Assistive Technology used by people with disabilities shall be provided.	Fully Supported	Fully Supported	Can be achieved through existing cell-related APIs without requiring user speech.
(p) At least one mode of operation and information retrieval that does not require fine motor control or simultaneous actions, and that is operable with limited reach and	Fully Supported	Essential Grid controls provide functionality that conform to the criteria.	Can be achieved through existing cell-related APIs.

<b>Criteria</b>	<b>Supporting Features</b>	<b>Remarks</b>	<b>Explanations</b>
strength shall be provided.			

**Functional Performance Criteria**

<b>Criteria</b>	<b>Supporting Features</b>	<b>Remarks and explanations</b>
(a) At least one mode of operation and information retrieval that does not require user vision shall be provided, or support for Assistive Technology used by people who are blind or visually impaired shall be provided.	Fully Supported	Syncfusion Essential Windows Grid satisfies the requirements for the blind or visually-impaired in several operations.
(b) At least one mode of operation and information retrieval that does not require visual acuity greater than 20/70 shall be provided in audio and enlarged print output working together or independently, or support for Assistive Technology used by people who are visually impaired shall be provided.	Not Applicable	This criterion does not apply to Essential Grid controls.
(c) At least one mode of operation and information retrieval that does not require user -hearing shall be provided, or support for Assistive Technology used by people who are deaf or hard of hearing shall be provided	Fully Supported	Essential Grid controls have the support for retrieving information which doesn't require user hearing.
(d) Where audio information is important for the use of a product, at least one mode of operation and information retrieval shall be provided in an enhanced auditory fashion, or support	Not Applicable	Audio related functionality does not apply to Essential Grid controls.

Criteria	Supporting Features	Remarks and explanations
for assistive hearing devices shall be provided.		
(e) At least one mode of operation and information retrieval that does not require user -speech shall be provided, or support for Assistive Technology used by people with disabilities shall be provided.	Fully Supported	Existing grid specific APIs can be used to retrieve enough information without user speech.
(f) At least one mode of operation and information retrieval that does not require fine motor control or simultaneous actions and that is operable with limited reach and strength shall be provided.	Fully Supported	Essential Grid controls provide functionality that conforms to these criteria.

For more information on the accessibility features of Syncfusion products, visit Syncfusion's accessibility web site at <http://www.syncfusion.com/accessibility>.

## 3.2 List of Controls

Essential Grid provides the following controls.

- **Grid Control**-This class is derived from GridControlBase. It is the primary class that encapsulates both the state persistence (including data persistence) and the rendering of the grid. Unless you are using one of the special grids (Grid Data Bound Grid, Grid Grouping control, or Grid List control), GridControl is the class you will use. This is a cell-oriented grid which will easily allow you to set both row and column properties, as well as set cell specific properties for the grid to hold the data. It can also be used in a virtual mode where the data is provided on demand or the Grid control can physically hold the data for you.
- **Grid Data Bound Grid**-This class is also derived from GridControlBase. This control is more column-oriented than the Grid control. It is primarily used as a grid bound to a data source that supports either IList or IListSource. Classes such as ArrayList, DataTable, and DataView are included in this collection of possible data sources. Grid Data Bound Grid has a collection property called GridBoundColumns, that maintains the column

properties which is similar to the System.Windows.Forms.DataGridViewColumnStyle class. It is this property that allows Grid Data Bound Grid to be described as a column-oriented grid.

- **Grid Grouping Control**-This grouping control which is derived from the control class implements several interfaces that add Grouping support to this class. If you need grouping support, multi-column sort support, or true nested-table hierarchical support in a grid, then this control is the one to be used. You can easily add expression columns, filter columns, and summary rows to this grid, as well as bind it to any IList data source. It can be fully designed by using Visual Studio.
- **Grid List Control**-This class is derived from System.Windows.Forms.ListControl, but can display multiple columns in its list. It has a GridControl object as a property of the class. This Grid member gives you grid-like access to a ListControl, and also provides both the data and formatting for the list control. This control is easy to use, provided your data source has the exact data you want displayed. But if you need to customize this control by hiding columns in your data source or changing column names, then generally using either the Grid Data Bound Grid or the Grid control in the list box mode is a simpler solution. This control exists so it can serve as the drop list object for the combo box that serves as a cell control.
- **Grid Record Navigation Control**-This class provides MS Access-like navigation support. It is normally used in conjunction with the Grid Data Bound Grid to display record numbers, and to allow record scrolling within the grid through a record information window which is displayed at the lower-left corner of the grid. You can also use it in conjunction with a Grid control.
- **Grid Aware Text Box**-This class is derived from System.Windows.Forms.TextBox. It allows you to bind it to the CurrentCell of either a Grid control or a Grid Data Bound Grid. The standard use for such a text box is to provide a special edit bar for editing grid cells or to serve as a formula bar in a formula grid.

## 4 Grid Controls

This section covers information on the following topics.

### 4.1 Grid Control

The Grid control is a general purpose **GridControl** class. You can use it by allowing it to hold its own data or virtually binding it to an external data source.

	A	Date	C	FormulaCell	Text	floats	
1	-952	16 Jun 1972	7	-945	KHVKEE	291.47	
2	-914	14 Apr 2005	4	-910	VMZHNKFHT	388.61	
3	-900	07 Oct 2006	18	-882	KVO	-470.37	
4	-941	23 Nov 1991	84	-857	VGF	-314.53	
5	-900	08 Oct 1978	48	-852	KC	261.47	
6	-874	04 Feb 1976	31	-843	RIOZBYFXZ	388.54	
7	-863	09 May 2005	29	-834	TRHU	95.92	
8	-836	21 May 1986	6	-830	YJJQ	-50.36	
9	-857	21 Jun 1983	32	-825	DCWH	18.36	
10	-919	27 Nov 1997	96	-823	CFIKHVV	356.8	
11	-862	23 Oct 2007	44	-818	LUIKIZW	365.49	
12	-841	04 May 1996	23	-818	TNSSDPMGPJL	-59.29	
13	-848	24 Apr 1984	64	-784	MG	56.54	
14	-830	04 May 1977	58	-772	NBUIW	-164.69	
15	-787	19 Jan 1966	47	-740	K	350.3	
16	-809	04 Mar 1976	80	-729	NARURC	449.14	
17	-740	10 May 1992	15	-725	VAAESCYQVU	392.4	
18	-751	07 Mar 2007	35	-716	RAEW	-454.18	
19	-738	03 Sep 1976	45	-693	GRZKC	351.84	
20	-764	07 Aug 1961	75	-689	GRPLKLPGNG	129.71	

Figure 69: Grid control showing Random Data for a Sorting Sample

## 4.1.1 Creating Grid Control

This section will provide the step-by-step procedure to create a Grid control through designer and through programmatical approach in a .NET application.

### 4.1.1.1 Through Designer

To make the task of designing the Grid control easier on a cell level, a new Designer Editor has been added. With the editor, the grid can be modified and saved (and loaded) to xml formatted files, or Soap formatted templates. There is also no longer a Toggle Interactive Mode design verb that was present in versions prior to 4.1.

#### To add a Grid Control to the Application

Following steps illustrate how to add a Grid control to your application.

1. Drag the **GridControl** component from the toolbox onto the form.

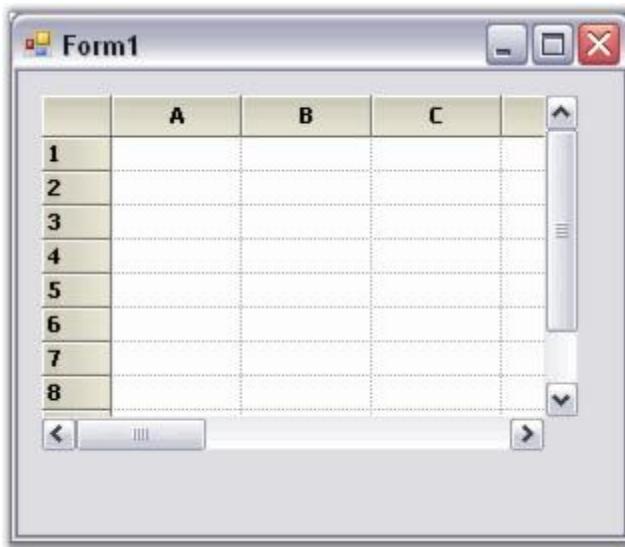


Figure 70: Grid Control

2. To edit the cell level properties of the grid (and also general Grid control properties), right-click anywhere in the Grid control and select **Edit**.

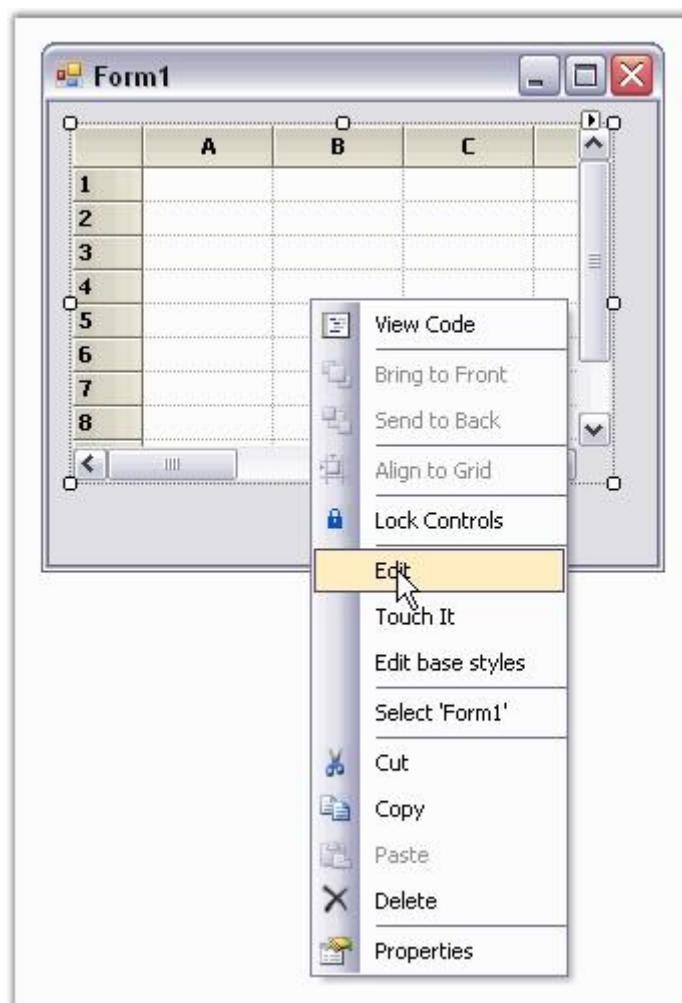
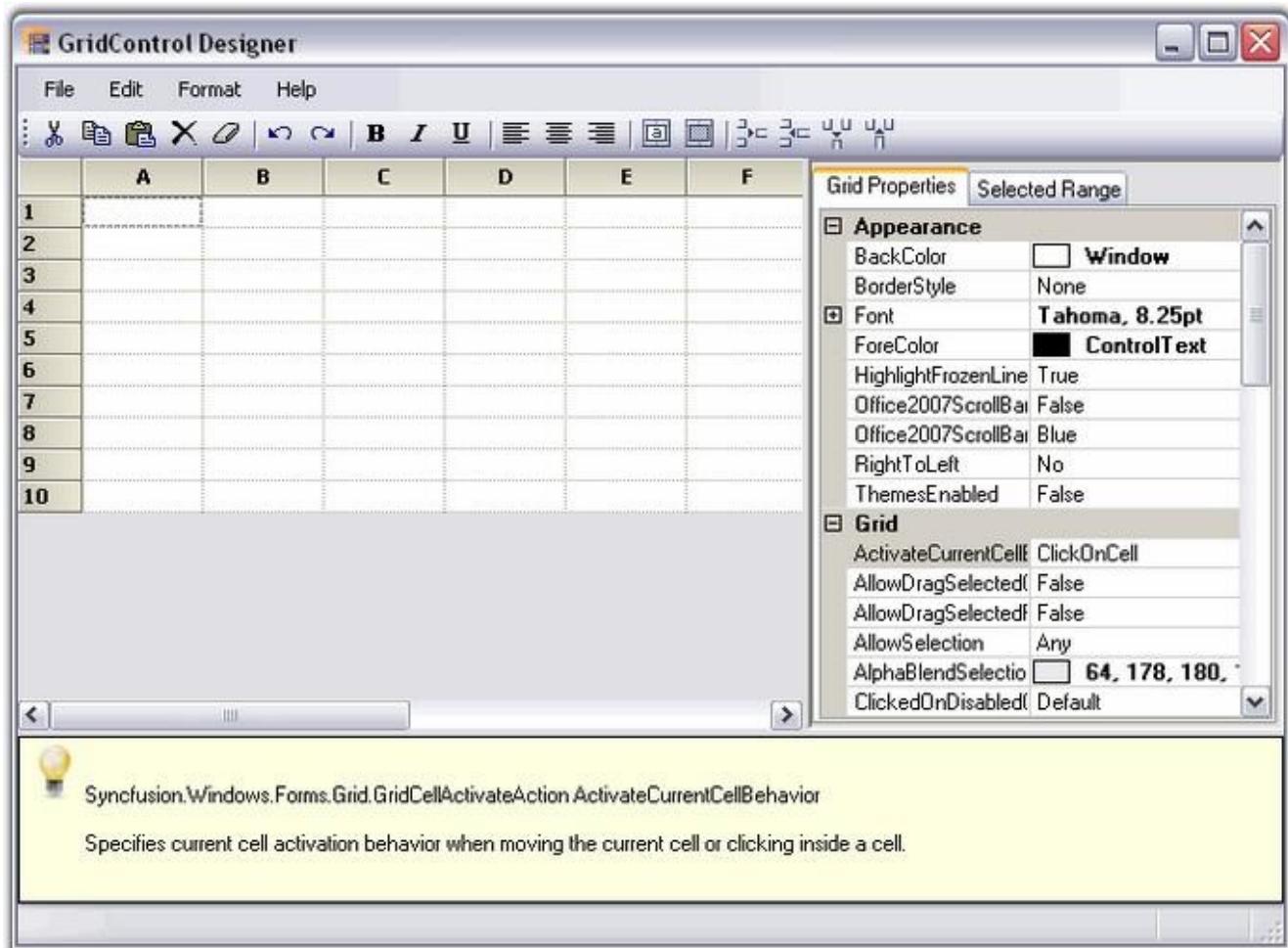


Figure 71: Select **Edit** option in the Grid Control Context Menu

3. This opens the **GridControl Designer** window. By using the GridControl Designer, the cell contents or styles, as well as general grid properties can be modified.



*Figure 72: Property Editor*

4. Single cells can be modified along with a selection of ranges. To do this, select a range of cells, and switch to the **Selected Range** tab to view the property grid for the selection.

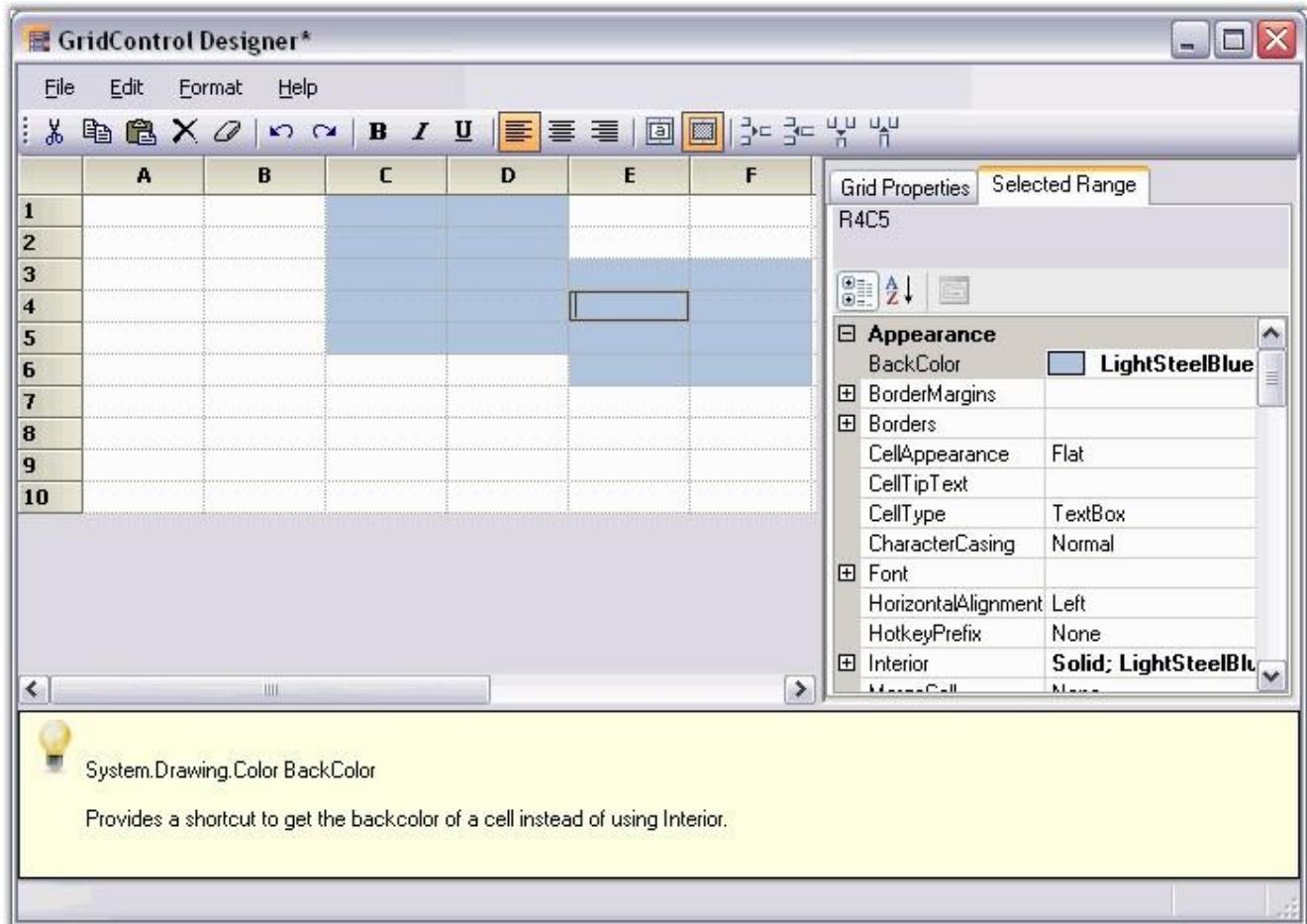


Figure 73: Property Window

GridControl Designer also lets you to save/load xml formatted files, and Soap templates.

5. When the changes are complete, simply exit out of the designer. If changes have been made, you will be prompted to save the changes to the Grid control in the designer.
6. Click **OK** to apply the settings to the grid control.

#### See Also

[Through Code](#)

#### 4.1.1.2 Through Code

The following code examples illustrate how to create a Grid control through code.

**[C#]**

```
// Create the Essential Grid.  
private Syncfusion.Windows.Forms.Grid.GridControl gridControl1;  
....  
this.gridControl1 = new Syncfusion.Windows.Forms.Grid.GridControl();  
  
// Set the number of rows and columns.  
this.gridControl1.ColCount = 10;  
this.gridControl1.RowCount = 100;  
  
// Position it on the form.  
this.gridControl1.Location = new System.Drawing.Point(20, 20);  
this.gridControl1.Size = new System.Drawing.Size(344, 200);  
  
// Add it to the form's controls.  
this.Controls.Add(this.gridControl1);
```

**[VB .NET]**

```
' Create the Essential Grid.  
Private WithEvents gridControl1 As GridControl  
....  
Me.gridControl1 = New Syncfusion.Windows.Forms.Grid.GridControl()  
  
' Set the number of rows and columns.  
Me.gridControl1.ColCount = 10  
Me.gridControl1.RowCount = 100  
  
' Position it on the form.  
Me.gridControl1.Location = New System.Drawing.Point(20, 15)  
Me.gridControl1.Size = New System.Drawing.Size(344, 150)  
  
' Add it to the form's controls.  
Me.Controls.Add(Me.gridControl1)
```

**See Also**

[Through Designer](#)

## 4.1.2 Elaborate Structure of the Control

This section lists the important elements which are used by the grouping grid to organize the data. These grid elements can have specific appearance and behavior. The following screen shot points out the various grid elements.

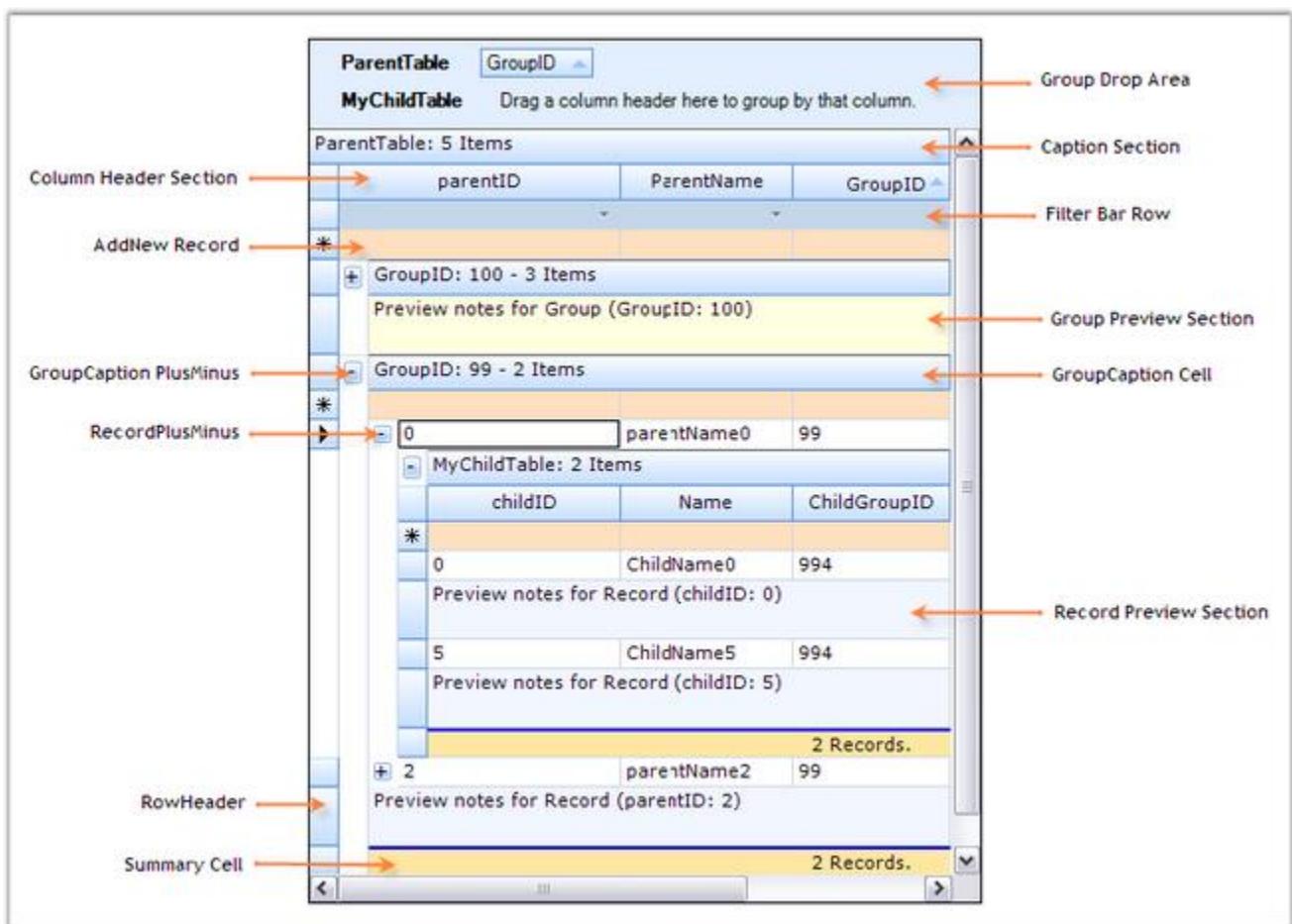


Figure 74: Structure of Grid Control

### Elements of Grid Control

- **Caption Section**-This is the first section within a group which provides the caption bar above the column headers. If a caption should be displayed at the beginning of each group, then Caption Sections are created when the grouping for a table is initialized.
- **Column Header Section**-This section will appear at the top of a table or a group, below the caption section. The Column Header Section is a place holder where the grouping grid displays the column headers.
- **Filter Bar Row**-It displays the filter bar for the table data. You can enable the record filters for specific columns.

- Add New Record Section**-This is the section within a group that is shown above the table records and/or below the records for each group, and implements logic to add new records.
- Preview Section**-A Preview Section can be added under each group and record. It is a suitable place to display custom data for a given record or group.
- Summary Section**-This section is a collection of rows used to display brief information about the groups or specific columns of the table.
- Group Drop Area**-It lets the user to group the grid data. The data can be grouped by simply dragging the column headers into the group drop area.

### 4.1.3 Feature Summary

Grid control is a powerful control that is implemented by using a unique Styles Architecture. This grid control which is similar to Microsoft Excel, allows detailed customization of the control down to the cell level. Multi-level Undo/Redo, shared scroll bar support, data/view separation, floating cells, more than 18 cell types and unmatched extensibility are unique features of the Grid control.

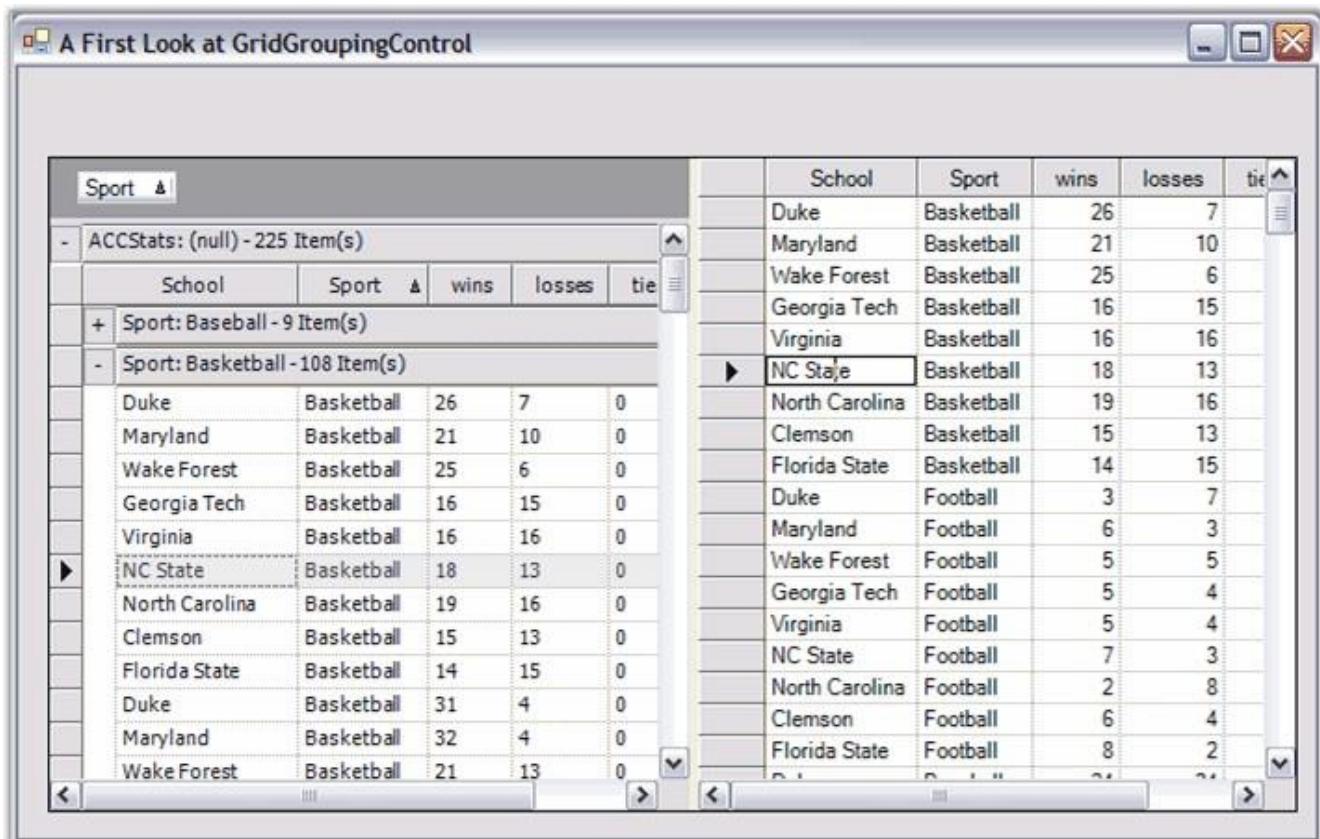


Figure 75: Grid Grouping control and Grid Data Bound Grid in One Form

This section discusses the following features.

#### 4.1.3.1 Cell Attributes

Cell attributes are properties that affect the appearance and behavior of the grid cells. They include features like the font used in the cell or the color of the cell. Following table lists the cell attributes.

Feature	Description
Cell TextAlign	Control horizontal and vertical alignment.
AutoSize	Automatically increase cell height.
PictureDisplay	Display picture within a cell.
CustomizeBorders	Change the appearance of grid borders.
Cell Appearances	Control cell edges (raised, flat, sunken).
Cell TipText	Add ToolTips to cells and scroll bars.
Cell AllowEnterMode	Allows enter mode during edit.
Cell TextColor	Change color of the text on a cell-by-cell basis.
Cell ImageList	Enable each cell to maintain a list of images.
Cell BaseStyle	Contains cells that depends on the same base style.
Cell Types	Specify different control types for each cell.
Cell InteriorColor	Specify a back color or gradient or pattern style.
Cell ChoiceList	Provides multiple choices as a single list.
CharacterCasing	Modify the case of characters as it is typed.
Control	Contains a custom control associated with a cell.

CultureInfo	Holds culture information rules.
CurrencyEdit	Holds currency text box properties for a cell.
DataSource	Assign data source on cell-by-cell basis.
DropDownStyle	Contains a drop-down list within a cell.
FloatCell	Control the text that floats into a neighboring cell.
FloodCell	Allows flooding from a previous cell.
Font	Set the font for drawing text.
Format	Control the format of the text within a cell.
FormulaTag	Associate a formula tag with a cell.
HotKey Prefix	Control the display of hot-key prefixes.
MaskedEdit	Control masked edit properties for a cell.
MergeCell	Merge cells with same data into single cell.
Cell TextOrientation	Control the angle at which the cell text is rendered.
Read-Only	Protect a cell from being edited.
Size	Control the size of the text within a cell.
ShowButtons	Control the display of the cell buttons.
StrictValueType	Control exception behavior parsing text.
Tag	Associate a custom tag with a cell.

The following path helps you in implementing the appearance of Grid Window:

**{Installed Drive}\Users\{User}\AppData\Local\Syncfusion\EssentialStudio\{Installed version}\Windows\Grid.Windows\Samples\2.0\Appearance**

The following path holds the Dashboard sample where Cell Style properties are implemented:

**{Installed Drive}\Users\{User}\AppData\Local\Syncfusion\EssentialStudio\{Installed version}\Windows\Grid.Windows\Samples\2.0\Appearance\Cell Style Demo**

### 4.1.3.2 Cell Types

Each cell may contain a specialized control such as a Text Box, Check Box or Combo Box, and this attribute of the cell is referred to as its Cell Type. Following table lists all the default cell types.

Feature	Description
Static Cell	Cannot be edited.
Custom Cell	Allows custom integration.
Formula Cell	Allows entries of algebraic formulas.
Currency Cell	Display currency type formats.
NumericUpDown	Allows to increase/decrease the numeric value.
ComboBox	Implements a standard combo box interface.
MaskedEdit	Allows users to input masks to control validation.
RichText Cell	Allows users to display and edit Rich Text.
WebBrowser Cell	Display a web browser.
ColorPicker	Allows the user to interactively select a color.
Grid Drop-Down	Allows drop-down grids in any mode.
HeaderText	Host a Header cell type.
CheckBox	Implements a check box.
PushButton	Implements a push button control.
MonthCalendar	Display a drop-down month calendar.
Password Cell	Accepts entries without displaying text.
ProgressBar Cell	Display Progress Bars.
Slider Cell	Display Slider Controls.
XHTML Cell	Display XHTML formatted text.

### 4.1.3.3 Data Binding

Essential Grid comes with a control to meet the user requirements such as a grid which is bound to an ADO.NET data source, or a grid that is virtually bound to the arbitrary collections, or a grid that maintains its own data storage. Following table lists the Data Binding features supported by the Grid control.

Feature	Description
ADO.NET	Supports a whole range of flexible data binding.
Virtual Mode	Loads data in a virtual manner.
Unbound Mode	Connects to an unbound data source.
RecordNavigationBar	Scrolls through the records in the table.
SplitViews	Splits the display of a data bound grid.
DraggingColumns	Rearranges the order of columns by using drag-and-drop operation.
CustomCollections	Binds to any collection that derives from the IList interface.
Multiple Rows per Record	Contains multiple rows per record.
Formatting Cells	Formats a cell depending on its value.
Images from Database	Display pictures inside cells.
DateTime/Boolean types	Chooses to have Boolean and DateTime.
Sorting	Sort the table alphabetically by values.
FilterBar	Filter the grid by items.
Drop-Down Lists	Display a foreign key table in a drop-down.

### 4.1.3.4 MS Office Simulation

Essential Grid supports MS Office features like dynamic splitters and undo/redo. Following table lists the available features.

Feature	Description
Accelerated Scrolling	Accelerate scrolling.
Shared Scrollbars	Share scroll bars with other windows.
Access Emulation	Scroll through records.
ScrollTips/Thumbtracking	Display scroll tips when the user performs the drag-and-drop operation.
Multilevel Undo/Redo	Allows multilevel undo/redo.
IntelliMouse Scrolling	Supports intellipoint mouse scrolling.
Excel-like Behavior	Has excel-like behavior with full features.
Workbook and Sheets	Has a tabbed workbook format.

#### 4.1.3.5 Functionalities

Essential Grid offers support for many functionalities such as OLE drag-and-drop, and resizing of rows and columns through property settings and special event handlers. Following table lists such features.

Feature	Description
PropertyGrid Integration	Set properties with the standard property grid.
Alpha-Blending	Blend a background image.
Supports XP Themes	Supports XP Themes.
Automatic Column Sizing	Auto resize column to hold content.
Reusable Grid Layouts	Save and load grid layouts.
Excel-like Splits	Split rows/columns similar to Excel.
Fixed Non-Scrolling Rows	Fix or freeze multiple rows.

Fixed Columns	Fix or freeze multiple columns.
Hierarchical Views	View data in a hierarchical way.
Worksheet Support	Has multiple worksheets.
Master/Detail	Show Master/Detail relationships.
MultiRow Per Record	Has one record in multiple rows.
HideRow/Column	Hide rows/columns.
Column Styles	Apply styles to entire columns.
Multi-line Column Header	Has multiline column header.
Select Multiple Blocks	Select multiple blocks of cells.
Selection Modes	Determines how users should select items.
Reusable Styles	Create shared styles.
Selection Styles	Customize the selection colors.
InsertRows/Columns	Insert rows and columns at run time.
DeleteRows/Columns	Delete rows and columns at run time.
MoveRows/Columns	Move rows and columns.
Clear Data	Clear data from the cells, rows, and columns.
Keyboard interface	Provides extensive support for proper keyboard handling.
Sorting	Sorts data displayed in the grid.
Filtering	Filter data based on a criteria.
Search	Find data in the grid.
Printing	Print the grid.
Print Preview	Preview the grid.
In-cell Formulas Support	Does advanced cell/ formula referencing.

Clipboard Support	Full clipboard support.
Resize Columns/Rows	Allows/prevents users from resizing columns/rows.
Drag-and-Drop Features	Drag cells, columns, or rows.
OLE Drag-and-Drop	Full OLE drag-and-drop support.
Mouse Operations	Change the appearance of the mouse pointer.

#### 4.1.3.6 General

Other features include Excel Export and versioned Serialization support. Following table lists the features.

Feature	Description
Document/View	Easily implement multiple views.
Dynamic Splitters	Use grid inside a dynamic splitter control.
Export to Excel BIFF 8	Export grid content to Excel with formatting.
100% Managed Code	Written in 100% managed code.
Tab Behavior	Control response selection of the tab.
Full Keyboard Support	Provides expected behavior for standard keys.
Serialization	Has versioned serialization of grid data.
GridAwareTextBox	Edit current cell outside the grid.

#### 4.1.4 Concepts and Features

This section provides you with detailed information on the features of Essential Grid. It gives you an overview of the major control classes.

You will also learn how to perform the following tasks:

- Add special controls to grid cells.
- Derive a cell control.
- Populate Essential Grid.
- Use the Grid control and GridStyleInfo classes.
- Use the Read-only attribute and Undo/Redo.
- Work with rows and columns.
- Create code for making the current row bold.
- Derive a GridPrintDocument.

Also, it discusses the following topics.

#### **4.1.4.1 Adding Special Controls to Grid Cells**

The **GridStyleInfo** property, **CellType**, lets you add special controls such as a check box or a combo box to a grid cell. Since the **CellType** is a member of **GridStyleInfo**, you can use it on a cell basis, row basis, column basis, or on a table basis, simply by setting this **CellType** property on the appropriate style. If you want the entire grid to be a combo box, then you simply have to set the grid's **CellValue** property to use combo boxes. You can derive your own controls to implement additional cell types. For more details, see [deriving a cell control](#).

Following table lists the cell types that are supported in Essential Grid.

<b>Grid Cell Control</b>	<b>Description</b>
Check Box	Displays a check box in the cell.
Color Edit	Displays a color selection and allows editing color choices.
Combo Box	Displays a combo box in the cell.
Control	Displays a System.Windows.Forms.Control in a cell.
Currency Edit	Displays a currency value and allows editing of it.
Formula Cell	Displays calculation from a formula entered in the cell.
Grid List Control	Displays a multicolumn list control as a drop

	down.
Header	Displays cells as grid header cells with static text.
Masked Edit	Uses a mask to control values entered into the cell.
Month Calendar	Displays a DateTime value and allows editing of it.
Numeric Up Down	Displays numeric text that can be edited or modified with spinner buttons.
Progress Bar	Displays a ProgressBar control in a cell.
Push Button	Displays a button in the cell that the user can click.
Rich Text	Displays rich text in the cell and allows editing while in a drop down.
Slider	Displays a slider control in a cell.
Static	Displays text in the cell that cannot be edited.
Text Box	Displays text in the cell that can be edited.

#### 4.1.4.1.1 Check Box

The **Check Box** cell type displays a check box in a grid cell. The check box has three states: **Checked**, **Unchecked** and **Indeterminate**. You can decide whether the check box should behave as a two-state check box or a three-state check box.

The following **GridStyleInfo** properties can be used to control the functioning of the check box.

GridStyleInfo Property	Description
CellType	Set to "check box" for a check box control.
CheckBoxOptions	Defines the display value of True, False, or indeterminate (i.e., the value returned by the GridStyleInfo.Text property).

Description	Text that appears next to the check box.
TriState	Whether or not indeterminate value is supported.
CellValue	Boolean true or false values, or empty (null or nothing).

The following code example illustrates how to set the cell type to CheckBox.

#### [C#]

```
// Specify display values for True/False/Indeterminate.
gridControl1.TableStyle.CheckBoxOptions = new
GridCheckBoxCellInfo("True", "False", "", false);

// Set up a check box with no tristate.
gridControl1[rowIndex,colIndex].CellValue = false;
gridControl1[rowIndex,colIndex].Description = "Click Me";
gridControl1[rowIndex,colIndex].CellType = "CheckBox";
gridControl1[rowIndex,colIndex].TriState = false;

// Set up a check box with tristate.
gridControl1[rowIndex,colIndex + 1].CellValue = true;
gridControl1[rowIndex,colIndex + 1].CellType = "CheckBox";
gridControl1[rowIndex,colIndex + 1].TriState = true;
gridControl1[rowIndex,colIndex + 1].Description = "TriState";
```

#### [VB.NET]

```
' Specify display values for True/False/Indeterminate.
gridControl1.TableStyle.CheckBoxOptions = New
GridCheckBoxCellInfo("True", "False", "", False)

' Set up a check box with no tristate.
gridControl1(rowIndex, colIndex).CellValue = False
gridControl1(rowIndex, colIndex).Description = "Click Me"
gridControl1(rowIndex, colIndex).CellType = "CheckBox"
gridControl1(rowIndex, colIndex).TriState = False

' Set up a check box with tristate.
gridControl1(rowIndex, colIndex + 1).CellValue = True
gridControl1(rowIndex, colIndex + 1).CellType = "CheckBox"
gridControl1(rowIndex, colIndex + 1).TriState = True
gridControl1(rowIndex, colIndex + 1).Description = "TriState"
```



Figure 76: Check Box Cells

#### 4.1.4.1.2 Color Edit

The **Color Edit** cell type allows you to pick colors and set a color object as the **CellValue**. To do this, you have to set the **CellType** property to *ColorEdit*.

The following code example illustrates how to set the cell type to *ColorEdit*.

[C#]

```
// Set up a Color Edit control.  
gridControl1[rowIndex, colIndex].CellType = "ColorEdit";  
gridControl1[rowIndex, colIndex].CellValue = Color.Aqua;
```

[VB.NET]

```
' Set up a Color Edit control.  
gridControl1(rowIndex, colIndex).CellType = "ColorEdit"  
gridControl1(rowIndex, colIndex).CellValue = Color.Aqua
```

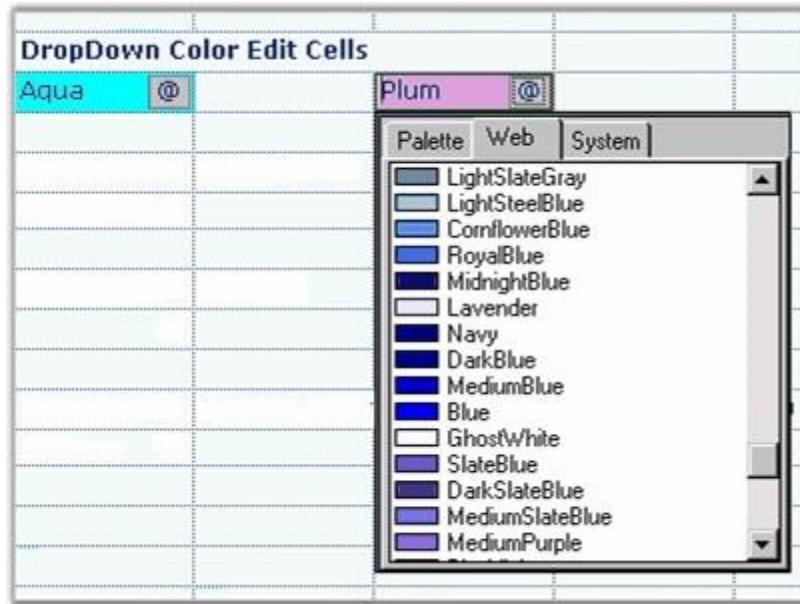


Figure 77: Color Edit Cells

#### 4.1.4.1.3 Combo Box

When you add a combo box to a grid cell, it will enable you to choose from a drop-down list of choices. You can populate this list in several ways by setting the appropriate **GridStyleInfo** properties. Other properties restrict the choices to those items listed in the drop down, and enable auto completion of possible matches as the user types new items.

GridStyleInfo Property	Description
CellType	Set to "combo box" for a combo box control.
ChoiceList	StringCollection holding the strings for the drop down.
ExclusiveChoiceList	<i>True</i> if you want to list the items in the drop-down, <i>false</i> otherwise.
DataSource	This property lets you to populate the drop-down list from by using an object that implements IListSource or IList. Examples include DataTable, DataSet, DataView and ArrayList.
DisplayMember	String that names the public property from the

	data source object to be displayed in the cell.
ValueMember	String that names the public property from the data source object to be used as the value for this cell.

The following code example illustrates how to set the cell type to ComboBox.

#### [C#]

```
// Generate the choices.
StringCollection items = new StringCollection();
items.Add("One");
items.Add("Two");
items.Add("Three");
items.Add("Four");
items.Add("Five");

// Set up the control.
gridControl1[rowIndex, colIndex].CellType = "ComboBox";
gridControl1[rowIndex, colIndex].ChoiceList = items;
gridControl1[rowIndex, colIndex].Text = "Five";
gridControl1[rowIndex, colIndex].CellType = "ComboBox";
gridControl1[rowIndex, colIndex].ExclusiveChoiceList = true;

// Or use a data source such as a table in a data set.
gridControl1[2, 2].CellType = "ComboBox";
gridControl1[2, 2].DataSource = this.dataSet11.Tables["Customers"];
gridControl1[2, 2].DisplayMember = "CustomerID";
gridControl1[2, 2].ValueMember = "CustomerID";
```

#### [VB .NET]

```
' Generate the choices.
Dim items As StringCollection = New StringCollection()
items.Add("One")
items.Add("Two")
items.Add("Three")
items.Add("Four")
items.Add("Five")

' Set up the control.
gridControl1(rowIndex, colIndex).CellType = "ComboBox"
gridControl1(rowIndex, colIndex).ChoiceList = items
```

```
gridControl1(rowIndex, colIndex).Text = "Five"
gridControl1(rowIndex, colIndex).CellType = "ComboBox"
gridControl1(rowIndex, colIndex).ExclusiveChoiceList = True

' Or use a data source such as a table in a dataset.
gridControl1(2, 2).CellType = "ComboBox"
gridControl1(2, 2).DataSource = Me.dataSet11.Tables("Customers")
gridControl1(2, 2).DisplayMember = "CustomerID"
gridControl1(2, 2).ValueMember = "CustomerID"
```

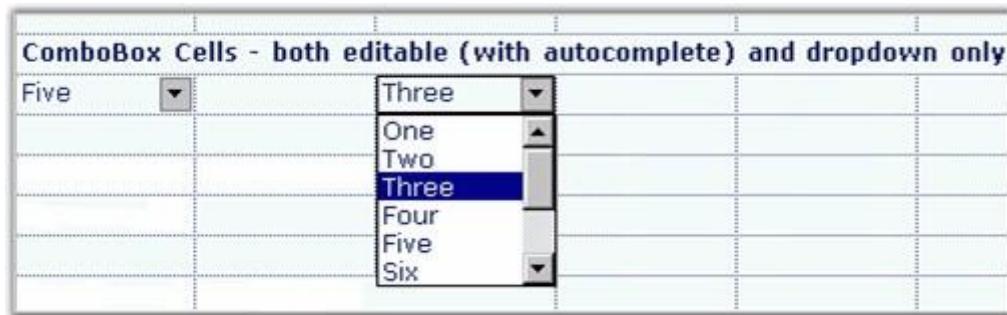


Figure 78: Combo Box Cells

#### 4.1.4.1.3.1 AutoComplete Support for Combo Box in Edit Mode

Essential Grid provides AutoComplete support for combo box cells. The AutoComplete feature is a filtered suggestion list presented in a drop-down that is pulled from a mapped data source as the user enters text into a text box. AutoComplete for combo box cells provides the following properties:

- AutoComplete—Displays suggestion in the text box. The content other than what you have typed will be highlighted.
- AutoSuggest—Dynamically populates a list based on the entered text.
- Both—Enables normal editable behavior.
- None—No operations will be performed in the text box and list box areas.

#### Use Case Scenarios

You can choose the suggestion instead of typing the entire content.

#### Properties

Table 1: Properties Table

Property	Description	Type	Data Type	Reference links
AutoComplete	Gets the a suggestion from the list based on the entered text. The suggestion will be highlighted.	Enumerator	N/A	N/A
AutoSuggest	Dynamically populate a list based on the entered text.	Enumerator	N/A	N/A
Both	Enables normal editable behavior.	Enumerator	N/A	N/A
None	No operations will be performed in the text box and list box areas.	Enumerator	N/A	N/A

**Enabling AutoComplete inEditMode for a Combo Box Celltype**

The following steps illustrates enabling AutoComplete inEditMode for a combo Box celltype:

1. Declare the Celltype as Combo Box as given in the following code:

[C#]

```
this.gridControl1[RowIndex, ColIndex].CellType = GridCellTypeName.ComboBox;
```

[VB]

```
Me.gridControl1(RowIndex, ColIndex).CellType = GridCellTypeName.ComboBox
```

2. Set the Dropdown style as Editable as given in the following code:

[C#]

```
this.gridControl1[RowIndex, ColIndex].DropDownStyle = GridDropDownStyle.Editable;
```

[VB]

```
Me.gridControl1(RowIndex, ColIndex).DropDownStyle = GridDropDownStyle.Editable
```

3. Set the *GridComboSelectionOptions* using the *AutoCompleteInEditMode* property as given in the following code:

[VB]

```
Me.gridControl1(RowIndex, ColIndex).AutoCompleteInEditMode =  
GridComboSelectionOptions.AutoSuggest
```

[VB]

```
Me.gridControl1(RowIndex, ColIndex).AutoCompleteInEditMode =  
GridComboSelectionOptions.AutoSuggest
```

#### 4.1.4.1.4 Control

You can place an arbitrary control in a grid cell through the **Control** cell type. This cell type differs from most other cell types shipped with Essential Grid in which it cannot be shared among several cells. The Control cell type requires you to instantiate a control object for each cell that uses this cell type, and set that object to *style.Control*. A different control object is required for every cell that makes use of the Control cell type.

The following code example illustrates how to set the cell type to Control.

[C#]

```
// Set up a Control Cell.  
this.radioButton1.Checked = true;  
this.gridControl1.CoveredRanges.Add(GridRangeInfo.Cells(2,2,8,2));  
this.gridControl1.ColWidths[2] = 200;  
this.gridControl1[2,2].CellType = "Control";  
  
// Set the control object.  
this.gridControl1[2,2].Control = this.dataPanel;
```

[VB.NET]

```
' Set up a Control Cell.  
Me.radioButton1.Checked = True  
Me.gridControl1.CoveredRanges.Add(GridRangeInfo.Cells(2, 2, 8, 2))  
Me.gridControl1.ColWidths(2) = 200  
Me.gridControl1(2, 2).CellType = "Control"  
  
' Set the control object.  
Me.gridControl1(2, 2).Control = Me.dataPanel
```

The following screen shot shows a panel holding two radio buttons and a push button in the cell.



Figure 79: Control Cells

#### 4.1.4.1.5 Currency Edit

The **Currency Edit** cell type lets you edit monetary values and display them by using different currency type formats. To achieve this, you must set the **CellType** property to *Currency*. You can set additional properties such as the decimal and group separator for the cell value.

The following code example illustrates how to set the cell type to CurrencyEdit.

**[C#]**

```
GridStyleInfo style = gridControl1[row, 2];
style.CellType = "Currency";
style.Text = "$1.00";

// Set the clip mode.
style.CurrencyEdit.ClipMode = CurrencyClipModes.IncludeFormatting;

// Set formatting properties.
style.CurrencyEdit.CurrencyDecimalDigits = 2;
style.CurrencyEdit.CurrencyDecimalSeparator = ".";
style.CurrencyEdit.CurrencyGroupSeparator = ",";
style.CurrencyEdit.CurrencyGroupSizes = new int[] {3};
style.CurrencyEdit.CurrencyNegativePattern = 1;
style.CurrencyEdit.CurrencyNumberDigits = 27;
style.CurrencyEdit.CurrencyPositivePattern = 0;
style.CurrencyEdit.CurrencySymbol = "$";
style.CurrencyEdit.NegativeColor = System.Drawing.Color.Red;
style.CurrencyEdit.NegativeSign = "-";
style.CurrencyEdit.PositiveColor = System.Drawing.Color.Black;
style.FloatCell = true;
```

**[VB .NET]**

```
Dim style As GridStyleInfo = gridControl1(row, 2)
style.CellType = "Currency"
style.Text = "$1.00

' Set the clip mode.
style.CurrencyEdit.ClipMode = CurrencyClipModes.IncludeFormatting

' Set formatting properties.
style.CurrencyEdit.CurrencyDecimalDigits = 2
style.CurrencyEdit.CurrencyDecimalSeparator = "."
style.CurrencyEdit.CurrencyGroupSeparator = ","
style.CurrencyEdit.CurrencyGroupSizes = New Integer() {3}
style.CurrencyEdit.CurrencyNegativePattern = 1
style.CurrencyEdit.CurrencyNumberDigits = 27
style.CurrencyEdit.CurrencyPositivePattern = 0
style.CurrencyEdit.CurrencySymbol = "$"
style.CurrencyEdit.NegativeColor = System.Drawing.Color.Red
style.CurrencyEdit.NegativeSign = "-"
style.CurrencyEdit.PositiveColor = System.Drawing.Color.Black
style.FloatCell = True
```

	A	B	C
1			
2		\$22.00	
3		\$1.03	
4		\$1.00	
5		-\$1,123.00	
6		\$1,333.00	
7		\$1,333.00	
8		\$1,122.21	
9		\$112.00	
10		-\$1,213.00	
11			

Figure 80: Currency Cells

#### 4.1.4.1.6 Formula Cell

The **FormulaCell** cell type allows you to add algebraic formulas to a cell that depends on other cells. The cell value should be a well-formed formula starting with an '=' and the **CellType** property set to *FormulaCell*. If a Formula Cell does not begin with an '=', the cell is treated as a text box cell. For details, see [Formula Support](#).

The following code example illustrates how to set the cell type to *FormulaCell*.

##### [C#]

```
// Set Cell Type as Formula Cell.
gridControl1[rowIndex, colIndex].CellType = "FormulaCell";

// Assign a Formula.
gridControl1[rowIndex, colIndex].CellValue = "= (A1+A2) / 2";
```

##### [VB .NET]

```
' Set Cell Type as Formula Cell.
gridControl1(rowIndex, colIndex).CellType = "FormulaCell"

' Assign a Formula.
gridControl1(rowIndex, colIndex).CellValue = "= (A1+A2) / 2"
```

	A	B	C	D
1	11	6	=a1+b1	
2	33	15	=a2+b2	
3	-2	18	=a3+b3	
4	=sum(a1:a3)	=avg(b1:b3)	=a4+b4	
5				

Figure 81: Formula Cells shown as Text Boxes

	A	B	C	D
1	11	6	17	
2	33	15	48	
3	-2	18	16	
4	42	13	=a4+b4	
5				

Figure 82: Same Cells shown as Formula Cells

#### 4.1.4.1.7 Grid List Control

The **GridListControl** cell type allows you to display a drop-down list that can contain multiple columns as an image. It uses **DataSource**, **DisplayMember** and **ValueMember** properties to control what is shown in the multiple columns. The **DataSource** member is generally stored in a parent style, and this member is then shared among grid cells which might use **DisplayMember** and **ValueMember** properties to customize their look if needed.

GridStyleInfo Property	Description
DisplayMember	Any object that implements either <b>IList</b> or <b>IListSource</b> . These include <b>DataTable</b> , <b>DataView</b> , or <b>ArrayList</b> objects.
ValueMember	Indicates the column from the data source that is to be used for the value of the cell.
ExclusiveChoiceList	Determines whether the user is required to select an item in the drop-down list.
MultiColumn	Determines whether all the columns in the data source are displayed or if the single <b>DisplayMember</b> column is displayed.

Let us assume you have an ArrayList of US State objects. When you set the cell type to GridListControl, you will get the output as displayed in the screen shot.

**[C#]**

```
// Set up the data source.  
// Here "USStates" is an arraylist of state objects, each of which have  
// the properties LongName and ShortName.  
gridControl1.TableStyle.DataSource = USStates;  
gridControl1.TableStyle.DisplayMember = "LongName";  
gridControl1.TableStyle.ValueMember = "ShortName";  
  
gridControl1[rowIndex, colIndex + 2].CellType = "GridListControl";  
gridControl1[rowIndex, colIndex + 2].Text = "Wisconsin";  
gridControl1[rowIndex, colIndex + 2].ExclusiveChoiceList = true;
```

**[VB .NET]**

```
' Set up the data source.  
' Here "USStates" is an arraylist of state objects each of which, have  
// the properties LongName and ShortName.  
gridControl1.TableStyle.DataSource = USStates  
gridControl1.TableStyle.DisplayMember = "LongName"  
gridControl1.TableStyle.ValueMember = "ShortName"  
  
gridControl1(rowIndex, colIndex + 2).CellType = "GridListControl"  
gridControl1(rowIndex, colIndex + 2).Text = "Wisconsin"  
gridControl1(rowIndex, colIndex + 2).ExclusiveChoiceList = True
```



Figure 83: Grid List Control Cells

A sample which demonstrates Grid List Control cell type is available in the following sample installation path.

**C:\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Windows\Samples\2.0\Grid List Control\Grid List Control Demo**

#### 4.1.4.1.8 Header

The **Header** cell type displays static text similar to the static **CellType**. But the Header cell type, in addition, has a button-like border that can have a depressed state.

The following code example illustrates how to set the cell type to Header.

**[C#]**

```
// Set Cell Type as "Header".
gridControl1[rowIndex,colIndex].Text = "HeaderText";

// Set Formatting properties.
gridControl1[rowIndex,colIndex].CellType = "Header";
gridControl1[rowIndex,colIndex].BackColor = Color.FromArgb(208, 208,
208);
```

**[VB .NET]**

```
' Set Cell Type as "Header".
gridControl1(rowIndex, colIndex).Text = "HeaderText"

' Set Formatting properties.
gridControl1(rowIndex, colIndex).CellType = "Header"
gridControl1(rowIndex, colIndex).BackColor = Color.FromArgb(208, 208,
208)
```

Header Cells - used as row and column headers		
HeaderText		HeaderText

Figure 84: Header Cells

#### 4.1.4.1.9 Masked Edit

The **MaskedEdit** cell type lets you to edit and display specially formatted text cells that conform to an edit mask that you specify. To make use of this cell type, set the **CellType** property to **MaskedEdit**. You can set additional properties like **Mask**, **ClipMode**, and so on, through the cell style's **GridMaskEditInfo** object. The various options will allow you to input masks to control the type of input that is valid within a cell. For example, you can use a **MaskedEdit** cell to facilitate the entry of a formatted Social Security number, a phone number, or a 3 character alpha-code.

The following code example illustrates how to set the cell type to **MaskedEdit**.

[C#]

```
gridControl1[2, 3].Text = "First Name";
GridStyleInfo style1 = gridControl1[2, 4];
GridMaskEditInfo maskedEditStyle1 = style1.MaskEdit;
gridControl1[4, 3].Text = "Last Name";
gridControl1[8, 3].Text = "Social Security";
GridStyleInfo style4 = gridControl1[8, 4];
GridMaskEditInfo maskedEditStyle4 = style4.MaskEdit;

// Masked Edit Box 1
style1.CellType = "MaskEdit";
maskedEditStyle1.AllowPrompt = false;
maskedEditStyle1.ClipMode =
Syncfusion.Windows.Forms.Tools.ClipModes.ExcludeLiterals;
style1.CultureInfo = new System.Globalization.CultureInfo("en-US");
maskedEditStyle1.DateSeparator = '-';
maskedEditStyle1.Mask = ">C<CCCCCCCCCC";
style1.MaxLength = 13;
style1.AutoSize = true;
maskedEditStyle1.SpecialCultureValue =
Syncfusion.Windows.Forms.Tools.SpecialCultureValues.None;
maskedEditStyle1.UseLocaleDefault = false;
maskedEditStyle1.UseUserOverride = true;

// Masked Edit Box 4
style4.CellType = "MaskEdit";
maskedEditStyle4.AllowPrompt = false;
maskedEditStyle4.ClipMode =
Syncfusion.Windows.Forms.Tools.ClipModes.IncludeLiterals;
style4.CultureInfo = new System.Globalization.CultureInfo("en-US");
maskedEditStyle4.DateSeparator = '-';
maskedEditStyle4.Mask = "999-99-9999";
style4.MaxLength = 11;
maskedEditStyle4.SpecialCultureValue =
Syncfusion.Windows.Forms.Tools.SpecialCultureValues.None;
```

```
style4.Text = "___-__-___";
maskedEditStyle4.UseLocaleDefault = false;
maskedEditStyle4.UseUserOverride = true;
```

**[VB.NET]**

```
gridControl1(2, 3).Text = "First Name"
Dim style1 As GridStyleInfo = gridControl1(2, 4)
Dim maskedEditStyle1 As GridMaskEditInfo = style1.MaskEdit
gridControl1(4, 3).Text = "Last Name"
gridControl1(8, 3).Text = "Social Security"
Dim style4 As GridStyleInfo = gridControl1(8, 4)
Dim maskedEditStyle4 As GridMaskEditInfo = style4.MaskEdit

' Masked Edit Box 1
style1.CellType = "MaskEdit"
maskedEditStyle1.AllowPrompt = False
maskedEditStyle1.ClipMode =
Syncfusion.Windows.Forms.Tools.ClipModes.ExcludeLiterals
style1.CultureInfo = New System.Globalization.CultureInfo("en-US")
maskedEditStyle1.DateSeparator = "-c"
maskedEditStyle1.Mask = ">C<CCCCCCCCCC"
style1.MaxLength = 13
style1.AutoSize = True
maskedEditStyle1.SpecialCultureValue =
Syncfusion.Windows.Forms.Tools.SpecialCultureValues.None
maskedEditStyle1.UseLocaleDefault = False
maskedEditStyle1.UseUserOverride = True

' Masked Edit Box 4
style4.CellType = "MaskEdit"
maskedEditStyle4.AllowPrompt = False
maskedEditStyle4.ClipMode =
Syncfusion.Windows.Forms.Tools.ClipModes.IncludeLiterals
style4.CultureInfo = New System.Globalization.CultureInfo("en-US")
maskedEditStyle4.DateSeparator = "-c"
maskedEditStyle4.Mask = "999-99-9999"
style4.MaxLength = 11
maskedEditStyle4.SpecialCultureValue =
Syncfusion.Windows.Forms.Tools.SpecialCultureValues.None
style4.Text = "___-__-___"
maskedEditStyle4.UseLocaleDefault = False
maskedEditStyle4.UseUserOverride = True
```

<b>First Name</b>	Sue
<b>Last Name</b>	gaskins
<b>Telephone</b>	(999) 555 - 1212
<b>Social Security</b>	222-33-4444

*Figure 85: Masked Edit Cells*

#### 4.1.4.1.10 Month Calendar

The **MonthCalendar** cell type lets you pick dates. To make use of this cell type in grid, set the **CellType** property to *MonthCalendar* and **CellValue** property to *DateTime* object.

The following code example illustrates how to set the cell type to MonthCalendar.

[C#]

```
// Set Cell Type.  
gridControl1[rowIndex, colIndex].CellType = "MonthCalendar";  
  
// Assign initial value.  
gridControl1[rowIndex, colIndex].CellValue = DateTime.Now;
```

[VB .NET]

```
' Set Cell Type.  
gridControl1(rowIndex, colIndex).CellType = "MonthCalendar"  
  
' Assign initial value.  
gridControl1(rowIndex, colIndex).CellValue = DateTime.Now
```

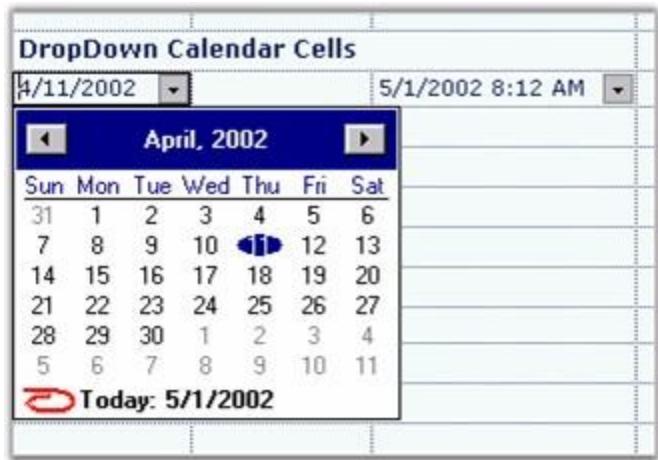


Figure 86: Month Calendar Cells

#### 4.1.4.1.11 Numeric Up Down

A **NumericUpDown CellType** lets you input numeric data either by editing the displayed text or by using spinner buttons to increase or decrease the displayed value. As your value hits a limit, you can either have it stick at that limit or wrap to the opposite limiting value. To hold information such as the upper and lower limits, Essential Grid uses a **GridNumericUpDownCellInfo** object whose constructor accepts the parameters used in the control. This is illustrated in the following code.

[C#]

```
// Set up a NumericUpDown Control and set up upper and lower limits.
public GridNumericUpDownCellInfo(int min, int max, int start, int step,
bool wrap)
```

[VB .NET]

```
' Set up a NumericUpDown Control and set up upper and lower limits.
Public Sub New(min As Integer, max As Integer, start As Integer, step1
As Integer, wrap As Boolean)
```



Figure 87: Numeric Up Down Cells

#### 4.1.4.1.12 Progress Bar

There are several formatting options that can be applied to an ProgressBar cell type embedded into the grid control. The following code example illustrates this.

[c#]

```
// Set up a Progress Bar Control.
GridStyleInfo style3 = gridControl1[12, 2];
GridProgressBarInfo progressBarEx3 = style3.ProgressBar;
style3.CellType = "ProgressBar";
style3.Themed = false;

// Apply Styles.
progressBarEx3.BackGradientEndColor = System.Drawing.Color.RosyBrown;
progressBarEx3.BackGradientStartColor = System.Drawing.Color.DarkRed;
progressBarEx3.BackgroundStyle =
Syncfusion.Windows.Forms.Tools.ProgressBarBackgroundStyles.VerticalGradient;
progressBarEx3.BackMultipleColors = new System.Drawing.Color[0];
progressBarEx3.BackSegments = false;
progressBarEx3.BackTubeEndColor = System.Drawing.SystemColors.Control;
progressBarEx3.BackTubeStartColor = System.Drawing.Color.LightGray;
progressBarEx3.FontColor = System.Drawing.Color.Lime;
progressBarEx3.ForegroundImage = null;
progressBarEx3.GradientEndColor = System.Drawing.Color.Lime;
progressBarEx3.GradientStartColor = System.Drawing.Color.Red;
progressBarEx3.MultipleColors = new System.Drawing.Color[]
{
    System.Drawing.SystemColors.ControlDarkDark,
    System.Drawing.SystemColors.ControlLight,
    System.Drawing.SystemColors.ControlDark,
    System.Drawing.SystemColors.Control
};
progressBarEx3.ProgressStyle =
Syncfusion.Windows.Forms.Tools.ProgressBarStyles.Tube;
progressBarEx3.SegmentWidth = 12;
progressBarEx3.TextVisible = false;
progressBarEx3.TubeEndColor = System.Drawing.Color.Black;
progressBarEx3.TubeStartColor = System.Drawing.Color.Red;
progressBarEx3.ProgressValue = 75;
```

**[VB.NET]**

```
' Set up a Progress Bar Control.  
Dim style3 As GridStyleInfo = gridControl1(GridStyleInfoType.ProgressBar, 2)  
Dim progressBarEx3 As GridProgressBarInfo = style3.ProgressBar  
style3.CellType = "ProgressBar"  
style3.Themed = False  
  
' Apply Styles.  
progressBarEx3.BackGradientEndColor = System.Drawing.Color.RosyBrown  
progressBarEx3.BackGradientStartColor = System.Drawing.Color.DarkRed  
progressBarEx3.BackgroundStyle = Syncfusion.Windows.Forms.Tools.ProgressBarBackgroundStyles.VerticalGradient  
progressBarEx3.BackMultipleColors = New System.Drawing.Color(0) {}  
progressBarEx3.BackSegments = False  
progressBarEx3.BackTubeEndColor = System.Drawing.SystemColors.Control  
progressBarEx3.BackTubeStartColor = System.Drawing.Color.LightGray  
progressBarEx3.FontColor = System.Drawing.Color.Lime  
progressBarEx3.ForegroundImage = Nothing  
progressBarEx3.GradientEndColor = System.Drawing.Color.Lime  
progressBarEx3.GradientStartColor = System.Drawing.Color.Red  
progressBarEx3.MultipleColors = New System.Drawing.Color() {  
    System.Drawing.SystemColors.ControlDarkDark,  
    System.Drawing.SystemColors.ControlLight, System.Drawing.SystemColors.ControlDark, System.Drawing.SystemColors.Control}  
progressBarEx3.ProgressStyle = Syncfusion.Windows.Forms.Tools.ProgressBarStyles.Tube  
progressBarEx3.SegmentWidth = 12  
progressBarEx3.TextVisible = False  
progressBarEx3.TubeEndColor = System.Drawing.Color.Black  
progressBarEx3.TubeStartColor = System.Drawing.Color.Red  
progressBarEx3.ProgressValue = 75
```

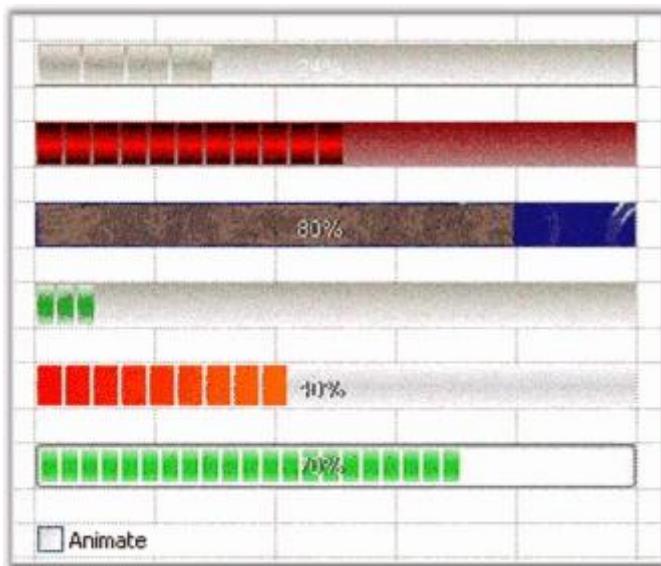


Figure 88: Push Button Cells

For other code samples, refer to the sample in the following location:

**C:\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Windows\Samples\2.0\Cell Types\Progress Bar Cell Demo**

#### 4.1.4.1.13 Push Button

To display a Push Button in a grid cell, use the **PushButton** cell type. To catch and handle a user click on a button, and you can add a **GridControl.CellButtonClicked** event handler. The event arguments passed into your handler will include the row and column of the click. The **GridStyleInfo** properties that control the behavior of a Push Button cell are listed in the following table.

Property	Description
CellAppearance	Specifies whether the button is raised, sunken or flat.
CellType	Set to "PushButton" for a push button control.

The following code example illustrates how to set the cell type to PushButton.

[C#]

```
gridControl1[rowIndex,colIndex].Description = "PushButton1";
gridControl1[rowIndex,colIndex].CellType = "PushButton";
gridControl1[rowIndex,colIndex ].CellAppearance =
GridCellAppearance.Raised;

// To catch a click, hook up a CellButtonClicked handler.
gridControl1.CellButtonClicked += new
GridCellButtonClickedEventHandler(gridControl1_CellButtonClicked);

// Add a handler.
private void gridControl1_CellButtonClicked(object sender,
GridCellButtonClickedEventArgs e)
{
    MessageBox.Show("You clicked row" + e.RowIndex.ToString() + "col"
+ e.ColumnIndex.ToString());
}
```

**[VB.NET]**

```
gridControl1(rowIndex, colIndex).Description = "PushButton1"
gridControl1(rowIndex, colIndex).CellType = "PushButton"
gridControl1(rowIndex, colIndex).CellAppearance =
GridCellAppearance.Raised

' To catch a click, hook up a CellButtonClicked handler.
AddHandler gridControl1.CellButtonClicked, AddressOf
gridControl1_CellButtonClicked

' Add a handler.
Private Sub gridControl1_CellButtonClicked(ByVal sender As Object,
ByVal e As GridCellButtonClickedEventArgs)
    MessageBox.Show("You clicked row " + e.RowIndex.ToString() + " col
" + e.ColumnIndex.ToString())
End Sub
```



Figure 89: Push Button Cells

#### 4.1.4.1.14 Rich Text

The **Rich Text** control will allow you to display and edit Rich Text in grid cells. The control enables you to optionally drop down an editable Rich Text window using which you can modify the Rich Text in the cell.

The following code example illustrates how to set the cell type to RichText.

##### [C#]

```
// Create a Rich Text Format.
string rtf =
@ "{" +
@"\rtf1\ansi\deff0\deftab720" +
@ "{" +
@ "\fonttbl" +
@ "{\f0\fswiss MS Sans Serif;}" +
@ "{\f1\froman\fcharset2 Symbol;}" +
@ "{\f2\fswiss\fprq2 System;}" +
@ "{\f3\fswiss\fprq2 Arial;}" +
@ "{\f4\froman Bookman Old Style;}" +
@ "}" +
@ "{\colortbl\red0\green0\blue0;\red255\green0\blue0;}" +
@ "\deflang1033\cfpat1\pard\plain\f3\fs16\cf0 * Change the " +
@ "\plain\f4\fs24\cf0\b\i\ul font \plain\f3\fs16\cf0 or " +
@ "\plain\f4\fs24\cf1\b\ul color\plain\f3\fs16\cf0 " +
@ "for individual characters.\par " +
" }"
;

// Set up a Rich Text Cell.
gridControl1[rowIndex, 1].CellType = "RichText";
gridControl1[rowIndex, 1].Text = rtf;
gridControl1.RowHeights[rowIndex] = 50;
gridControl1.CoveredRanges.Add(GridRangeInfo.Cells(rowIndex, 1,
rowIndex, 5));
```

##### [VB .NET]

```
' Create a Rich Text Format.
Dim rtf As String = "{" +
"\rtf1\ansi\deff0\deftab720" +
"{" +
"\fonttbl" +
"{\f0\fswiss MS Sans Serif;}" +
```

```

"\"f1\froman\fcharset2 Symbol;" + _
"\"f2\fswiss\fprq2 System;" + _
"\"f3\fswiss\fprq2 Arial;" + _
"\"f4\froman Bookman Old Style;" + _
"}" + _
"\"{\\colortbl\\red0\\green0\\blue0;\\red255\\green0\\blue0;}" + _
"\deflang1033\cfpat1\pard\plain\f3\fs16\cf0 * Change the " + _
"\plain\f4\fs24\cf0\b\i\ul font \plain\f3\fs16\cf0 or " + _
"\plain\f4\fs24\cf1\b\ul color\plain\f3\fs16\cf0 " + _
"for individual characters.\par " + _
" }"

' Set up a Rich Text Cell.
gridControl1(rowIndex, 1).CellType = "RichText"
gridControl1(rowIndex, 1).Text = rtf
gridControl1.RowHeights(rowIndex) = 50
gridControl1.CoveredRanges.Add(GridRangeInfo.Cells(rowIndex, 1,
rowIndex, 5))

```

The screenshot shows a Windows application window with a grid control. The grid has 5 columns labeled A, B, C, D, and E. Row 1 contains the number 1. Row 2 contains the number 2. Row 3 contains the text "\* Change the **font** or **color** for individual characters." The cell in row 3, column D, is highlighted with a red border.

Figure 90: Rich Text Cells

#### 4.1.4.1.15 Slider

You can use slider cells in grid cells. You can also share a single Slider control among multiple cells. To set the slider properties for a cell, make use of the **SliderStyleProperties** object.

The following code example illustrates how to set the cell type to Slider.

```

[C#]

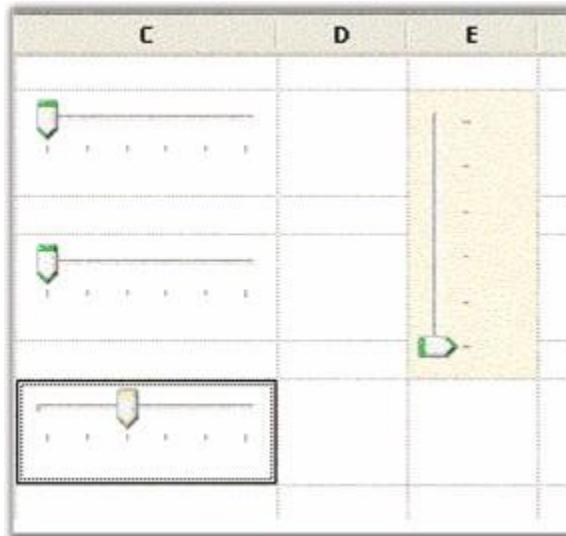
// Set up a Slider control.
GridStyleInfo style = gridControl1[row, 3];
SliderStyleProperties sp = new SliderStyleProperties(style);
style.CellType = "Slider";

```

```
// Set Slider Properties.  
sp.Maximum = 40;  
sp.Minimum = 0;  
sp.TickFrequency = 8;  
sp.LargeChange = 16;  
sp.SmallChange = 4;
```

**[VB.NET]**

```
' Set up a Slider control.  
Dim style As GridStyleInfo = gridControl1(row, 3)  
Dim sp As SliderStyleProperties = New SliderStyleProperties(style)  
style.CellType = "Slider"  
  
' Set Slider Properties.  
sp.Maximum = 40  
sp.Minimum = 0  
sp.TickFrequency = 8  
sp.LargeChange = 16  
sp.SmallChange = 4
```



*Figure 91: Slider Cells*

#### 4.1.4.1.16 Static

A **Static** cell type will display text that cannot be edited. You can select it to make it the current cell, but the cell cannot be activated for editing. Static cells can be deleted by the user, if the static cells are part of the selection when the **DELETE** key is pressed (To prevent this deletion behavior, set static cells to **ReadOnly**). Static cells may also include an image in addition to the text.

The following code example illustrates how to set the cell type to Static.

[C#]

```
// Use a static cell.  
gridControl1[rowIndex,colIndex].CellType = "Static";  
gridControl1[rowIndex,colIndex].Text = "Static";  
  
// Use a static cell with an image.  
// Create an image list and add some images during the initialization.  
ImageList imageList1 = new ImageList();  
imageList1.Images.Add(SystemIcons.Warning.ToBitmap());  
imageList1.Images.Add(SystemIcons.Application.ToBitmap());  
imageList1.Images.Add(SystemIcons.Asterisk.ToBitmap());  
imageList1.Images.Add(SystemIcons.Error.ToBitmap());  
  
// Set the imagelist into the TableStyle.  
gridControl1.TableStyle.ImageList = imageList1;  
  
// To use an image, set the ImageIndex in the cell GridInfoStyle.  
gridControl1[rowIndex,colIndex + 1].CellType = "Static";  
gridControl1[rowIndex,colIndex + 1].Text = "Static2";  
  
// Show the third icon in the imagelist which is inherited from the  
TableStyle.  
gridControl1[rowIndex,colIndex + 1].ImageIndex = 2;
```

[VB.NET]

```
' Use a static cell.  
gridControl1(rowIndex, colIndex).CellType = "Static"  
gridControl1(rowIndex, colIndex).Text = "Static"  
  
' Use a static cell with an image.  
' Create an image list and add some images during the initialization.  
Dim imageList1 As New ImageList()  
imageList1.Images.Add(SystemIcons.Warning.ToBitmap())  
imageList1.Images.Add(SystemIcons.Application.ToBitmap())  
imageList1.Images.Add(SystemIcons.Asterisk.ToBitmap())
```

```
imageList1.Images.Add(SystemIcons.Error.ToBitmap())  
  
' Set the imagelist into the TableStyle.  
gridControl1.TableStyle.ImageList = imageList1  
  
' To use an image, set the ImageIndex in the cell GridInfoStyle.  
gridControl1(rowIndex, colIndex + 1).CellType = "Static"  
gridControl1(rowIndex, colIndex + 1).Text = "Static2"  
  
' Show the third icon in the imagelist which, is inherited from  
TableStyle.  
gridControl1(rowIndex, colIndex + 1).ImageIndex = 2
```



Figure 92: Static Cells

#### 4.1.4.1.17 Text Box

A **Text Box** cell type displays text and images that can be edited in place.

The following code example illustrates how to set the cell type to TextBox.

##### [C#]

```
gridControl1[rowIndex,colIndex].Text = "TextBox";  
gridControl1[rowIndex,colIndex].CellType = "TextBox";  
  
// Text box with image - assumes ImageList set the same Static sample code.  
gridControl1[rowIndex,colIndex + 1].Text = "TextBox/Image";  
gridControl1[rowIndex,colIndex].CellType = "TextBox";  
gridControl1[rowIndex,colIndex + 1].ImageIndex = 1;
```

##### [VB .NET]

```
gridControl1(rowIndex, colIndex).Text = "TextBox"  
gridControl1(rowIndex, colIndex).CellType = "TextBox"  
  
' Text box with image - assumes ImageList set the same Static sample code.  
gridControl1(rowIndex, colIndex + 1).Text = "TextBox/Image"
```

```
gridControl1(rowIndex, colIndex).CellType = "TextBox"  
gridControl1(rowIndex, colIndex + 1).ImageIndex = 1
```



Figure 93: Text Box Cells

#### 4.1.4.2 Cell Style Architecture

The Essential Grid's cell style architecture plays an integral role in almost every aspect of Essential Grid. A basic understanding of this layered cell style architecture will help you understand and learn the grid behavior. This is particularly important when you are trying to modify or extend some existing functionality.

##### 4.1.4.2.1 GridStyleInfo Class Overview

Grid control can be thought of as a rectangular table of grid cells. Each cell contains distinct information and can be displayed independently of other cells. EssentiGrid uses **GridStyleInfo** objects to store state information about the appearance of a grid cell. So attributes like **font**, **backcolor**, **cellvalue** and **celltype** are all reflected in a single GridStyleInfo object.

Every cell in a grid may have such an object associated with it, giving the individual cell its unique appearance. It is not necessary that all cells should require fully populated GridStyleInfo objects stored in memory to function. And, for a given GridStyleInfo object, not all possible properties need to be populated in the object. So for example, a particular cell may or may not have a stored GridStyleInfo object, and if it does, this GridStyleInfo object may, or may not, contain a particular property such as Font.

In general, when Essential Grid needs a cell's state information, usually to draw the cell, it uses an inheritance process to generate a GridStyleInfo from several parent styles. The following parent styles are GridStyleInfo objects associated with particular grid entities:

- **TableStyle** is a single GridStyleInfo object that is associated with the entire grid.
- **RowStyles** are GridStyleInfo objects that are associated with each row.
- **ColumnStyles** are GridStyleInfo objects that are associated with each column.

These three GridStyleInfo objects may not be fully populated, meaning that some properties may not have been set. However, there is a fourth parent style referred to as the **StandardStyle**, which is a fully populated style object, meaning every property has a setting in the StandardStyle.

So when Grid control needs to generate a composite GridStyleInfo object for a particular cell, it first looks at any property that may be specifically set in a stored cell GridStyleInfo (if one exists) for this cell. If there are properties not set in this cell-specific GridStyleInfo object, Grid control will then pick up the rowstyle GridStyleInfo for this cell. From this rowstyle, it will populate any property that was explicitly set in the RowStyle, and those that were not explicitly set in the cell-specific GridStyleInfo object. After adding on unset properties to the composite GridStyleInfo from the RowStyle, it does the same for the columnstyle, the tablestyle, and finally the standardstyle. In this manner, Grid control comes up with a fully populated composite GridStyleInfo object to use.

The following graphic illustrates the effect of using the GridStyleInfo inheritance to come up with the appearance of a cell 3, 2. Even though the BackColor property is set in each of the tablestyle, rowstyle and columnstyle objects, it is the cell specific style that determines the back color of the cell.

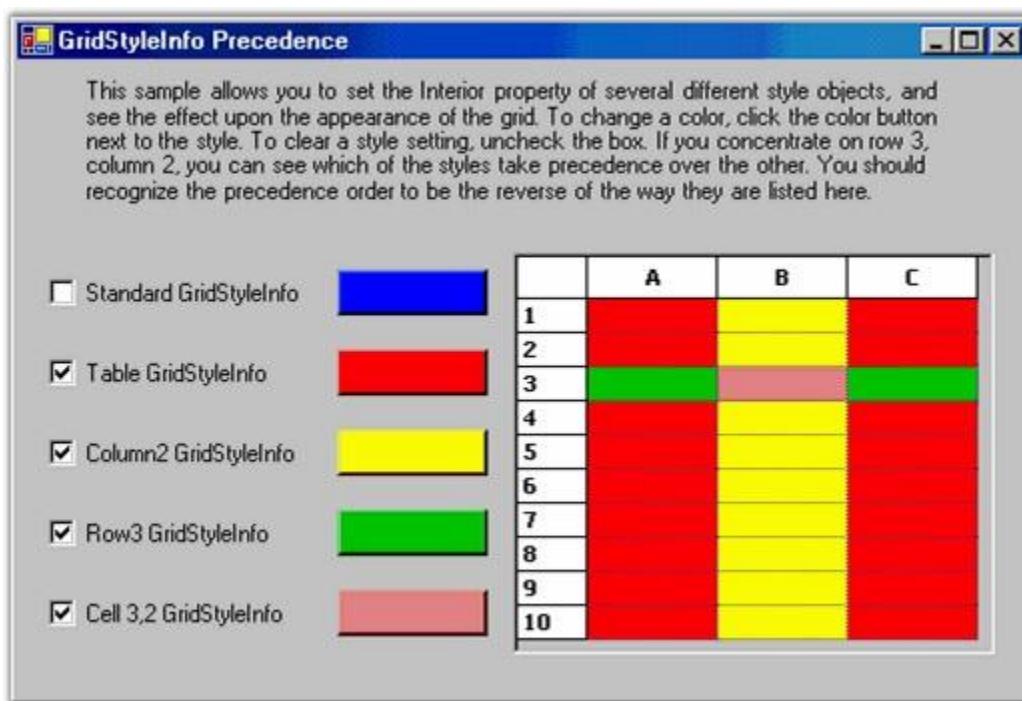


Figure 94: GridStyleInfo

The next graphic shows the effect of removing the BackColor property from the cell specific style. In this case, it is the rowstyle that determines the back color setting for the displayed cell. If you remove the rowstyle setting for BackColor, then the columnstyle would contribute its BackColor property to determine the cell's displayed color. Run the GridStyleInfo sample to experiment using the different parent styles.

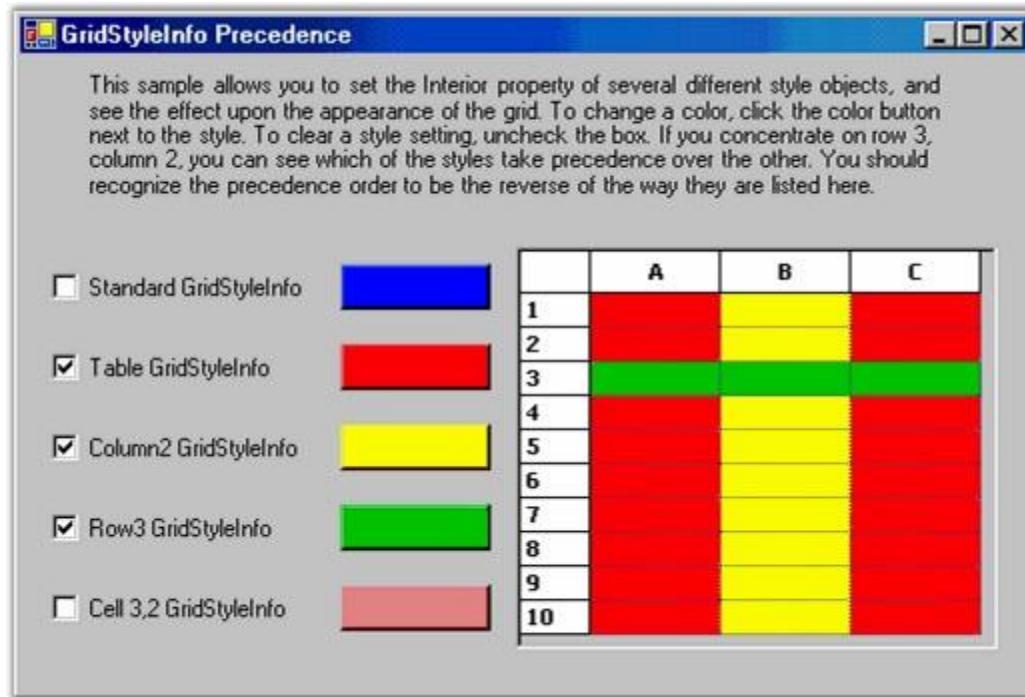


Figure 95:: Removing BackColor from the Cell Specific Style

## See Also

### [4.1.4.2.1.1 Properties](#)

GridStyleInfo provides many properties to control the appearance and behavior of grid cells. The following table lists some of the properties.

GridStyleInfo Property	Description
String GridStyleInfo.Text	Formatted string value of the cell.
Object GridStyleInfo.CellValue	Value of the object stored in the cell.
BrushInfo GridStyleInfo.BackColor	Back color of the cell.
Color GridStyleInfo.TextColor	Color of the displayed text.
GridFontInfo GridStyleInfo.Font	Font used to display the text.

ImageList GridStyleInfo.ImageList	Holds a list of images for use by the cell.
Int GridStyleInfo.ImageIndex	Picks a particular image from the ImageList property.



**Note:** Refer the *GridStyleInfo* topic in the *Essential Grid Class Reference* for a complete description of all the *GridStyleInfo* class members.

#### 4.1.4.2.1.1.1 BackColor

The **BackColor** property specifies the background color for the cell. If you want to use a special brush to get a gradient background, you can use the **Interior** property of *GridStyleInfo* to specify a brush that can be used to draw the cell background.

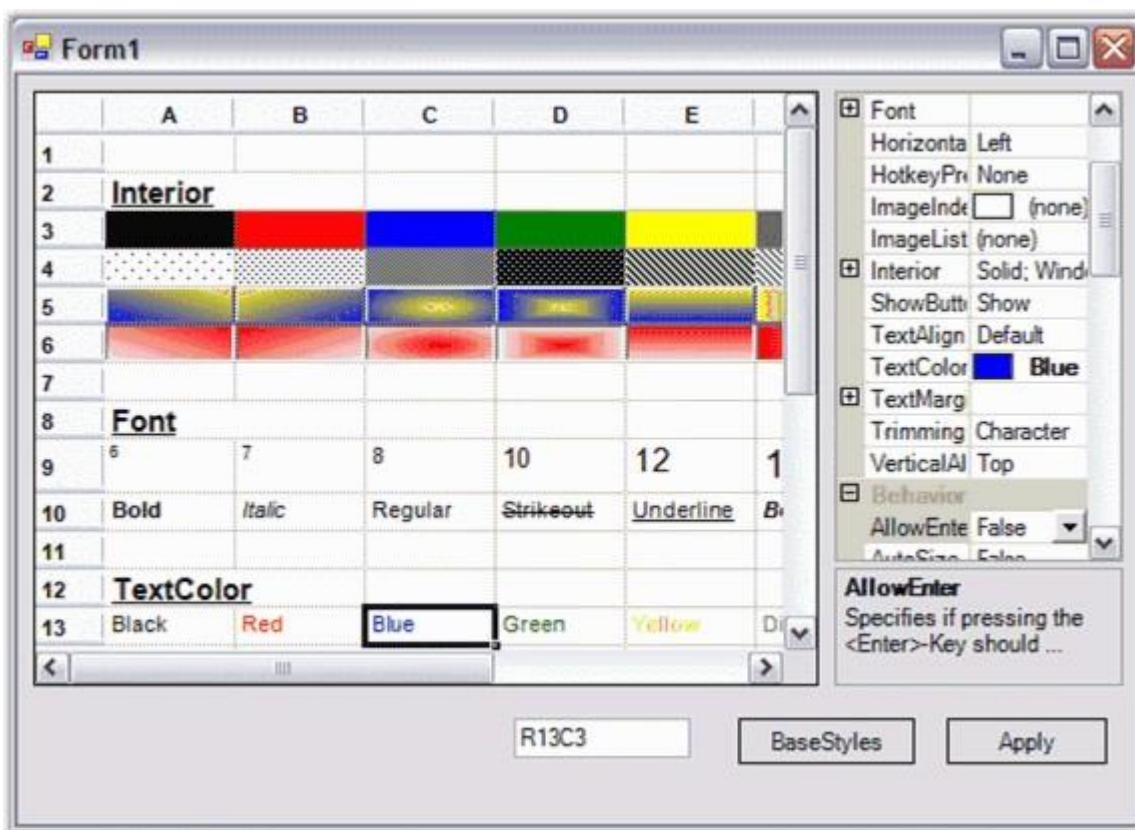


Figure 96: Interior Property

#### 4.1.4.2.1.1.2 GridFontInfo

The **GridFontInfo** class is an Essential Grid wrapper class for the standard **System.Drawing.Font** class. The **Font** property of the **GridStyleInfo** class specifies the font for the text displayed in the cell. The **GridFontInfo** class has special static members that enable you to easily modify font property members.

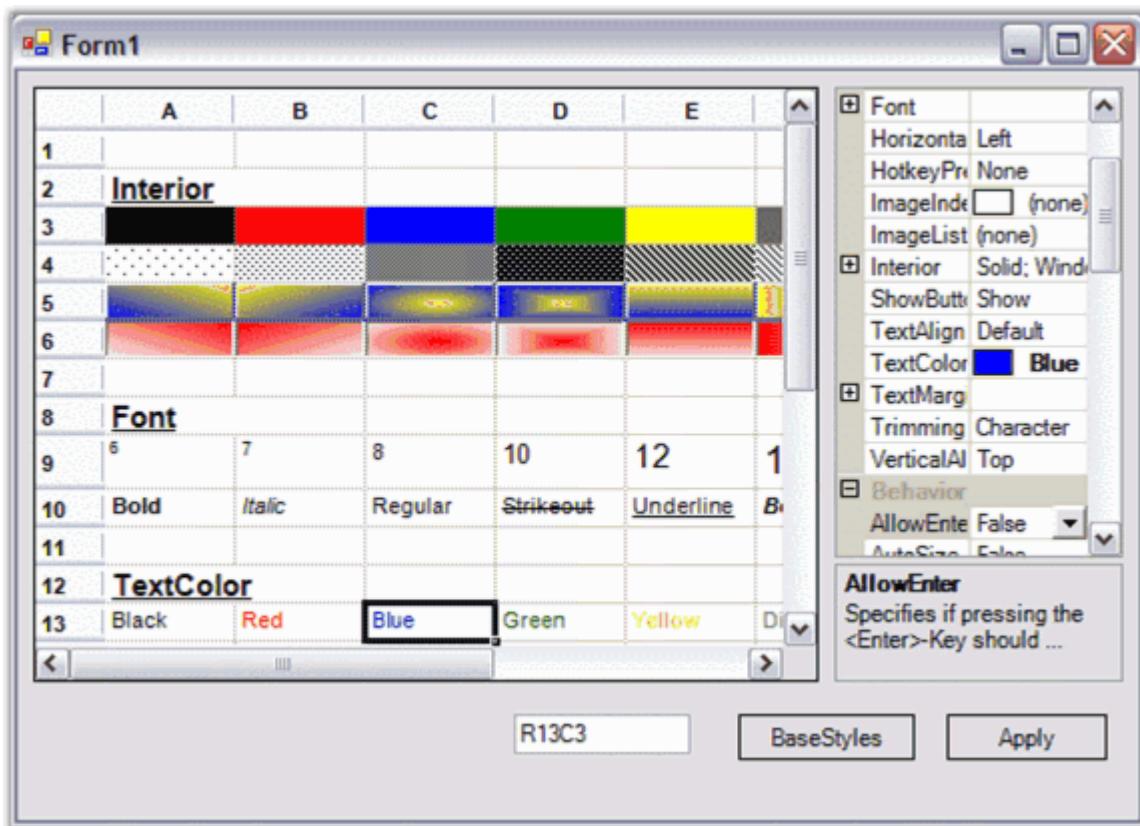


Figure 97: GridFontInfo

#### 4.1.4.2.1.1.3 ImageList

The **ImageList** property holds a **System.Windows.Forms.ImageList**. Generally, there is one **ImageList** stored in a parent **GridInfoStyle** such as the **standardstyle** or the **tablestyle**. This single **ImageList** is shared by all cells in the grid through the **ImageIndex** property, which has been set on a cell-by-cell basis.

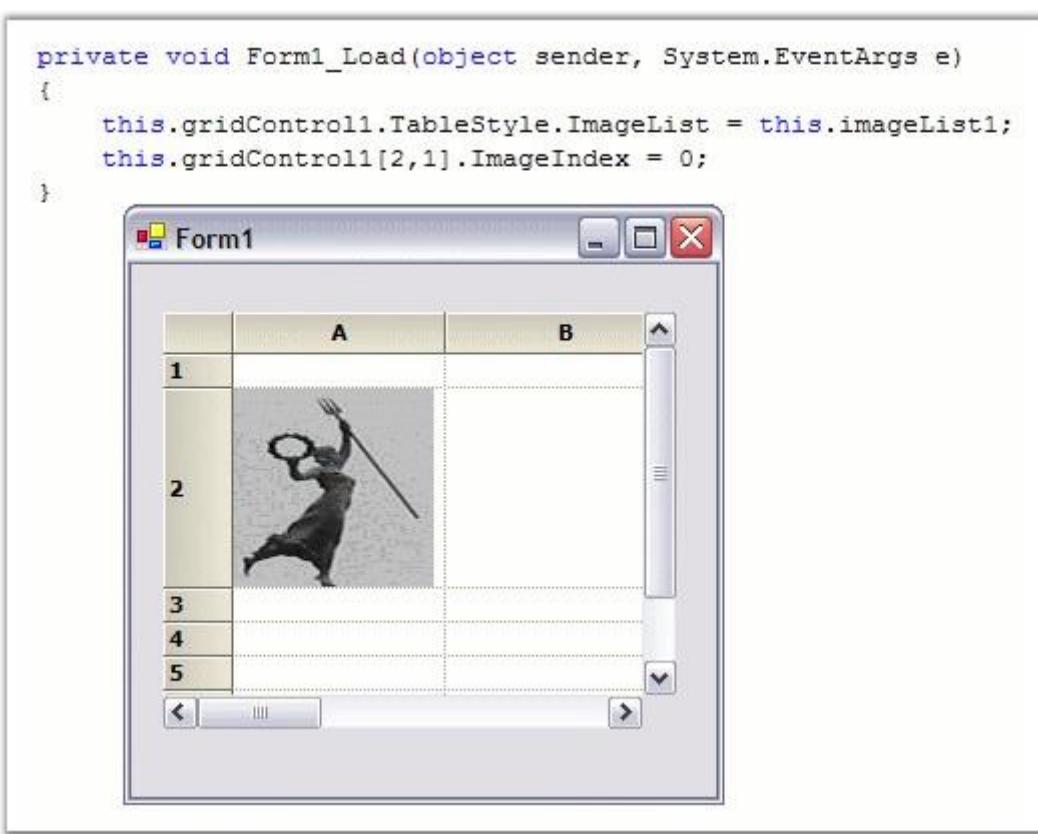


Figure 98: Displaying an Image in a Cell by using `ImageList` and `ImageIndex` Properties

#### 4.1.4.2.1.1.4 Text and CellValue

The **Text** and **CellValue** properties are closely related. You can set the value of either by using the other. The major difference is that the **Text** property is a string and the **CellValue** property is an object. This means, for example, that you can assign a **DateTime** object to a cell value, but you cannot assign it to a text. Grid control generally sets the **Text** property by using the **CultureInfo** formatting on the **CellValue** property. The **Text** property can also be set directly through code.

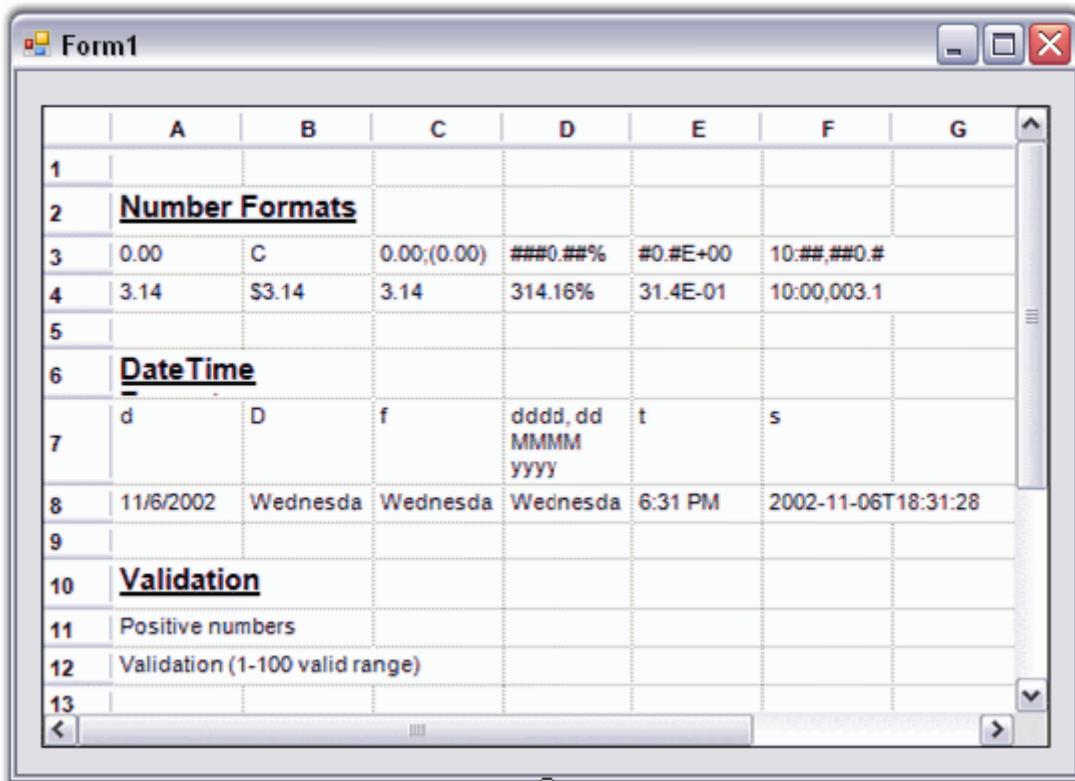


Figure 99: Text and Cell Value

#### 4.1.4.2.2 BaseStyles

Grid control supports another parent-type style, **BaseStyles**, which is used to customize a cell's appearance. **BaseStyles** are **GridStyleInfo** objects which can be associated with an arbitrary collection of cells. In a Word Processing software, there is the common task of defining a particular style (such as style Header1 representing a bold, 20-point Helvetica font), and then using it repeatedly in your document whenever you need a 'Header1' type. **BaseStyles** play the same role within Grid control. You can define a **BaseStyle** named Header1 as having certain properties, and then you can place these properties onto any cell just by applying this **BaseStyle** Header1 to the cell. More importantly, if you want to change Header1 (for example, changing its **BackColor** property from white to red), you can make the change one time by just changing the Header1 **BaseStyle**, and not having to relabel every other cell assigned to this **BaseStyle**.

Since **BaseStyles** are considered as parent styles, where do they fit within the precedence hierarchy that we have discussed above? **BaseStyles** are applied between the **TableStyle** and the **StandardStyle**. Thus, they are the 'weakest' style other than the fully populated **StandardStyle**. **BaseStyles** are stored in the **GridControl.BaseStylesMap** class. In addition to the **StandardStyle**, other **BaseStyles** used by all Essential Grids include Row Header, Header and Column Header. You can define and apply your own **BaseStyles** as well.

To work with BaseStyles from within the Visual Studio designer, you need to use the **Edit base styles** verb that appears at the bottom of the Grid control's property grid.



Figure 100: PropertyGrid with Edit BaseStyles Option

When you click the **Edit base styles** verb, the **GridBaseStyle Collection Editor** dialog box is displayed. You can use the GridBaseStyle Collection Editor to edit the existing BaseStyles or add new ones.

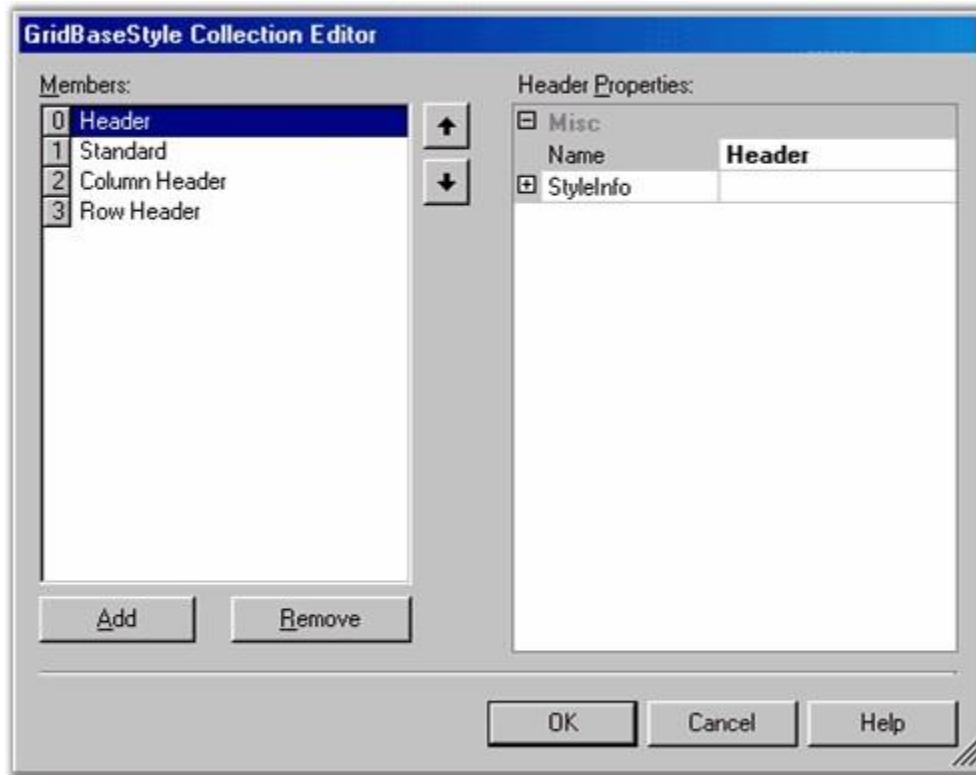


Figure 101: GridBaseStyle Collection Editor

The following code example illustrates how to create a BaseStyle. When you define a BaseStyle you can apply it to any cell (or row or column) by just setting the **GridStyleInfo.BaseStyle** for that cell to the name used to define the BaseStyle.

**[C#]**

```
// Add a new base style.
GridBaseStyle gridBaseStyle1 = new GridBaseStyle("BackColorTest", false);
gridBaseStyle1.StyleInfo.BackColor = Color.SkyBlue;
gridBaseStyle1.StyleInfo.TextColor = Color.RosyBrown;
gridControl1.BaseStylesMap.AddRange(new GridBaseStyle[]{gridBaseStyle1});
...

// Apply this base style to a couple of cells.
gridControl1[1,2].BaseStyle = "BackColorTest";
gridControl1[4,2].BaseStyle = "BackColorTest";
```

**[VB .NET]**

```
' Add a new base style.
Dim gridBaseStyle1 As GridBaseStyle = New GridBaseStyle("BackColorTest", False)
gridBaseStyle1.StyleInfo.BackColor = Color.SkyBlue
```

```

gridBaseStyle1.StyleInfo.TextColor = Color.RosyBrown
gridControl1.BaseStylesMap.AddRange(New GridBaseStyle() {gridBaseStyle1})
...
' Apply this base style to a couple of cells.
gridControl1(1, 2).BaseStyle = "BackColorTest"
gridControl1(4, 2).BaseStyle = "BackColorTest"

```

#### 4.1.4.2.3 GridRangeInfo

This class is used extensively to specify a collection of grid cells that are to be used as parameters for other method calls. **GridRangeInfo** class contains static methods that will allow you to specify a single cell, a rectangular range of cells, a row or rows, a column or columns, or the entire table.

GridRangeInfo Method	Description
GridRangeInfo.Cell(int row, int col)	Returns the GridRangeInfo object with cell row, col.
GridRangeInfo.Cells(int top, int left, int bottom, int right)	Returns a GridRangeInfo object containing a rectangular collection of cells with top left cell (top, left) and bottom right cell (bot, right).
GridRangeInfo.Row(int row)	Returns GridRangeInfo object with row = row.
GridRangeInfo.Rows(int fromRow, int toRow)	Returns a GridRangeInfo object containing rows fromRow through toRow.
GridRangeInfo.Col(int col)	Returns GridRangeInfo object with column col.
GridRangeInfo.Cols(int fromCol, int toCol)	Returns a GridRangeInfo object containing columns fromCol through toCol.
GridRangeInfo.Table()	Returns a GridRangeInfo object containing the whole table.



**Note:** For a complete description of the **GridRangeInfo** class, see the **Essential Grid Class Reference**.

#### 4.1.4.2.4 The GridControl.ChangeCells Method

The **GridControl.ChangeCells** method is used to modify GridStyleInfo objects. This overloaded method accepts GridRangeInfo and GridStyleInfo objects. The ChangeCells method depends upon the value of the **ModifyType** parameter. The current GridStyleInfo settings are modified by using the new GridStyleInfo settings that are present in the **CellInfo** parameter, according to the value of the ModifyType parameter.

##### [C#]

```
// Apply an array of styles to the specified range in grid.
public bool ChangeCells(GridRangeInfo range, GridStyleInfo cellInfo,
StyleModifyType modifyType);
```

##### [VB .NET]

```
' Apply an array of styles to the specified range in grid.
Public Function ChangeCells(range As GridRangeInfo, cellInfo As GridStyleInfo,
modifyType As StyleModifyType) As Boolean
```

#### 4.1.4.2.5 Activating Current Cell Behavior

When moving the current cell or clicking inside a cell, you can control the current cell's activation behavior by using the **ActivateCurrentCellBehavior** property. The **GridCellActivateAction** enumeration defines when to set the focus on the cell or toggle to edit mode for the current cell.

Here is the list of options under the GridCellActivateAction enumeration:

- **ClickOnCell**-Setting ActivateCurrentCellBehavior to this option sets the cell to editing mode or sets focus on the cell after user clicks the cell.
- **DbClickOnCell**-Setting ActivateCurrentCellBehavior to this option sets the cell to editing mode or sets focus on the cell when user double clicks the cell.
- **None**-Setting ActivateCurrentCellBehavior to this option deactivates the cell, even if the user clicks it.
- **PositionCaret**-Setting ActivateCurrentCellBehavior to this option sets the caret to be positioned at the character where the user clicks.
- **SelectAll**-Setting ActivateCurrentCellBehavior to this option sets the cell to editing mode or sets focus on the cell and keeps the entire text in the cell selected whenever it becomes the current cell irrespective of the click on the cell or movement over it using arrow keys.
- **SetCurrent**-Setting ActivateCurrentCellBehavior to this option sets the cell to editing mode or sets focus on the cell whenever it becomes the current cell irrespective of the click on the cell or movement over it using arrow keys.

The following code examples illustrate how to set the ActivateCurrentCellBehavior property:

1. Using C#

[C#]

```
this.gridControl1ActivateCurrentCellBehavior =  
GridCellActivateAction.SelectAll;
```

2. Using VB.NET

[VB .NET]

```
Me.gridControl1ActivateCurrentCellBehavior =  
GridCellActivateAction.SelectAll
```

#### 4.1.4.2.6 Scroll Cell into View

You can use the grid method, **ScrollCellInView**, to scroll the specified cell or range into view. The range that should be scrolled into the visible grid view area is given as the parameter to the method. The following code examples illustrate this:

1. Using C#

[C#]

```
// Scroll into view cell(2,2).  
this.gridControl1.ScrollCellInView(GridRangeInfo.Cell(2, 2));  
  
// Scroll into view range Col(2).  
this.gridControl1.ScrollCellInView(GridRangeInfo.Col(2));
```

2. Using VB.NET

[VB .NET]

```
' Scroll into view cell(2,2).  
Me.gridControl1.ScrollCellInView(GridRangeInfo.Cell(2, 2))  
  
' Scroll into view range Col(2).  
Me.gridControl1.ScrollCellInView(GridRangeInfo.Col(2))
```

#### 4.1.4.2.7 Managing Current Cell Operations

**GridCurrentCell** class provides storage for current cell information and manages all the current cell operations such as activating, deactivating, saving, editing and moving the current cell.

The following code examples illustrate how to set a GridCurrentCell:

##### 1. Using C#

[C#]

```
GridCurrentCell cc = this.gridControl1.CurrentCell;
```

##### 2. Using VB.NET

[VB .NET]

```
Dim cc As GridCurrentCell = Me.gridControl1.CurrentCell
```

#### 4.1.4.2.8 Show/Hide Current Cell Border

**ShowCurrentCellBorderBehavior** property of the grid determines the behavior of the current cell's border. The **GridShowCurrentCellBorder** enumeration specifies display of current cell's frame or border.

Here is the list of options in GridShowCurrentCellBorder enumeration.

- **AlwaysVisible**-Setting ShowCurrentCellBorderBehavior property with this option displays the current cell borders/frame.
- **GrayWhenLostFocus**-Setting ShowCurrentCellBorderBehavior property with this option shows the current cell's borders in gray when it is not focused upon.
- **HideAlways**-Setting ShowCurrentCellBorderBehavior property with this option hides the borders of the current cell.
- **WhenGridActive**-Setting ShowCurrentCellBorderBehavior property with this option highlights the current cell's border when the grid is under focus.

The following code examples illustrate how to set the ShowCurrentCellBorderBehavior property:

##### 1. Using C#

[C#]

```
this.gridControl1.ShowCurrentCellBorderBehavior =  
GridShowCurrentCellBorder.AlwaysVisible;
```

## 2. Using VB.NET

[VB .NET]

```
Me.gridControl1.ShowCurrentCellBorderBehavior =  
GridShowCurrentCellBorder.AlwaysVisible
```

### 4.1.4.2.9 Refreshing Behavior of Current Cell

The Grid property, **RefreshCurrentCellBehavior**, determines the behavior of refreshing the cells while the focus is moved from current cell to another. The **GridRefreshCurrentCellBehavior** enumeration specifies which cells to refresh when the focus is moved from current cell to another.



**Note:** Refreshing behavior of the cells enables them to display the current data automatically after updates.



**Note:** Refreshing the cells denote reloading the cell's value.

Following are the list of options provided by the GridRefreshCurrentCellBehavior enumeration.

- **None**-Setting ShowCurrentCellBorderBehavior property with this option does not initiate refresh when moving the current cell.
- **RefreshCell**-Setting ShowCurrentCellBorderBehavior property with this option refreshes the current cell only.
- **RefreshRow**-Setting ShowCurrentCellBorderBehavior property with this option refreshes the entire row to which the current cell belongs. Use this setting if you are using GridShowButtons.ShowCurrentRow.

The following code examples illustrate how to set the RefreshCurrentCellBehavior property:

## 1. Using C#

[C#]

```
this.gridControl1.RefreshCurrentCellBehavior =
```

```
GridRefreshCurrentCellBehavior.RefreshCell;
```

## 2. Using VB.NET

**[VB.NET]**

```
Me.gridControl1.RefreshCurrentCellBehavior =  
GridRefreshCurrentCellBehavior.RefreshCell
```

### 4.1.4.2.10 Style Properties

This section provides information on the following topics:

#### 4.1.4.2.10.1 *GridStyleInfo* Properties

**GridStyleInfo** class comprises properties that let users to control the appearance and behavior of grid cells.

	A	B	C	D	E	F	G	
1								
2	Interior							
3	Black	Red	Blue	Green	Yellow			
4	Horizontal	Diagonal	Vertical	Dot	Checkered	Grid		
5	Brown	Brown	Brown	Brown	Brown	Brown		
6	Brown	Brown	Brown	Brown	Brown	Brown		
7								
8	Font							
9	6	7	8	10	12	14		
10	<b>Bold</b>	<i>Italic</i>	Regular	<del>Strikeout</del>	<u>Underline</u>	<b><i>Bold, Italic</i></b>		
11								
12	TextColor							
13	Black	Red	Blue	Green	Yellow	DimGray		
14								
15	Borders							
16	Solid; Black	Dotted; Red	DashDot;	DashDotD	Dashed; Black	Dotted; Red; Medium		
17	Dashed; Black	Dotted; Red	DashDot;	DashDotD	Solid; Black	Solid; Red; Medium		
18								
19	Orientation							
20	Angle = 0	Angle = 45	Angle = 60	Angle = 90	Angle = 180	Angle = 270		

Figure 102: GridStyleInfo Properties

The above screen shot provides information on the following properties:

1. Interior

Lets you specify a solid, gradient, or pattern style for a cell's background.

The grid cells can be painted by using the **Interior** property under **Syncfusion.Drawing.BrushInfo** class. BrushInfo holds information on filling the background of a grid cell. **PatternStyle** specifies the pattern style to be used and **GradientStyle** specifies the gradient style to be used.

- a. Using C#

```
[c#]
```

```
gridControl1[2, 2].Interior = new BrushInfo(GradientStyle.Horizontal,  
Color.Yellow, Color.Blue);  
gridControl1[3, 2].Interior = new  
BrushInfo(PattenStyle.DashedHorizontal, Color.Black, Color.White);
```

b. Using VB.NET

**[VB.NET]**

```
gridControl1(2, 2).Interior = New BrushInfo(GradientStyle.Horizontal,  
color.Yellow, color.Blue)  
gridControl1(3, 2).Interior = New  
BrushInfo(PattenStyle.DashedHorizontal, color.Black, color.White)
```

2. Font

Lets you specify the font for drawing text. The cells can be given required styles by using the **Font** property under **GridFontInfo**. GridFontInfo holds information on the font settings.

a. Using C#

**[C#]**

```
GridFontInfo boldFont = new GridFontInfo();  
boldFont.Bold = true;  
boldFont.Size = 11;  
boldFont.Underline = true;  
gridControl1[3, 4].Font = boldFont;
```

b. Using VB.NET

**[VB.NET]**

```
Dim boldFont As GridFontInfo = New GridFontInfo()  
boldFont.Bold = True  
boldFont.Size = 11  
boldFont.Underline = True  
gridControl1(3, 4).Font = boldFont
```

3. Text Color

The color for the cell text can be set by using the **TextColor** property.

a. Using C#

[C#]

```
gridControl1[rowIndex, colIndex].TextColor = Color.Red;
```

b. Using VB.NET

[VB .NET]

```
gridControl1(rowIndex, colIndex).TextColor = color.Red
```

4. Border

The border can be set to all sides of a cell by setting the **Border** property to an instance of **GridBorder**. The **GridBorder** class holds the formatting information for borders of the cell.

a. Using C#

[C#]

```
gridControl1[rowIndex, colIndex].Borders.All = new  
GridBorder(GridBorderStyle.DashDotDot, Color.Red);
```

b. Using VB.NET

[VB .NET]

```
gridControl1(rowIndex, colIndex).Borders.All = New  
GridBorder(GridBorderStyle.DashDotDot, color.Red)
```

5. Orientation

Lets you specify the orientation of the grid cell text, inturn specifying the angle at which the text is displayed.

a. Using C#

[C#]

```
gridControl1[3, 4].Font.Orientation = 270;
```

b. Using VB.NET

[VB .NET]

```
gridControl1[3, 4].Font.Orientation = 270
```

A sample demonstrating this feature is available under the following sample installation path.

**C:\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Windows\Samples\2.0\Appearance\Cell Style Demo**

#### 4.1.4.2.10.2 Custom Borders

You can draw custom borders around cells by using the **DrawCellFrameAppearance** event of the Grid.

The **DrawCellFrameAppearance** event is triggered for every cell, before the grid draws the frame of a specified cell and after the cell's background is drawn. This event can be used with any cell type such as TextBox, CheckBox, and so on. You can draw texture-brush border and gradient borders.

The following code examples illustrate drawing custom borders by using the **DrawCellFrameAppearance** event:

##### 1. Using C#

[C#]

```
private void grid_DrawCellFrameAppearance(object sender,
GridDrawCellEventArgs e)
{
    // Draw a custom cell frame/border.
    int rowIndex = e.Style.CellIdentity.RowIndex;
    int colIndex = e.Style.CellIdentity.ColIndex;
    if (rowIndex > 0 && colIndex > 0)
    {
        Brush brush;
        Graphics g = e.Graphics;

        // Allocate and cache bitmap and texture brush.
        if (tb == null)
        {
            if (backBmp == null)
                backBmp = GetImage("back3.jpg");
            tb = new TextureBrush(backBmp);
        }
        g.FillRectangle(tb, e.Bounds);
        g.DrawRectangle(new Pen(Color.Black), e.Bounds);
    }
}
```

```
}

// Use TextureBrush for top-left cells.
if (colIndex < 6 && rowIndex < 12)
brush = tb;
else

// Otherwise use a gradient brush.
brush = new
System.Drawing.Drawing2D.LinearGradientBrush(e.TargetBounds,
Color.FromArgb( 204, 212, 230 ), Color.FromArgb( 252, 172, 38 ),
45f);

// Draw custom border for the cell.
// Space has been reserved for this area with the
TableStyle.BorderMargins property.
Rectangle rect = e.TargetBounds;
rect.Inflate(-2, -2);
Rectangle[] rects = new Rectangle[]
{
    new Rectangle(rect.X, rect.Y, rect.Width, 4),
    new Rectangle(rect.X, rect.Y, 4, rect.Height),
    new Rectangle(rect.Right-4, rect.Y, 4, rect.Height),
    new Rectangle(rect.X, rect.Bottom-4, rect.Width, 4),
};
g.FillRectangles(brush, rects);

// Disallow grid's default drawing of cell frame for this cell.
e.Cancel = true;
}
}
```

## 2. Using VB.NET

```
[VB.NET]

Private Sub grid_DrawCellFrameAppearance(ByVal sender As Object, ByVal
e As GridDrawCellEventArgs)

    ' Draw a custom cell frame/border.
    Dim rowIndex As Integer = e.Style.CellIdentity.RowIndex
    Dim colIndex As Integer = e.Style.CellIdentity.ColIndex
    If rowIndex > 0 AndAlso colIndex > 0 Then
        Dim brush As Brush
        Dim g As Graphics = e.Graphics
```

```
' Allocate and cache bitmap and texture brush.  
If tb Is Nothing Then  
    If backBmp Is Nothing Then  
        backBmp = GetImage("back3.jpg")  
    End If  
    tb = New TextureBrush(backBmp)  
End If  
  
' Use TextureBrush for top-left cells.  
If colIndex < 6 AndAlso rowIndex < 12 Then  
    brush = tb  
Else  
  
    ' Otherwise use a gradient brush.  
    brush = New  
    System.Drawing.Drawing2D.LinearGradientBrush(e.TargetBounds,  
    Color.FromArgb(204, 212, 230), Color.FromArgb(252, 172, 38),  
    45.0F)  
End If  
  
' Draw custom border for the cell.  
' Space has been reserved for this area with the  
TableStyle.BorderMargins property.  
Dim rect As Rectangle = e.TargetBounds  
rect.Inflate(-2, -2)  
Dim rects() As Rectangle = New Rectangle() {New Rectangle(rect.X,  
rect.Y, rect.Width, 4), New Rectangle(rect.X, rect.Y, 4,  
rect.Height), New Rectangle(rect.Right - 4, rect.Y, 4,  
rect.Height), New Rectangle(rect.X, rect.Bottom - 4, rect.Width,  
4)}  
g.FillRectangles(brush, rects)  
  
' Disallow grid's default drawing of cell frame for this cell.  
e.Cancel = True  
End If  
End Sub
```

	A	B	C	D	E	F	G	H
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								
11								
12				<input checked="" type="checkbox"/> Check				
13								
14								

Figure 103: Custom Borders

A sample demonstrating this feature is available under the following sample installation path.

**C:\Syncfusion\EssentialStudio\Version  
Number\Windows\Grid.Windows\Samples\2.0\Appearance\Custom Border Demo**

#### 4.1.4.2.10.3 Number Formats

The formats of a numeric field (cell value) can be masked by using the **Format** object. You can specify numeric format string as a mask. Format mask objects are assigned to date and numeric fields, and are used to define how the data returned for that field is displayed.



**Note:** Format masks object cannot be deleted once assigned to a field.

The following code examples illustrate masking numeric fields by using the Format object:

## 1. Using C#

[C#]

```
this.gridControl1[2, 2].Format = "###0.##%";
```

## 2. Using VB.NET

[VB .NET]

```
Me.gridControl1(2, 2).Format = "###0.##%"
```

2	Number Formats				
3	0.00	C	0.00; (0.00)	###0.# #%	#0.#E+00
4	3.14	\$3.14	3.14	314.16%	31.4E-01

Figure 104: Number Formats

### 4.1.4.2.10.4 Cell Tips

**CellTipText** object lets you specify the ToolTip Text to be displayed when the mouse pointer is moved over a cell. Cell tip text can be set for rows, columns, tables and for individual cells.

The following code examples illustrate how to set cell tips by using the CellTipText object:

## 1. Using C#

[C#]

```
// Tip Text for cell (2,3).
this.gridControl1[2, 3].CellTipText = "TipText for cell 2,3";

// Tip Text for row 3.
this.gridControl1.RowStyle[3].CellTipText = "TipText for row 3";

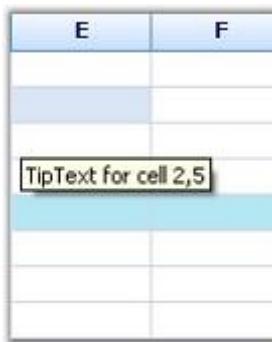
// Tip Text for column 4.
this.gridControl1.ColStyles[4].CellTipText = "TipText for column 4";
```

```
// Tip Text for table.  
this.gridControl1.TableStyle.CellTipText = "TipText for table";
```

## 2. Using VB.NET

[VB.NET]

```
' Tip Text for cell (2,3).  
Me.gridControl1(2, 3).CellTipText = "TipText for cell 2,3"  
  
' Tip Text for row 3.  
Me.gridControl1.RowStyle(3).CellTipText = "TipText for row 3"  
  
' Tip Text for column 4.  
Me.gridControl1.ColStyles(4).CellTipText = "TipText for column 4"  
  
' Tip Text for table.  
Me.gridControl1.TableStyle.CellTipText = "TipText for table"
```



E	F
	TipText for cell 2,5

Figure 105: Cell Tips

### 4.1.4.2.10.5 Cell Comment Tips

Excel-like Cell Comment Tips can be included in a Grid by deriving the mouse controller class. The comment text is a custom style property added to cells that hold comments. To change, add or delete a comment, right-click the cell or left-click the red corner.

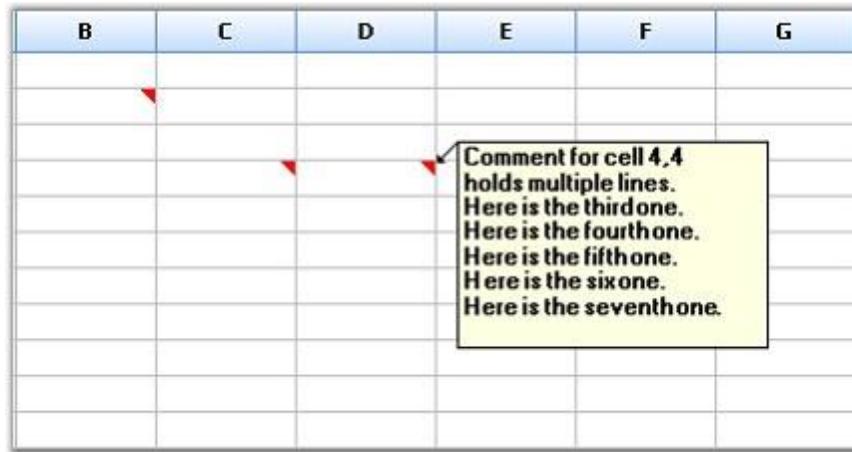


Figure 106: Cell Comment Tips

A sample demonstrating this feature is available under the following sample installation path.

**C:\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Windows\Samples\2.0\Appearance\Cell Comment Tip Demo**

#### 4.1.4.2.11 Grid New Feature

**ShowCurrentCellBorderBehavior** property of the grid determines the behavior of the current cell's border. The **GridShowCurrentCellBorder** enumeration specifies display of current cell's frame or border.

Here is the list of options in **GridShowCurrentCellBorder** enumeration.

- **AlwaysVisible**-Setting **ShowCurrentCellBorderBehavior** property with this option displays the current cell borders/frame.
- **GrayWhenLostFocus**-Setting **ShowCurrentCellBorderBehavior** property with this option shows the current cell's borders in gray when it is not focused upon.
- **HideAlways**-Setting **ShowCurrentCellBorderBehavior** property with this option hides the borders of the current cell.
- **WhenGridActive**-Setting **ShowCurrentCellBorderBehavior** property with this option highlights the current cell's border when the grid is under focus.

The following code examples illustrate how to set the **ShowCurrentCellBorderBehavior** property:

1. Using C#

[C#]

```
this.gridControl1.ShowCurrentCellBorderBehavior =  
GridShowCurrentCellBorder.AlwaysVisible;
```

2. Using VB.NET

[VB.NET]

```
' Set the ShowCurrentCellBorderBehavior property.  
Me.gridControl1.ShowCurrentCellBorderBehavior =  
GridShowCurrentCellBorder.AlwaysVisible
```

#### 4.1.4.2.12 Browse-Only Grid

The BrowseOnly property permits the Grid control to be set to a non-editable state without affecting its background appearance, displaying an unavailable effect, etc.

This feature is available in the GridDataBoundGrid and GridGrouping controls, and it is directly exposed under the controls' properties.

Applying this property will reflect in all cell types and will make the Grid non-editable. It affects the following cell types as follows:

- ComboBox—Items in the drop-down can be viewed but cannot be selected.
- CheckBox—Items cannot be selected or deselected.
- Textbox—Text cannot be entered in default edit mode

DropDown cells like MonthCalendar, ColorEdit, etc. will behave the same as the ComboBox cell type: items can be viewed but cannot be selected.

#### Properties

Property	Description	Type	Data Type
BrowseOnly	Gets or sets a value to determine whether the Grid should be available only for viewing and not for editing.	Boolean	true/false

#### 4.1.4.2.12.1 Sample Link

{installed  
drive}\AppData\Local\Syncfusion\EssentialStudio\{version}\Windows\Grid.Windows\Samples\2.0\  
Cell Types\Editor Cell Demo

#### 4.1.4.2.12.2 Making a Grid Browse-Only

To make the Grid non-editable, the following property must be set to true:

[C#]

```
this.gridControl1.BrowseOnly = true;
```

[VB]

```
Me.gridControl1.BrowseOnly = true
```

#### 4.1.4.3 Deriving a Cell Control

Derived cell controls can be used to add cells that have a special functionality. You can employ such controls to produce special behavior like a masked edit control, a draggable button or a tree node within a grid cell. Being able to extend a cell behavior through the derived cell controls makes Essential Grid very adaptable.

There are two classes involved in defining the cell architecture within Essential Grid.

**GridCellModelBase** serves as the base class for the first class which is involved in the cell model. **GridCellRendererBase** is the base class for the second class which is involved in defining a cell control, the cell renderer. For example, the Static cell control is defined in the **GridStaticCellModel** and **GridStaticCellRenderer** classes which are derived from these base classes. So, whether the cell control is a text box, a combo box or a NumericUpDown cell, the behavior is accomplished through these two classes which are derived from GridCellModelBase and GridCellRendererBase.

In the next sections, you will learn how to derive a cell control from the

**C:\Syncfusion\EssentialStudio\VersionNumber\Windows\Grid.Windows\Samples\[Version Number]\CustomCellTypes\DerivedCellControlTutorial** sample.

Also,

**C:\Syncfusion\EssentialStudio\VersionNumber\Windows\Grid.Windows\Samples\[Version Number]\CustomCellTypes\DropDownFormAndUserControlSample** sample illustrates how to add your own cell controls. It has two derived cell controls, one drops a modal form when a cell button is pressed, and the other is displays a pop up UserControl when a cell button is pressed.

Among the samples shipped with Grid control, there are several that provide custom cell types. The following table lists some of the samples.

Sample	Description
ExcelSelectionMarker	Has a derived <b>celltype</b> that displays in a cell a BMP that is stored in the <b>GridStyleInfo.Tag</b> for that cell.
GridDataBoundImageCell	Has two cell-derived cell types. Both display picture data stored in a database in a grid cell. One displays the picture directly in the grid cell with several choices regarding how the image is fit to the cell. The second celltype is a drop-down cell that displays a picture box when you click the cell.
VirtTreeGrid	Has an expandable tree node cell with an indentation property. This celltype is used in this example to make the grid have collapsible rows, and generally functions as a multicolumn tree control.
3. LinkLabelCells	4. Has a <b>LinkLabel</b> -derived cell type. This cell type allows you to place a link in a <b>GridCell</b> , and then launch the link in a browser window by clicking the link.
CellButtons	Shows two different types of derived cell controls with buttons. The first is an ellipsis cell that displays further information when you click the button. This particular cell uses a bitmap button to display the ellipsis. The second custom cell type in this sample is a bar of three buttons that display different drag effects as you mouse down and drag.
Drop-down grid	Shows how you can drop a new grid in a cell. This sample is useful as it shows how to pass keystrokes onto the control that is embedded in the grid. For example, you may want the dropped grid to handle the ARROW keys, but not have the ARROW keys move to another cell in the parent grid.
CalendarCells	Shows how you can drop a Windows Forms <b>DateTimePicker</b> in a cell.
SliderCells	Shows how you can drop a Windows Forms

	Slider Control in a cell.
RepeaterUserControl	Shows how you can use a derived cell control in a grid to create a repeater control used to edit DataTables.

**See Also****4.1.4.3.1 GridCellModelBase**

The main function of the **GridCellModelBase** derived class is to serialize your control. It can hold any persistent state independent data that your control uses. The state dependent data should be part of the **GridStyleInfo** object for the cell, but any other persistent property (for example, the initial height of a drop-down) will be in this GridCellModelBase derived class.

The other major function of this class is to create a **cellrenderer** object of the correct type. In fact, the only required override in a GridCellModelBase derived class is the **CreateRenderer** method. In that override, you can create and return a **GridCellRendererBase** derived object that would handle the user interactions in your new cell type.

In general, you probably will not be able to derive directly from the base class, but instead from an existing Essential Grid derived class such as the **GridStaticCellModel**. The following code example illustrates this.

**[C#]**

```
// Define a custom Cell Model by inheriting GridStaticCellModel.
public class LinkLabelCellModel : GridStaticCellModel
{
    protected LinkLabelCellModel(SerializationInfo info,
        StreamingContext context): base(info, context)
    {
    }

    public LinkLabelCellModel(GridModel grid): base(grid)
    {
        AllowFloating = false;
    }

    // Override CreateRenderer Method to create a CellRenderer object.
    public override GridCellRendererBase CreateRenderer(GridControlBase
```

```
control)
{
    return new LinkLabelCellRenderer(control, this);
}
```

**[VB.NET]**

```
' Define a custom Cell Model by inheriting GridStaticCellModel.
Public Class LinkLabelCellModel
    Inherits GridStaticCellModel
    Protected Sub New(ByVal info As SerializationInfo, ByVal context As
StreamingContext)
        MyBase.New(info, context)

    End Sub

    Public Sub New(ByVal grid As GridModel)
        MyBase.New(grid)
        AllowFloating = False

    End Sub

    ' Override CreateRenderer Method to create a CellRenderer object.
    Public Overloads Overrides Function CreateRenderer(ByVal control As
GridControlBase) As GridCellRendererBase
        Return New LinkLabelCellRenderer(Control, Me)
    End Function
End Class
```

#### 4.1.4.3.2 GridCellRendererBase

The class derived from the **GridCellRendererBase** handles the drawing of the cell and the user interaction aspect of the cell architecture. It takes care of things like the handling of the mouse and keyboard messages. Some of the virtual members you might override are listed in the following table.

Virtual Method	Description
OnInitialize(int rowIndex, int collIndex)	Override this method if you need to do any initialization for the current cell. One primary use of this method is to move state information from the <b>GridStyleInfo</b> object to the cell control

	that handles active editing of the cell. For example, if your custom control is a checked list box control, you will use this method to check the selected items in the embedded checked list box from information stored in the cell's style object.
OnDraw(Graphics g, Rectangle clientRectangle, int rowIndex, int colIndex, GridStyleInfo style)	Called to draw the contents of the client bounds for the cell, for example, the text for a static cell. If your cell has an active state, then normally there are two paths through <b>OnDraw</b> . One path draws the active cell and the other handles the drawing of the cell when it is not active. Override it when you want to draw the content of your cell.
OnKeyDown(KeyEventArgs e)	Called when the user presses a key.
OnKeyUp(KeyEventArgs e)	Called when the user releases a key.
OnKeyPress(KeyPressEventEventArgs e)	Called for a user key press.
OnHitTest(int rowIndex, int colIndex, MouseEventArgs mouseEventArgs, IMouseController controller)	Override the <b>OnHitTest</b> in your derived cell renderer if you want to catch mouse events. If you want your renderer class to handle mouse actions like <b>OnMouseDown</b> or <b>OnMouseMove</b> , this OnHitTest should return a non-zero value. Otherwise, your mouse methods will not be called.
OnMouseHoverLeave(int rowIndex, int colIndex, EventArgs e)	Called when your cell renderer has indicated in its OnHitTest override that it wants to receive mouse events, and that the user is moving the mouse out of the cell.
OnMouseHover(int rowIndex, int colIndex, MouseEventArgs e)	Called when your cell renderer has indicated in its OnHitTest override that it wants to receive mouse events, and that the user is moving the mouse over the cell.
OnMouseHoverEnter(int rowIndex, int colIndex)	Called when your cell renderer has indicated in its OnHitTest override that it wants to receive mouse events, and that the user has moved the mouse into the cell.

OnMouseDown(int rowIndex, int colIndex, MouseEventArgs e)	Called when your cell renderer has indicated in its OnHitTest override that it wants to receive mouse events, and that the user has pressed a mouse button.
---	---

These are only a few of the many virtual methods available to you. For a complete list, take a look at the Essential Grid Class Reference.

In general, you probably will not derive directly from the base class, but instead from an existing Essential Grid derived class such as **GridStaticCellModel**.

For a sample implementation of a derived cell control that is based on the existing Static cell control, take a look at the sample in the following path:

**C:\Syncfusion\EssentialStudio\VersionNumber\Windows\Grid.Windows\Samples\[Version Number]\CustomCellTypes\DropDownFormAndUserControlSample**. It shows two implementations of drop-downs. One drops a modal form by deriving **GridStaticCellModel** and **GridStaticCellRenderer**. The other drops a **UserControl** using popup architecture that handles focus issues by deriving **GridDropDownCellModel** and **GridDropDownCellRenderer**.

#### 4.1.4.4 Custom Cell Types

Following are the custom cell types discussed under this section:

##### 4.1.4.4.1 Button Edit

The Button Edit cell type will allow you to add images to the button which can be embedded into the grid cells. This Button Edit cell type can be added by registering its cell model to the corresponding grid by using the **RegisterCellModel** class. There are some in-built images which will be added to the button by providing the button type, and also custom images can be added by specifying the button type as image and providing the location of the image. These Button Edit types can be used by initializing the **ButtonEditStyleProperties** class for the corresponding cell. The Button Edit types provided by grid control are listed as follows.

- Browse
- Check
- Down
- Left
- Leftend
- Redo
- Right

- Rightend
- Undo
- Up
- Image

You can also add custom buttons that you have created to the grid cells. This enables you to add custom buttons like split button to the grid.

The following code examples illustrate how to set the cell type to ButtonEdit.

### 1. Using C#

**[C#]**

```
RegisterCellModel.GridCellType(gridControl1,
CustomCellTypes.ButtonEdit);
ButtonEditStyleProperties sp;
this.gridControl1[rowIndex, colIndex].CellType =
CustomCellTypes.ButtonEdit.ToString();
sp = new ButtonEditStyleProperties(this.gridControl1[rowIndex,
colIndex]);
sp.ButtonEditInfo.ButtonEditType = ButtonType.Browse;
sp.ButtonEditInfo.Width = 50;
```

### 2. Using VB.NET

**[VB .NET]**

```
RegisterCellModel.GridCellType(gridControl1,
CustomCellTypes.ButtonEdit)
Private sp As ButtonEditStyleProperties
Me.gridControl1[rowIndex += 1, colIndex].CellType = "ButtonEdit";
Me.gridControl1(rowIndex, colIndex).CellType =
CustomCellTypes.ButtonEdit.ToString()
sp = New ButtonEditStyleProperties(Me.gridControl1(rowIndex, colIndex))
sp.ButtonEditInfo.ButtonEditType = ButtonType.Browse
```

	A	B	C	D	E
1	<b>ButtonEdit displaying common images</b>				
2		...	✓	▼	◀
3					▶
4	<b>Option to show the button to the Left side</b>				
5	...	✓			
6					
7	<b>Setting text, textcolor and text alignment for the</b>				
8		Sync	Sync	Sync	Sync
9					
10	<b>Setting the width of button</b>				
11		...			
12					
13	<b>Setting the BackColor (In a themed application,</b>				
14		Green	Red	Blue	

Figure 107: Button Edit Cells

#### 4.1.4.4.2 Embedding OLE Objects in the Grid Cell

OLE objects can be directly embedded to a grid's cell which by default displays the icon of the file attached and opens through its default associated application when the cell is activated.

This custom cell type will host the cell as an OLE container. The address of the file should be passed through the cell's `Style.Description` value.

#### Use Case Scenarios

In a payroll application, the generated report can be attached to the grid and viewed directly from the grid. The grid should be exported to the Excel when the operation is completed to view the recent results; otherwise the document will not reflect the recent changes.

#### Sample Link

Find a sample in the following location:

<Install Location>\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Windows\Samples\2.0\ Grid Controls / Grid Control / Concepts and Features/Custom Cell Types

#### Adding OleContainer Cell to an Application

The following code illustrates how to set the cell type to **OleContainer**:

```
[C#]
RegisterCellModel.GridCellType(this.gridControl1,
CustomCellTypes.OleContainerCell);
this.gridControl1[rowIndex, colIndex].CellType = CustomCellTypes.
```

```
OleContainerCell.ToString();
this.gridControl1[rowIndex, colIndex].Description =
GetIconFile(@"common\Data\DocIO\SalesInvoiceDemo.doc");
```

```
[VB.NET]
RegisterCellModel.GridCellType(this.gridControl1,
CustomCellTypes.OleContainerCell);
Me.gridControl1(rowIndex, colIndex).CellType = CustomCellTypes.
OleContainerCell.ToString()
Me.gridControl1(rowIndex, colIndex).Description =
GetIconFile("common\Data\DocIO\SalesInvoiceDemo.doc")
```

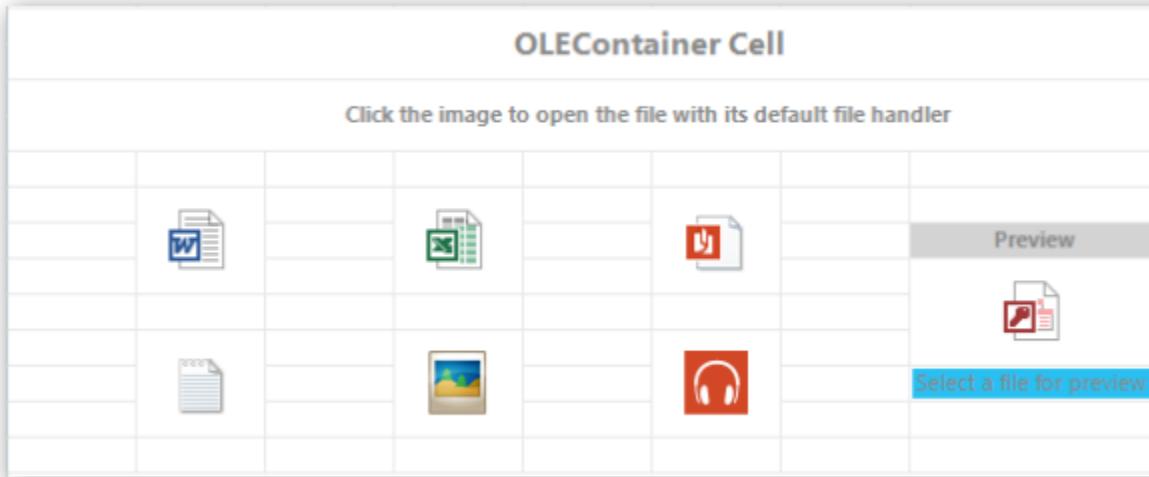


Figure 108 Grid OLE Container Cell

#### 4.1.4.4.3 Calculator Text Box

The Calculator Text Box cell type is implemented as a drop-down container, embedded in the cell where the drop-down contains the calculator which displays the value in the cell.

The following code examples illustrate how to set the cell type to CalculatorTextBox.

##### 1. Using C#

```
[C#]
```

```
RegisterCellModel.GridCellType(gridControl1,
CustomCellTypes.CalculatorTextBox);
CalculatorControl c2 = new CalculatorControl();
c2.BorderStyle = Border3DStyle.RaisedOuter;
c2.BackColor = Color.BlanchedAlmond;
style = gridControl1[6, 2];
style.CellType = CustomCellTypes.CalculatorTextBox.ToString();
style.Control = c2;
```

## 2. Using VB.NET

### [VB .NET]

```
RegisterCellModel.GridCellType(Me.gridControl1,
CustomCellTypes.CalculatorTextBox)
Dim c2 As CalculatorControl = New CalculatorControl()
c2.BorderStyle = Border3DStyle.RaisedOuter
c2.BackColor = Color.BlanchedAlmond
style = gridControl1(6, 2)
style.CellType = CustomCellTypes.CalculatorTextBox.ToString()
style.Control = c2
```

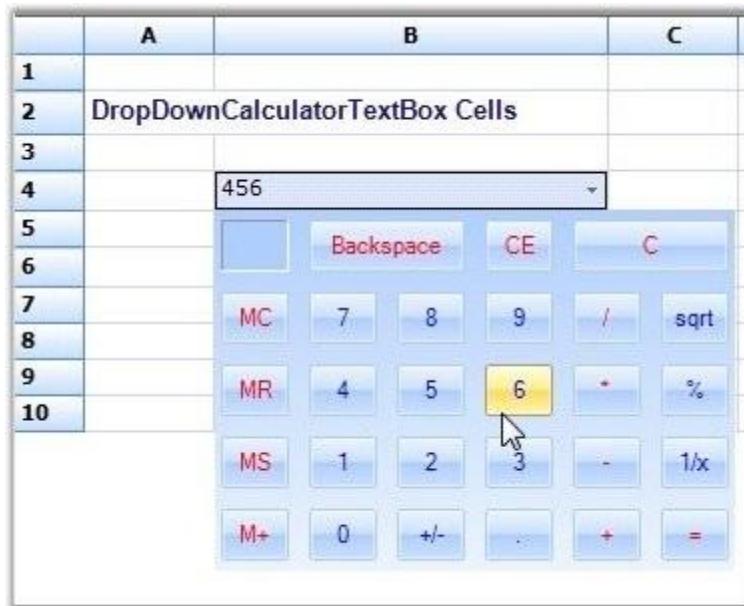


Figure 109: Calculator Text Box Cell

#### 4.1.4.4.4 Calendar

The Calendar cell type can be added by registering the cell model by using the **RegisterCellModel** class.

The following code examples illustrate how to set the cell type to Calendar.

##### 1. Using C#

[C#]

```
GridStyleInfo style;
style = gridControl1[row, 2];
style.CellType = CustomCellTypes.Calendar.ToString();
style.Control = new MonthCalendar();
```

##### 2. Using VB.NET

[VB .NET]

```
Dim style As GridStyleInfo
style = gridControl1(row, 2)
style.CellType = CustomCellTypes.Calendar.ToString()
style.Control = New MonthCalendar()
```

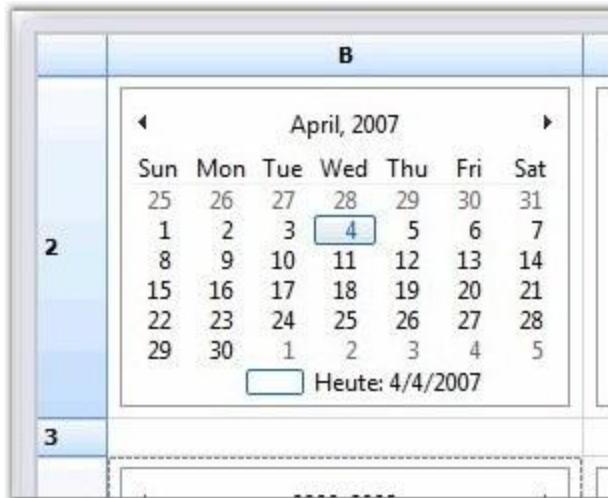


Figure 110: Calendar Cell

#### 4.1.4.4.5 Date Time Picker

The Date Time Picker cell type can be embedded into a cell as a drop-down container, where the date and time picker will be added in the drop-down. The cell value of the corresponding cell has to be specified as date value. Various formats of the date and time can be provided in the **Format** style property.

The following code examples illustrate how to set the cell type to DateTimePicker.

##### 1. Using C#

[C#]

```
RegisterCellModel.GridCellType(gridControl1,
CustomCellTypes.DateTimePicker);
this.gridControl1[4, 2].CellType =
CustomCellTypes.DateTimePicker.ToString();
this.gridControl1[4, 2].CellValueType = typeof(DateTime);
this.gridControl1[4, 2].CellValue = DateTime.Now;
this.gridControl1[4, 2].Format = "MM/dd/yyyy hh:mm";
```

##### 2. Using VB.NET

[VB .NET]

```
RegisterCellModel.GridCellType(Me.gridControl1,
CustomCellTypes.DateTimePicker)
Me.gridControl1(4, 2).CellType =
CustomCellTypes.DateTimePicker.ToString()
Me.gridControl1(4, 2).CellValueType = GetType(DateTime)
Me.gridControl1(4, 2).CellValue = DateTime.Now
Me.gridControl1(4, 2).Format = "MM/dd/yyyy hh:mm"
```

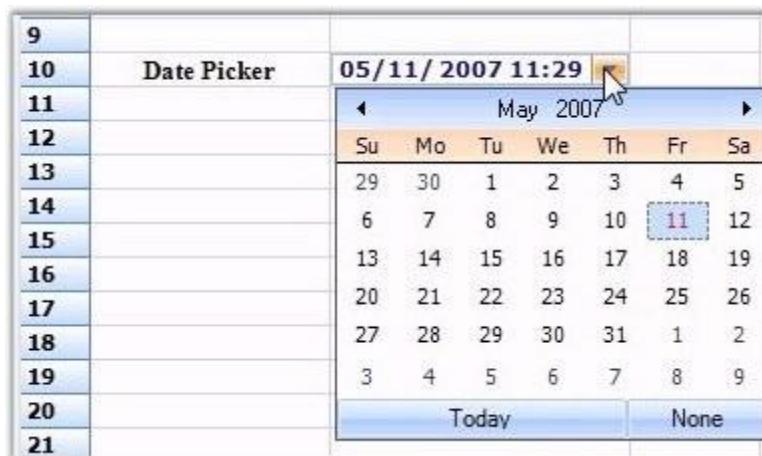


Figure 111: Date Time Picker Cell

#### 4.1.4.4.6 Enhanced Numeric Up Down

The Numeric Up Down cell type has been enhanced to provide more styles and properties that can be added to the numeric up down control by using the **FloatNumericUpDownStyleProperties** class. It enables you to set the limitations of the numeric values and several other properties.

Float Numeric Up Down Style Property	Description
BackColor	Specifies the back color of the container.
Maximum	Indicates the maximum value that the cell can have.
StartValue	The starting value of the embedded cell.
Step	The value that has to be incremented for each click of the button.
WrapValue	The bool value that will allow to wrap the text.
DecimalPlaces	The decimal values after the decimal point.
Orientation	Orientation of the cell container on NumericUpDown.
InterceptArrowkeys	Allows to change the value by using ARROW keys from keyboard.
ThousandsSeparator	The bool value which allows to separate the thousand basis.

The following code examples illustrate how to set the cell type to FNumericUpDown.

##### 1. Using C#

[C#]

```
RegisterCellModel.GridCellType(this.gridControl1,
CustomCellTypes.FNumericUpDown);
GridStyleInfo style = this.gridControl1[1, 1];
```

```
style.Text = "Allow Decimal Increment and Decrement(step by  
0.2,0.01,0.001)";  
style.TextColor = Color.MidnightBlue;  
  
// Set up FNumericUpDown Cell.  
style = gridControl1[2, 2];  
style.CellType = CustomCellTypes.FNumericUpDown.ToString();  
style.Text = "0.5";  
  
// Assign the Style Properties of Up Down Control.  
FloatNumericUpDownStyleProperties sp = new  
FloatNumericUpDownStyleProperties(style);  
sp.StyleInfo.BackColor = SystemColors.Window;  
sp.FloatNumericUpDownProperties.Maximum = 15.0;  
sp.FloatNumericUpDownProperties.Minimum = 0.0;  
sp.FloatNumericUpDownProperties.StartValue = 0.5;  
sp.FloatNumericUpDownProperties.Step = 0.2;  
sp.FloatNumericUpDownProperties.WrapValue = true;  
sp.FloatNumericUpDownProperties.DecimalPlaces = 1;
```

## 2. Using VB.NET

### [VB.NET]

```
RegisterCellModel.GridCellType(gridControl1,  
CustomCellTypes.FNumericUpDown)  
Dim style As GridStyleInfo = Me.gridControl1(1, 1)  
style.Text = "Allow Decimal Increment and Decrement(step by  
0.2,0.01,0.001)"  
style.TextColor = Color.MidnightBlue  
  
' Set up FNumericUpDown Cell.  
style = gridControl1(2, 2)  
style.CellType = CustomCellTypes.FNumericUpDown.ToString()  
style.Text = "0.5"  
  
' Assign the Style Properties of Up Down Control.  
Dim sp As FloatNumericUpDownStyleProperties = New  
FloatNumericUpDownStyleProperties(style)  
sp.StyleInfo.BackColor = SystemColors.Window  
sp.FloatNumericUpDownProperties.Maximum = 15.0  
sp.FloatNumericUpDownProperties.Minimum = 0.0  
sp.FloatNumericUpDownProperties.StartValue = 0.5  
sp.FloatNumericUpDownProperties.Step = 0.2  
sp.FloatNumericUpDownProperties.WrapValue = True  
sp.FloatNumericUpDownProperties.DecimalPlaces = 1
```

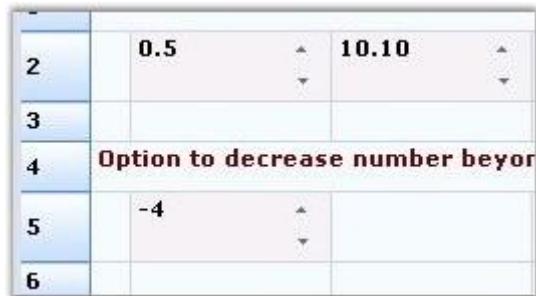


Figure 112: Enhanced Numeric Up Down Cells

#### 4.1.4.4.7 GridInCell

The GridInCell cell type provide a covered range of cells to embed the grid, which is added as a control to the cells. The registered cell model will initialize the range by calculating the size of the grid control to be embedded, and add some style such as borders and scroll bar to have the control within the range.

The following code examples illustrate how to set the cell type to GridinCell.

##### 1. Using C#

```
[C#]

RegisterCellModel.GridCellType(gridControl1,
CustomCellTypes.GridinCell);
gridControl1.BackColor = Color.FromArgb(0xda, 0xe5, 0xf5);
GridControl grid;

this.gridControl1[3, 2].CellType =
CustomCellTypes.GridinCell.ToString();
this.gridControl1.CoveredRanges.Add(GridRangeInfo.Cells(3, 2, 7, 4));
grid = new CellEmbeddedGrid(this.gridControl1);
grid.BackColor = Color.FromArgb(0xb4, 0xe7, 0xf2);
grid.RowCount = 10;
grid.ColCount = 4;
grid[1, 1].Text = "this is a 10x4 grid";
grid.ThemesEnabled = true;
this.gridControl1[3, 2].Control = grid;
this.gridControl1.Controls.Add(grid);
```

## 2. Using VB.NET

### [VB.NET]

```
RegisterCellModel.GridCellType(Me.gridControl1,  
CustomCellTypes.GridInCell)  
Dim grid As GridControl  
  
Me.gridControl1(3, 2).CellType = CustomCellTypes.GridInCell.ToString()  
Me.gridControl1.CoveredRanges.Add(GridRangeInfo.Cells(3, 2, 7, 4))  
grid = New CellEmbeddedGrid(Me.gridControl1)  
grid.BackColor = Color.FromArgb(&HB4, &HE7, &HF2)  
grid.RowCount = 10  
grid.ColCount = 4  
grid(1, 1).Text = "this is a 10x4 grid"  
grid.ThemesEnabled = True  
Me.gridControl1(3, 2).Control = grid  
Me.gridControl1.Controls.Add(grid)
```

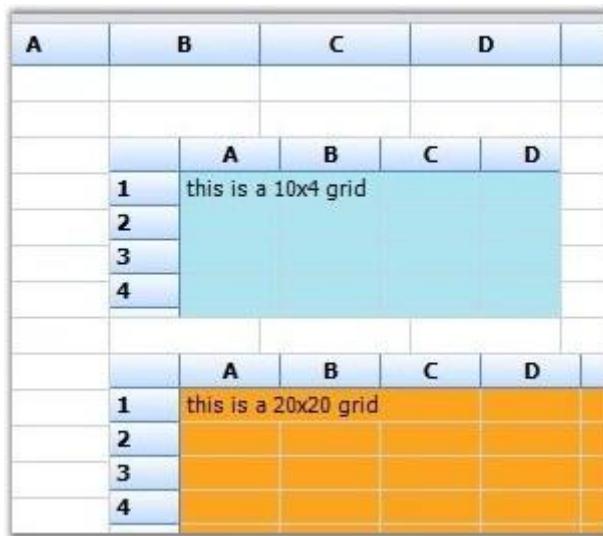


Figure 113: "GridInCell" Cells

#### 4.1.4.4.8 Link Label Cell

The Link Label Cell cell type holds the link that has been provided in the **Tag** property. This displays ordinary text in the cell which links to the specified location.

The following code examples illustrate how to set the cell type to LinkLabelCell.

1. Using C#

[C#]

```
RegisterCellModel.GridCellType(gridControl1,
CustomCellTypes.LinkLabelCell);
int rowIndex = 5;
gridControl1[rowIndex, 2].CellType =
CustomCellTypes.LinkLabelCell.ToString();
gridControl1[rowIndex, 2].Text = "Syncfusion, Inc.";
gridControl1[rowIndex, 2].Font.Bold = true;
gridControl1[rowIndex, 2].Tag = "http://www.syncfusion.com";
```

2. Using VB.NET

[VB .NET]

```
RegisterCellModel.GridCellType(gridControl1,
CustomCellTypes.LinkLabelCell)
Dim rowIndex As Integer = 5
gridControl1(rowIndex, 2).CellType =
CustomCellTypes.LinkLabelCell.ToString()
gridControl1(rowIndex, 2).Text = "Syncfusion, Inc."
gridControl1(rowIndex, 2).Font.Bold = True
gridControl1(rowIndex, 2).Tag = "http://www.syncfusion.com"
```

4		
5		Syncfusion, Inc.
6		<a href="#">Windows Forms FAQ</a>
7		<a href="#">Microsoft Windows Forms</a>
8		<a href="#">MSDN</a>
9		<a href="#">Yahoo</a>
10		<a href="#">Google</a>

Figure 114: "Link Label Cell" Cells

#### 4.1.4.4.9 Picture Box

The Picture Box cell type can be embedded into a cell by calculating the size of the picture, and extending the width and height of the cell accordingly. The **PictureBoxStyleProperties** class provides the style where it holds the information of the picture that has to be added.

The following code examples illustrate how to set the cell type to PictureBox.

1. Using C#

[C#]

```
RegisterCellModel.GridCellType(gridControl1,
CustomCellTypes.PictureBox);
PictureBoxStyleProperties sp;
style = gridControl1[2, 2];
style.CellType = CustomCellTypes.PictureBox.ToString();
sp = new PictureBoxStyleProperties(style);
sp.Image = GetImage("one.jpg");
```

2. Using VB.NET

[VB .NET]

```
RegisterCellModel.GridCellType(Me.gridControl1,
CustomCellTypes.PictureBox)
Dim sp As PictureBoxStyleProperties
style = gridControl1(2, 2)
style.CellType = CustomCellTypes.PictureBox.ToString()
sp = New PictureBoxStyleProperties(style)
sp.Image = GetImage("one.jpg")
```

	A	B	C
1			
2			
3			

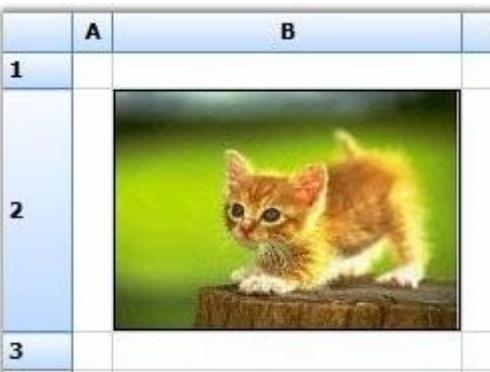


Figure 115: Picture Box Cell

#### 4.1.4.4.10 Slider

A Slider control embedded in a grid cell is termed as a Slider Cell. Slider control can be embedded in the grid cells by using Slider cell type. The class **SliderStyleProperties** provides custom properties specific to the Slider control. All the properties support the style inheritance mechanism.

The Slider control can be embedded by using the following set of codes:

1. Using C#

[C#]

```
gridControl1.CellModels.Add("Slider", new
SliderCellModel(gridControl1.Model));
GridStyleInfo style;
style = gridControl1[4, 5];
SliderStyleProperties sp = new SliderStyleProperties(style);
style.CellType = "Slider";
sp.Maximum = 40;
sp.Minimum = 0;
sp.TickFrequency = 8;
sp.LargeChange = 16;
sp.SmallChange = 4;
sp.Orientation = Orientation.Vertical;
```

2. Using VB.NET

[VB .NET]

```
gridControl1.CellModels.Add("Slider", New
SliderCellModel(gridControl1.Model))
Dim style As GridStyleInfo
style = gridControl1(4, 5)
Dim sp As SliderStyleProperties = New SliderStyleProperties(style)
style.CellType = "Slider"
sp.Maximum = 40
sp.Minimum = 0
sp.TickFrequency = 8
sp.LargeChange = 16
sp.SmallChange = 4
sp.Orientation = Orientation.Vertical
```

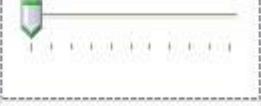
	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					
8					

Figure 116: Slider Cell

#### 4.1.4.4.11 XHTML Cell

An XHTML page can be displayed in a grid cell by using Xhtml Cell cell type. A custom cell type can be created and registered to provide XHTML functionality. It requires derivation of two classes:

- GridCellModelBase
- GridCellRendererBase

The **CellModel** (read GridCellModelBase class in this document) handles any serialization that a cell type requires, and also creates the CellRenderer (read GridCellRendererBase class in this document) class that is associated with the cell type. The **CellRenderer** class manages the UI aspects of the cell type.

The XHTML page can be displayed by using the following set of codes.

##### 1. Using C#

[C#]

```
string xhtml1 = "<body style=\"font-family:Arial; line-height:1em\"> ";
xhtml1 += "<h1 style=\"text-align:center; color:#EE7A03
\">XhtmlCells</h1>";
```

```
xhtml1 += "<p>";  
xhtml1 += "<p>XhtmlCells use the RichTextBoxSupportsXHTML control from  
GotDotNet user samples to display XHTML formatted text inside a  
cell.</p>";  
xhtml1 += "</body>";  
gridControl1[rowIndex, 1].CellType = "XhtmlCell";  
gridControl1[rowIndex, 1].Text = xhtml1;
```

## 2. Using VB.NET

### [VB .NET]

```
Dim xhtml1 As String = "<body style=""font-family:Arial; line-  
height:1em""> "  
xhtml1 += "<h1 style=""text-align:center; color:#EE7A03  
"">XhtmlCells</h1>"  
xhtml1 += "<p>"  
xhtml1 += "<p>XhtmlCells use the RichTextBoxSupportsXHTML control from  
GotDotNet user samples to display XHTML formatted text inside a  
cell.</p>"  
xhtml1 += "</body>"  
gridControl1(rowIndex, 1).CellType = "XhtmlCell"  
gridControl1(rowIndex, 1).Text = xhtml1
```

	A	B	C	D	E	F	G
1							
2							
3							
4							
5							
6	XhtmlCells use the RichTextBoxSupportsXHTML control from						
7	GotDotNet user samples to display XHTML formatted text inside a ce						
8							
9							
10							
11							
12							
13							
14							

*Figure 117: XHTML Cell*

#### 4.1.4.4.12 Chart Cell

Essential Chart control can be embedded in grid cells by creating and registering a custom Chart Cell cell type. The **CellModel** class handles any serialization that a cell type requires, and also creates the CellRenderer class associated with the cell type.

The actions mentioned can be performed by using the following code example.

##### 1. Using C#

[C#]

```
ChartStyleProperties csp;
this.gridControl1.CellModels.Add("ChartCell", new
GridChartCellModel(this.gridControl1.Model));
style = this.gridControl1[8, 2];
style.CellType = "ChartCell";
csp = new ChartStyleProperties(style);
csp.ChartType = ChartSeriesType.Column;
csp.TitleText = "Chart Cell";
csp.Series3D = false;
csp.TitleAlignment = StringAlignment.Center;
```

##### 2. Using VB.NET

[VB .NET]

```
Dim csp As ChartStyleProperties
Me.gridControl1.CellModels.Add("ChartCell", New
GridChartCellModel(Me.gridControl1.Model))
style = Me.gridControl1(8, 2)
style.CellType = "ChartCell"
csp = New ChartStyleProperties(style)
csp.ChartType = ChartSeriesType.Column
csp.TitleText = "Chart Cell"
csp.Series3D = False
csp.TitleAlignment = StringAlignment.Center
```

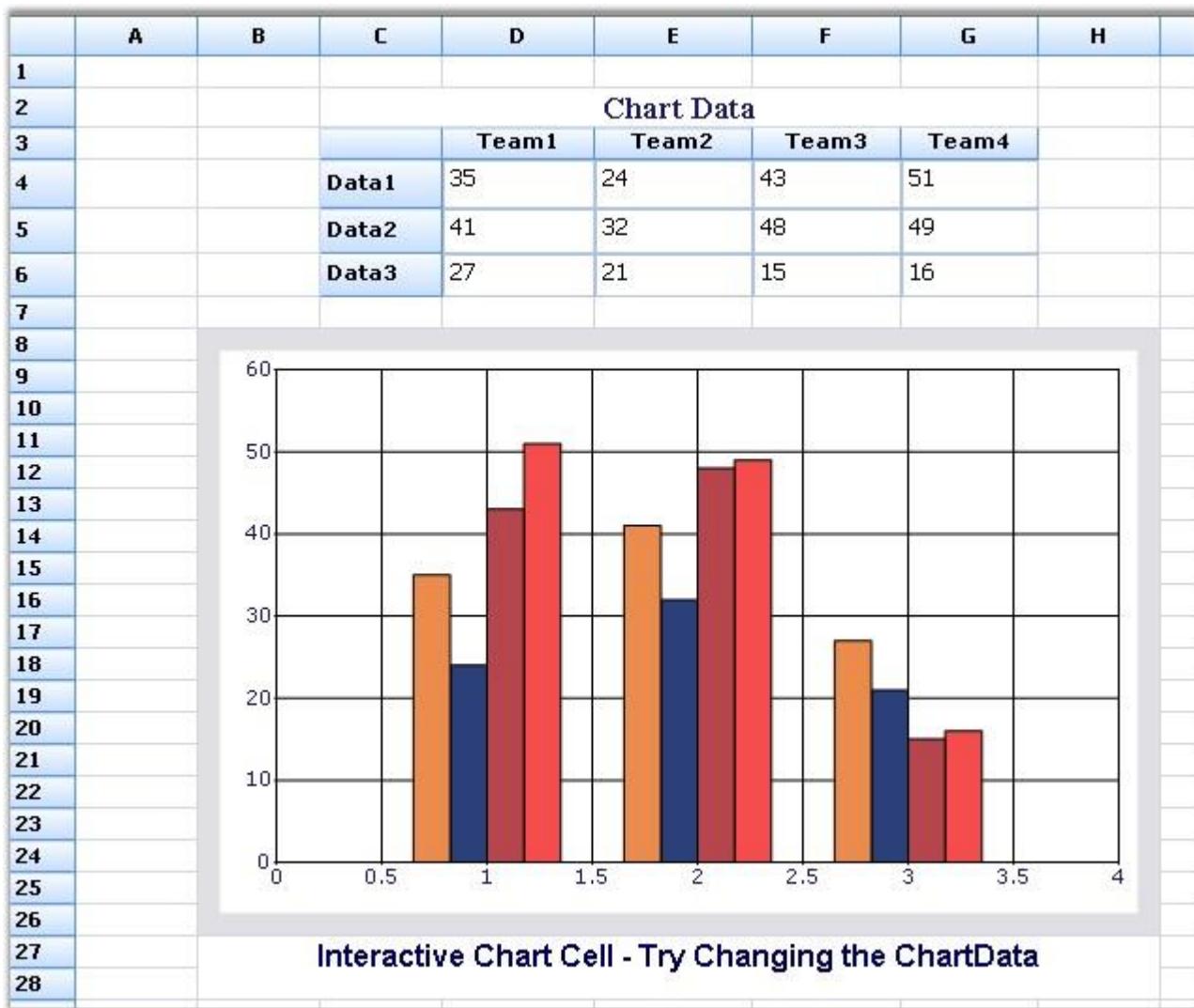


Figure 118: Chart Cell

#### 4.1.4.4.13 Drop-Down Grid Cell

Essential Grid has flexible support for displaying drop-down grids in cells. It uses a custom cell derived from the **GridDropDownGridCellModel/GridDropDownGridCellRenderer** classes to display a drop-down grid. The GridDropDownGridCellModel gets an instance of the GridControlBase, and displays it through the GridDropDownGridCellRenderer.

The actions mentioned can be performed by using the following code examples.

1. Using C#

[C#]

```
// Create and register drop-down grid cells.  
DropDownGridCellModel aModel = new  
DropDownGridCellModel(this.gridControl1.Model);  
aModel.EmbeddedGrid = GridA;  
gridControl1.CellModels.Add("GridADropCell", aModel);  
  
// Set the drop-downs in the cell.  
this.gridControl1[rowIndex,1].Text = "Grid A";  
this.gridControl1[rowIndex,1].CellType = "GridADropCell";
```

2. Using VB.NET

[VB.NET]

```
' Create and register drop-down grid cells.  
Dim aModel As DropDownGridCellModel = New  
DropDownGridCellModel(Me.gridControl1.Model)  
aModel.EmbeddedGrid = GridA  
gridControl1.CellModels.Add("GridADropCell", aModel)  
  
' Set the drop-downs in the cell.  
Me.gridControl1(rowIndex,1).Text = "Grid A"  
Me.gridControl1(rowIndex,1).CellType = "GridADropCell"
```

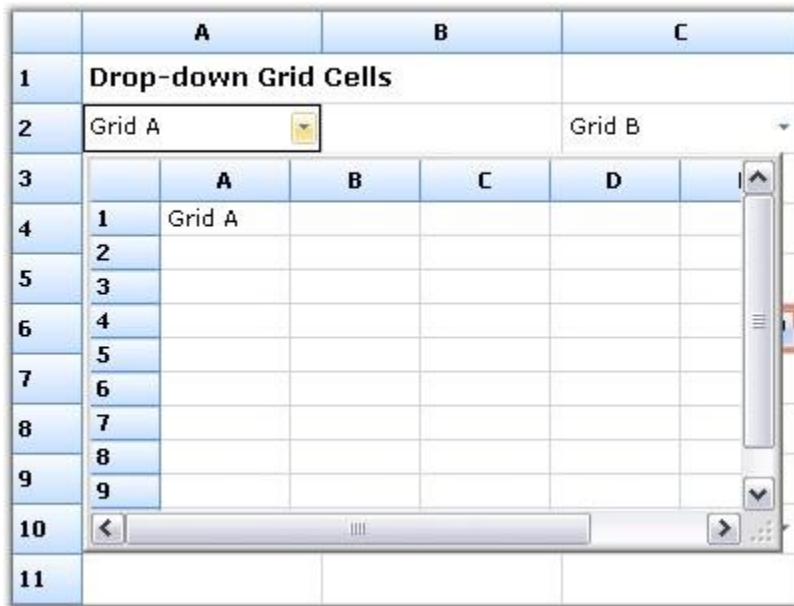


Figure 119: Drop-Down Grid Cell

#### 4.1.4.4.14 Drop-Down Form and User Control Cell

A custom control cell that displays a drop-down form or a user control in a grid cell can be created.

You can create:

- A drop-down form in a grid cell by deriving **GridStaticCellModel/GridStaticCellRenderer** classes.
- A drop-down User control in a grid cell by deriving **GridDropDownCellModel/GridDropDownCellRender** classes.

The actions mentioned can be performed by using the following code example.

##### 1. Using C#

[C#]

```
// Register your custom cell type.  
this.gridControl1.CellModels.Add("DropDownForm", new  
DropDownFormCellModel(this.gridControl1.Model, new DropDownForm()));  
  
// Set the style.CellType for the cells.  
this.gridControl1[2, 2].CellType = "DropDownForm";  
  
// Register your custom cell type.  
this.gridControl1.CellModels.Add("DropDownUserControl", new  
DropDownUserCellModel(this.gridControl1.Model, new DropDownUser()));  
  
// Set the style.CellType for the cells.  
this.gridControl1[6, 2].CellType = "DropDownUserControl";
```

##### 2. Using VB.NET

[VB.NET]

```
' Register your custom cell type.  
Me.gridControl1.CellModels.Add("DropDownForm", New  
DropDownFormCellModel(Me.gridControl1.Model, New DropDownForm()))  
  
' Set the style.CellType for the cells.
```

```

Me.gridControl1(2, 2).CellType = "DropDownForm"

' Register your custom cell type.
Me.gridControl1.CellModels.Add("DropDownUserControl", New
DropDownUserCellModel(Me.gridControl1.Model, New DropDownUser()))

' Set the style.CellType for the cells.
Me.gridControl1(6, 2).CellType = "DropDownUserControl"

```

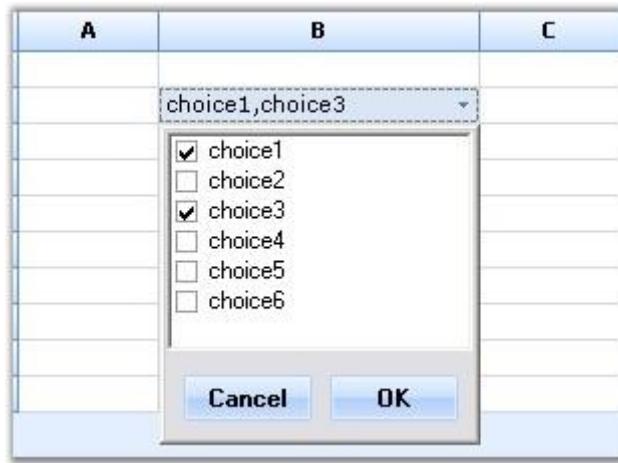


Figure 120: Drop-Down Grid Form

#### 4.1.4.4.15 Integer Text Box

Integer text box is used to display integer data-type values in the grid cells. This cell type can be added to the grid cells by registering its cell model using the **RegisterCellModel** class.

The following code example illustrates how to add the integer text box to grid cells using the **RegisterCellModel** class.

[C#]

```

RegisterCellModel.GridCellType(this.gridControl1,
CustomCellTypes.IntegerTextBox);

this.gridControl1[4, 2].CellType =
CustomCellTypes.IntegerTextBox.ToString();

```

[VB]

```
RegisterCellModel.GridCellType(Me.gridControl1,  
CustomCellTypes.IntegerTextBox)
```

```
Me.gridControl1(4, 2).CellType =  
CustomCellTypes.IntegerTextBox.ToString()
```

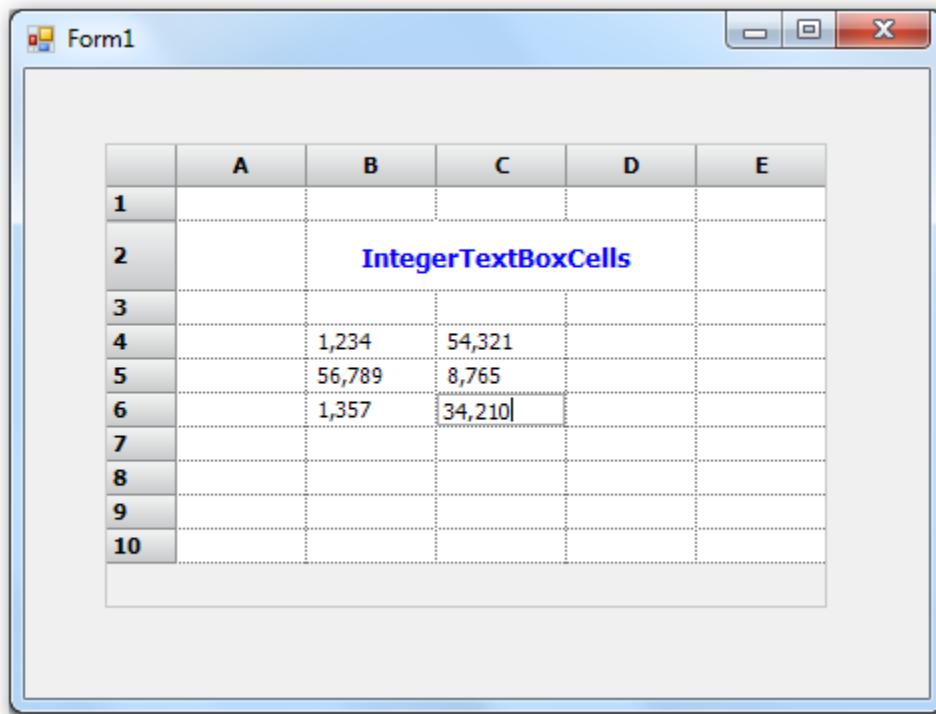


Figure 121: Integer Text Box

#### 4.1.4.4.16 Double Text Box

Double text box is used to display double data-type values in the grid cells. This cell type can be added to the grid cells by registering its cell model using the **RegisterCellModel** class.

The following code example illustrates how to add the double text box to grid cells using the **RegisterCellModel** class.

[C#]

```
RegisterCellModel.GridCellType(this.gridControl1,  
CustomCellTypes.DoubleTextBox);  
  
this.gridControl1[4, 2].CellType=  
CustomCellTypes.DoubleTextBox.ToString();
```

[VB]

```
RegisterCellModel.GridCellType(Me.gridControl1,  
CustomCellTypes.DoubleTextBox)  
  
Me.gridControl1(4, 2).CellType =  
CustomCellTypes.DoubleTextBox.ToString()
```

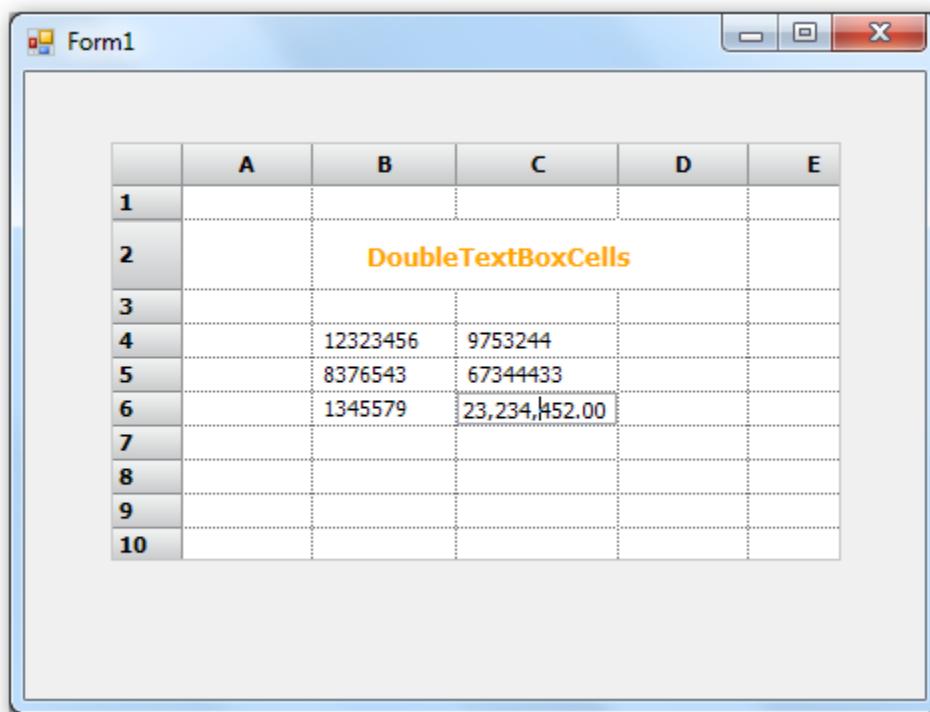


Figure 122: Double Text Box

#### 4.1.4.4.17 Percent Text Box

Percent text box is used to display percentage values in the grid cell. This cell type can be added to the grid cells by registering its cell model using the **RegisterCellModel** class.

The following code example illustrates how to add the percent text box to grid cells using the **RegisterCellModel** class.

[C#]

```
RegisterCellModel.GridCellType(this.gridControl1,
CustomCellTypes.PercentTextBox);

this.gridControl1[5, 2].CellType =
CustomCellTypes.PercentTextBox.ToString();
```

[VB]

```
RegisterCellModel.GridCellType(Me.gridControl1,
CustomCellTypes.PercentTextBox)

Me.gridControl1(4, 2).CellType =
CustomCellTypes.PercentTextBox.ToString()
```

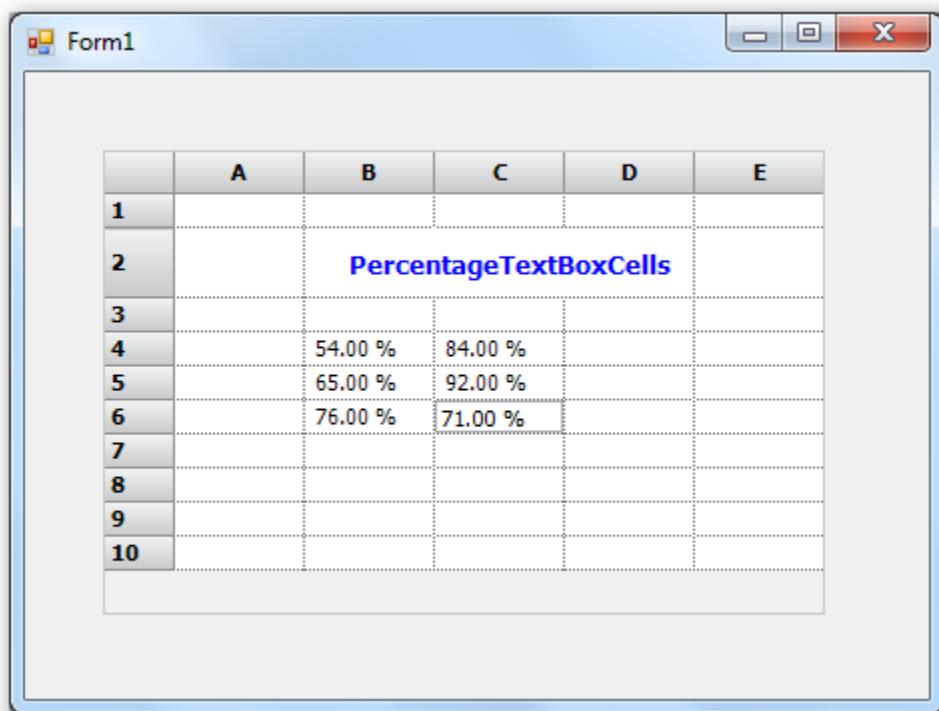


Figure 123: Percent Text Box

#### 4.1.4.5 MS Excel-like Features

Essential Grid offers a wide range of MS Excel-like features. Following is the list of features offered:

#### 4.1.4.5.1 Selection Frame

Essential Grid supports MS Excel-like **Selection Frame** feature. This enables to highlight selected cells in a frame. This feature can be enabled by setting **ExcelLikeSelectionFrame** property to *true*.

Selection Frame feature can be enabled for Essential Grid by using the following code:

## 1. Using C#

[C#]

```
this.gridControl1.ExcelLikeSelectionFrame = true;
```

## 2. Using VB.NET

[VB .NET]

```
Me.gridControl1.ExcelLikeSelectionFrame = True
```

A	B	C	D	E	F

*Figure 124: Selection Frame*

#### 4.1.4.5.2 Current Cell

Essential Grid supports MS-Excel like **Current Cell** feature. This feature can be enabled by setting **ExcelLikeCurrentCell** property to *true*. When the user moves the current cell out of a selected range, the range will be cleared. If the user moves the current cell inside a selected range, the range will stay.

Current Cell feature can be enabled for Essential Grid by using the following code:

## 1. Using C#

[C#]

```
this.gridControl1.ExcelLikeCurrentCell = true;
```

## 2. Using VB.NET

## [VB .NET]

```
Me.gridControl1.ExcelLikeCurrentCell = True
```

A	B	C	D	E

*Figure 125: Current Cell*

#### 4.1.4.5.3 Workbook

The worksheets in a workbook can be displayed as tabs for easier view and selection of worksheets, similar to Excel. This is achieved by using the Tab Bar Splitter control. You can add any number of Tab Bar pages to the Tab Bar Splitter control, and then add the Grid control to each Tab Bar page, to get the appearance similar to the Workbook in Excel.

##### 1. Using C#

[C#]

```
this.tabBarSplitterControl1.Controls.Add(this.tabBarPage1);
this.tabBarSplitterControl1.Controls.Add(this.tabBarPage2);
this.tabBarSplitterControl1.Controls.Add(this.tabBarPage3);

// Adding Grid controls to the Tab Bar Pages.
this.tabBarPage1.Controls.Add(this.gridControl1);
this.tabBarPage2.Controls.Add(this.gridControl2);
this.tabBarPage3.Controls.Add(this.gridControl3);
```

##### 2. Using VB.NET

[VB .NET]

```
Me.tabBarSplitterControl1.Controls.Add(Me.tabBarPage1)
Me.tabBarSplitterControl1.Controls.Add(Me.tabBarPage2)
Me.tabBarSplitterControl1.Controls.Add(Me.tabBarPage3)

' Adding Grid controls to the Tab Bar Pages.
Me.tabBarPage1.Controls.Add(Me.gridControl1)
Me.tabBarPage2.Controls.Add(Me.gridControl2)
Me.tabBarPage3.Controls.Add(Me.gridControl3)
```

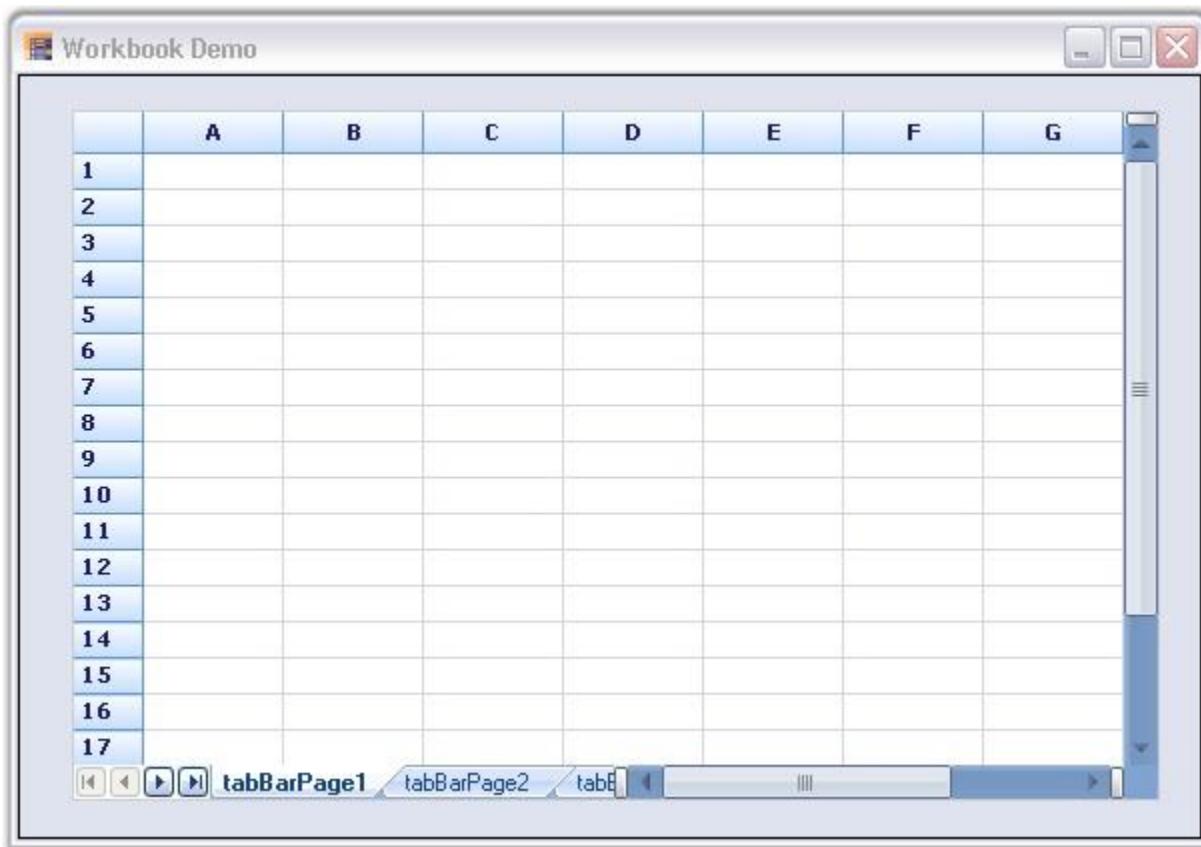


Figure 126: Workbook

#### 4.1.4.5.4 Splitter

A dynamic-splitter window can be embedded in Essential Grid to show multiple views of the same grid by using a Splitter. This MS Excel-like feature enables you to view more than one copy of a worksheet, and scroll through each pane of the worksheet independently. The panes work simultaneously, i.e., the changes made in one pane are reflected in the other. The splitter can be scrolled by placing the mouse pointer over it, holding down the left mouse button and dragging it to the required position. It can be split horizontally and vertically.

Following are the events associated with the Splitter control.

Event	Description
PaneCreated	This event is triggered when the splitter is moved across the Grid.

PaneClosing	This event is triggered either when the splitter is moved to the end/beginning or when it cannot be located on the worksheet.
-------------	---

The splitter can be created in a worksheet by using the following code:

1. Using C#

[C#]

```
this.splitterControl1.Controls.Add(this.gridControl1);

// PaneCreated event.
private void splitterControl1_PaneCreated(object sender,
Syncfusion.Windows.Forms.SplitterPaneEventArgs e)
{
    Console.WriteLine("Created: " + e.ToString());
}

// PaneClosing event.
private void splitterControl1_PaneClosing(object sender,
Syncfusion.Windows.Forms.SplitterPaneEventArgs e)
{
    Console.WriteLine("Closed: " + e.ToString());
}
```

2. Using VB.NET

[VB.NET]

```
Me.splitterControl1.Controls.Add(Me.gridControl1)

' PaneCreated event.
private void splitterControl1_PaneCreated(Object sender,
Syncfusion.Windows.Forms.SplitterPaneEventArgs e)
Console.WriteLine("Created: " & e.ToString())

' PaneClosing event.
private void splitterControl1_PaneClosing(Object sender,
Syncfusion.Windows.Forms.SplitterPaneEventArgs e)
Console.WriteLine("Closed: " & e.ToString())
```

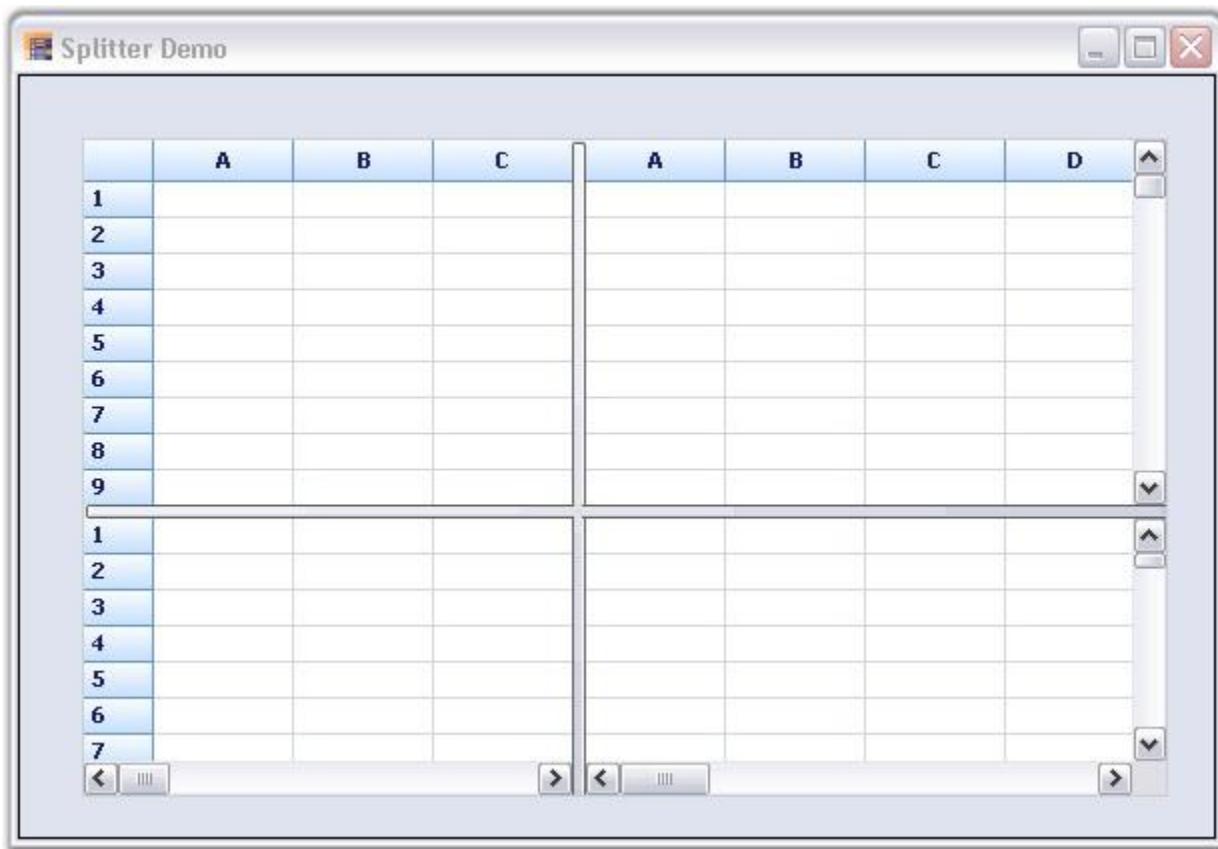


Figure 127: Splitter

#### 4.1.4.5.5 Freeze Pane

Essential Grid supports MS Excel-like **Freeze Pane** feature. In a large worksheet, it is often required that column or row labels remain in view. This feature enables you to freeze either columns or rows in the Grid, so that they remain visible while you scroll. The number of rows to be frozen can be specified by using **Model.Rows.FrozenCount** property and the number of columns to be frozen can be specified by using **Model.Cols.FrozenCount** property.

The Freeze Pane feature can be enabled for Essential Grid by using the following code:

1. Using C#

[C#]

```
this.gridControl1.Model.Rows.FrozenCount = 4;
this.gridControl1.Model.Cols.FrozenCount = 3;
```

## 2. Using VB.NET

[VB .NET]

```
Me.gridControl1.Model.Rows.FrozenCount = 4  
Me.gridControl1.Model.Cols.FrozenCount = 3
```



**Note:** You can unfreeze the frozen rows/columns by clicking **Unfreeze Current Row/Col** button on the UI.

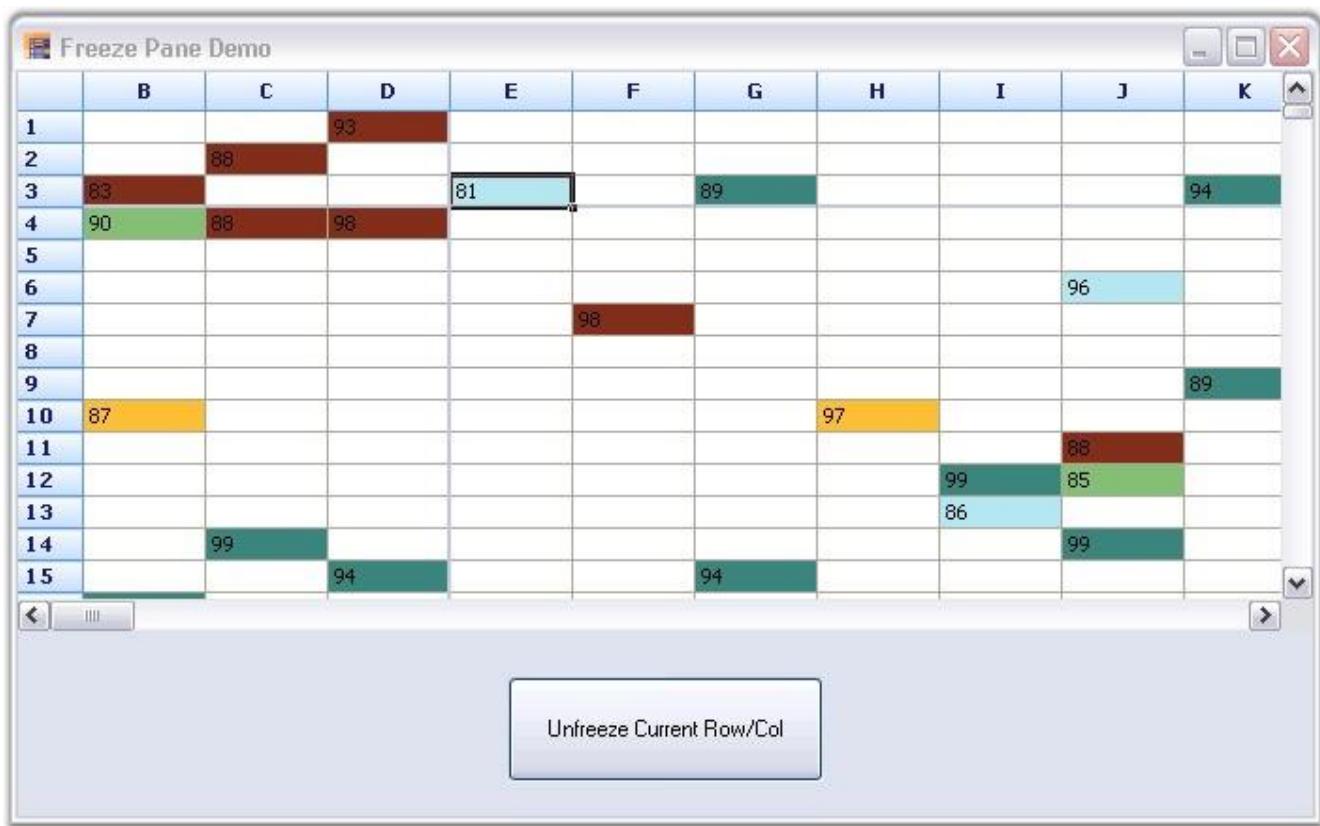


Figure 128: Freeze Pane feature Illustrated

### 4.1.4.5.6 MultiLevel Undo and Redo

Essential Grid has flexible support for **Multilevel Undo/Redo**. This feature enables the user to undo history for most actions that are performed. This feature can be enabled by setting the **CommandStack.Enabled** property to *true*. Using the functions of the **GridModelCommandManager** class, various tasks like undo and redo can be done. You can access this class from a Grid with the **CommandStack** property of a GridModel instance.

The Multilevel Undo/Redo feature can be enabled for Essential Grid by using the following code:

1. Using C#

[C#]

```
this.gridControl1.CommandStack.Enabled = true;
this.gridControl1.CommandStack.Undo();
this.gridControl1.CommandStack.Redo();
```

2. Using VB.NET

[VB .NET]

```
Me.gridControl1.CommandStack.Enabled = True
Me.gridControl1.CommandStack.Undo()
Me.gridControl1.CommandStack.Redo()
```

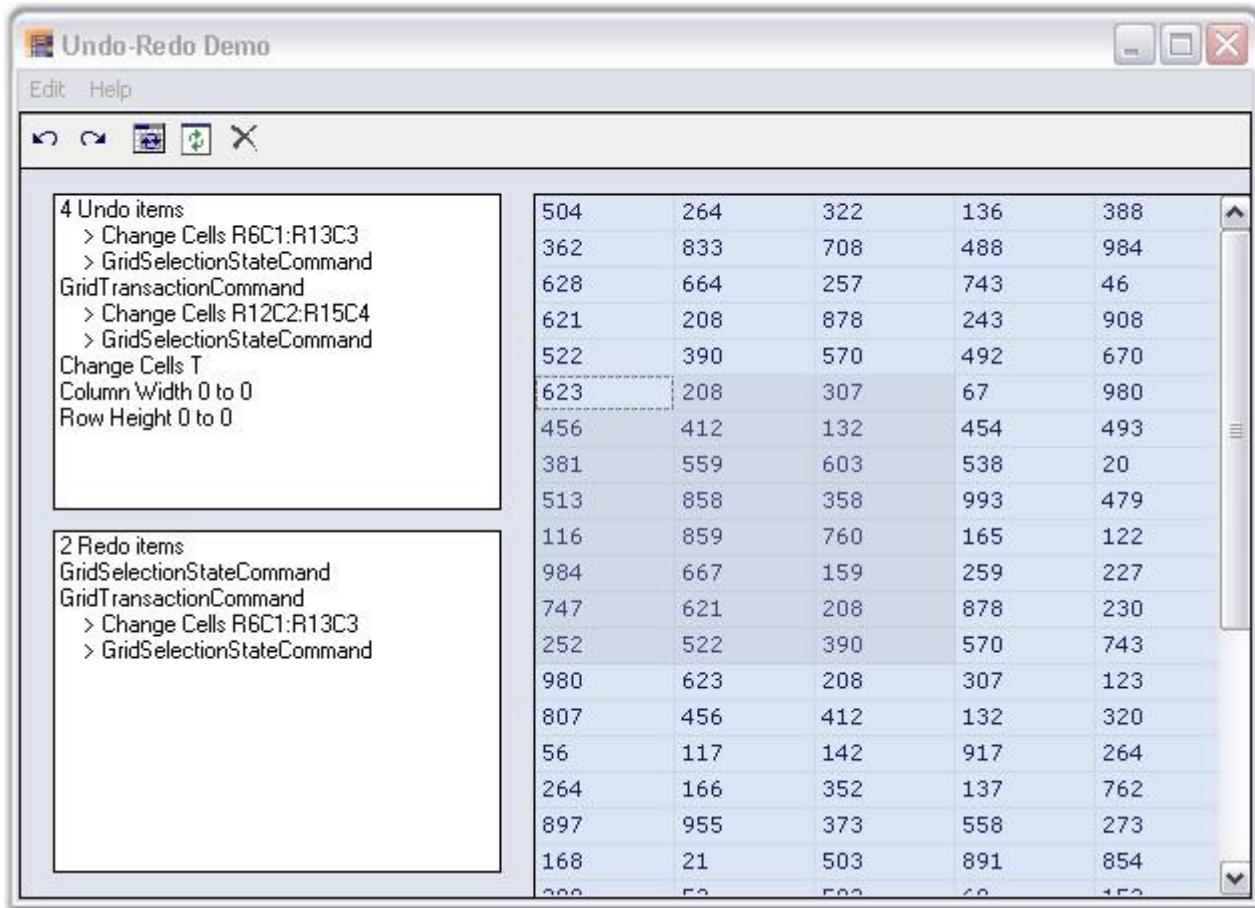


Figure 129: Multilevel Undo/Redo

#### 4.1.4.5.7 Find Replace

**Find and Replace** feature enables you to search and replace the required element present in the Grid/Worksheet. You can implement the fastest Find and Replace functionality with Grid controls by using **GridFindReplaceDialogSink** and **GridFindReplaceEventArgs** classes. The **GridFindReplaceDialogSink** class provides the methods that are necessary to perform a Find and Replace operation. The **GridFindReplaceEventArgs** class provides information about the Find and Replace dialog box.

The value entered in the **Search For** field is highlighted in the worksheet after the search action is performed. You can switch over to each highlighted text by clicking **Find Next** button. This functionality is available only when there is more than one search result.

#### Search and Replace Options

The search and replace actions:

- Can be performed independently or simultaneously.
- Can be done for individual search or for the entire worksheet by using the **Find Next/Replace** buttons.
- Can be done for all the search results by clicking **Find All/Replace All** buttons.

## Search Options

The search options are as follows:

- **Match Case**-Matches case while performing search.
- **Match Whole Cell**-Matches the search text with the entire text in a grid cell.
- **Search Up**-Specifies if the search can be performed bottom-up.
  - **Column Only**-Searches only the current column.
  - **Selection Only**-Searches only the current selection.
  - **Whole Table**-Searches the whole table.

The Find and Replace feature can be enabled for Essential Grid by using the following code:

### 1. Using C#

[C#]

```
GridFindTextOptions options = GridFindTextOptions.WholeTable |  
GridFindTextOptions.SearchUp;  
object locInfo = GridRangeInfo.Table();  
GridFindReplaceEventArgs frEvents = new  
GridFindReplaceEventArgs(cmbSearch.Text, "", options, locInfo);  
GridFindReplaceDialogSink frDialog.Find(frEvents);
```

### 2. Using VB.NET

[VB .NET]

```
Private options As GridFindTextOptions = GridFindTextOptions.WholeTable  
Or GridFindTextOptions.SearchUp  
  
Private locInfo As Object = GridRangeInfo.Table()  
Private frEvents As New GridFindReplaceEventArgs(cmbSearch.Text, "",  
options, locInfo)  
GridFindReplaceDialogSink(frDialog.Find(frEvents))
```



Figure 130: Find and Replace

#### 4.1.4.5.8 Unhide Column by Double-Clicking Disabled

Essential Grid has changed the unhide column operation to emulate the behavior found in Microsoft Excel. Previously, hidden columns could be shown by double-clicking a row. This behavior has been disabled so that applications created using Essential Grid will be similar to the hide/unhide behavior found in Microsoft Excel.

## Properties

Table 2: Property Table

Property	Description	Type	Data Type	Reference links
UnHideColsOnDbClick	Indicates whether to allow unhide the hidden columns when double click the row.	Property	Boolean	N/A.

### Sample Link

A demo of this feature is available in the following location:

**..\\AppData\\Local\\Syncfusion\\EssentialStudio\\Installed  
Version}\\Windows\\Grid.Windows\\Samples\\2.0\\Grid Layout\\Hide Rows and Columns Demo**

### Disabling Unhide Column by Double-Clicking

To disable unhide column by double-clicking, set the *UnHideColsOnDbClick* property to true. By default this is set to true.

[C#]

```
//Disables unhide column when double clicking as found in Excel.
this.gridControl1.UnHideColsOnDbClick = true;
```

[VB]

```
'Disables unhide column when double clicking as found in Excel.
Me.gridControl1.UnHideColsOnDbClick = True
```

#### 4.1.4.5.9 Highlighting Row and Column Headers

This feature is used to highlight the corresponding row and column headers of one or more cells that you selected. Using the **MarkRowHeader** and **MarkColHeader** properties, you can enable highlighting the row and column headers of the selected cells.

## Properties

Table 3: Properties Table

Property	Description	Data Type
MarkRowHeader	It is used to highlight row headers of the selected cells.	Boolean
MarkColHeader	It is used to highlight the column headers of the selected cells.	Boolean

By default the **MarkRowHeader** and **MarkColHeader** properties are set to **false**. To enable highlighting the row and column headers, set the **MarkRowHeader** and **MarkColHeader** properties to **true**. The following code examples illustrate this.

**[C#]**

```
this.gridControl1.MarkColHeader = true;
this.gridControl1.MarkRowHeader = true;
```

**[VB]**

```
Me.gridControl1.MarkColHeader = true
Me.gridControl1.MarkRowHeader = true
```



Figure 131: Highlighting Row and Column Headers

## Sample Link

To view samples from the dashboard:

- Open **Syncfusion Dashboard**.
- Select **UI > Windows Forms**.
- Click **Run Samples**.
- Navigate to **Grid > MS Excel-Style Features > Mark Header Demo**

#### 4.1.4.6 Formula Support

Setting the cell type of a cell to **FormulaCell**, will allow you to enter algebraic expressions using formulas and cell references. Cell references are entries such as A11 for column A row 11 or BA3 for column BA row 3. A formula is a defined calculation from the Formula Library which is included with Essential Grid. This Formula Library is extensible and lets you to add additional formulas.

This section discusses the following topics:

##### 4.1.4.6.1 Defining a FormulaCell

You can use Formula Cells for every cell in a grid or for just a few cells. Even if you set the **CellType** property to *FormulaCell* to every cell in a grid, the default behavior is to treat such cells as text box cells, unless you start the cell entry with an equal sign. If the cell value starts with an equal sign, then the cell is considered as a formula cell and its contents are treated as such.

To make all cells present in a grid as potential formula cells, you will have to set the cell type of the standard **BaseStyle** to *FormulaCell* by using the following code.

[C#]

```
// Set up a Formula Cell.  
this.gridControl1.BaseStylesMap["Standard"].StyleInfo.CellType =  
"FormulaCell";
```

[VB.NET]

```
' Set up a Formula Cell.  
Me.gridControl1.BaseStylesMap("Standard").StyleInfo.CellType =  
"FormulaCell"
```

#### 4.1.4.6.2 Using the Formula Library

Essential Grid's Formula Library contains the mathematical functions that are available in the .NET Framework's System.Math class. In addition, there are Sum and Avg members. You can also add additional functions to this library by using your own code.

	A	B	C	D
1	11	6	=a1+b1	
2	33	15	=a2+b2	
3	-2	18	=a3+b3	
4	=sum(a1:a3)	=avg(b1:b3)	=a4+b4	
5				

Figure 132: Sample Formula Library Usage

In the above screen shot, cell A2 has a formula that uses four different library functions: Sqrt, Pow, Cos, and Sin.



**Note:** For a complete list of these library functions, refer the Class Reference for "GridFormulaEngine".

#### 4.1.4.6.3 Supported Arithmetic Operators and Calculation Precedence

The current formula support will let you to enter well-formed parenthetical algebraic expressions with operators and operands. The nine supported operators are shown in the following precedence table, with operators on the same level being calculated as encountered when the expression is scanned from left to right.

##### Code Tables

Operation	Symbol	Calculation Precedence
Multiplication, Division	/ *	1st
Addition, Subtraction	+ -	2nd
Less Than, Greater Than, Equal, Less Than Or Equal, Greater Than Or Equal, Not Equal	< > = <= >= <>	3rd

The supported operands include those listed in the following table. An operand by itself is also a well-formed algebraic expression that can serve as an entire formula in a cell.

Operand	Example
number	532.1, -10.2, or 18.
cell reference	A12, BB1010, or Q18.
library formula with valid arguments	Abs(E14), Cos(-3.14), or Sum(A1:A14).
any well formed algebraic expression	E1+E2, Cos(2)<A4, or Abs(A1-A5).

Within a formula cell, a case is ignored. So, a1 is the same as A1, and Cos(3) is the same as COS(3).

#### 4.1.4.6.4 Inside Essential Grid's Formula Support

The Formula Cell control is implemented with four classes: **GridFormulaCellModel**, **GridFormulaCellRenderer**, **GridFormulaEngine** and **GridFormulaTag**. The **GridFormulaCellRenderer** class handles a couple of activation issues which are specific to displaying formulas when a formula cell gets activated. The **GridFormulaCellModel** class does some significant work in its **GetFormattedText** method override where calculations and formula parsing are initiated dynamically as required.

The **GridFormulaEngine** class does the actual parsing and calculation that is required to evaluate a formula in a cell. This class also maintains the Formula Library. The programmer can gain access to an engine object by using the **GridFormulaCellModel.Engine** property. It is this property that will let you add functions to (or remove functions from) the Formula Library. The use of the class is discussed in the next section.

Finally, the **GridFormulaTag** class is used in conjunction with the **GridStyleInfo** class that has a property of this type. The **GridFormulaTag** tracks the computed value of the cell in its **Text** property.

#### 4.1.4.6.5 Adding Formulas to the Formula Library

Here are the steps that are required to add a function to the Function Library.

1. First, define a method that has this signature.

[C#]

```
// Define a method whose name is the FormulaName.
public string MyLibraryFormulaName(string args)
```

[VB.NET]

```
' Define a method whose name is the FormulaName.
Public Function MyLibraryFormulaName(ByVal args As String) As String
```

Here `MyLibraryFormulaName` must be a name that has not been already used in the Function Library, and must include only letters and digits. If you want to replace an existing formula with a formula of the same name, first remove the existing formula before adding the new formula. Use the **GridFormulaEngine.RemoveFunction** method to remove a formula.

2. Then, write an implementation for your method. Here code is used to implement a function that will sum only positive numbers that are passed in as either a range like A1:A5 and/or a list such as A1, A4, A10. The code uses the **FormulaEngine** helper method to extract these values from the cells. The **GetCellsFromArgs** method will return an array of cells from a range such as A1:A5, and **GetValueFromArg** method will take cells such as A3 and return a value such as 123.3.

[C#]

```
// Implement your method.
public string ComputeSumPosNums(string args)
{
    GridFormulaCellModel model =
this.gridControl1.CellModels["FormulaCell"] as GridFormulaCellModel;
    if(model != null)
    {
        GridFormulaEngine engine = model.Engine;
        double sum = 0d;
        double d;
        string s1;

        // Loop through arguments and sum up the positive values.
        foreach(string r in args.Split(new char[]{',', '}))
        {
            // Cell Range.
            if(r.IndexOf(':') > -1)
            {
                foreach(string s in engine.GetCellsFromArgs(r))
```

```
{  
// s is a cell line a21 or c3...  
try  
{  
    s1 = engine.GetValueFromArg(s);  
}  
  
catch(Exception ex)  
{  
    return ex.Message;  
}  
if(s1 != "")  
{  
    // Add only if positive.  
    if(double.TryParse(s1, NumberStyles.Number,  
null, out d)  
        && d > 0)  
    {  
        sum += d;  
    }  
}  
}  
else  
{  
    try  
{  
        s1 = engine.GetValueFromArg(r);  
    }  
    catch(Exception ex)  
{  
        return ex.Message;  
}  
if(s1 != "")  
{  
    if(double.TryParse(s1, NumberStyles.Number, null,  
out d) && d > 0)  
    {  
        sum += d;  
    }  
}  
}  
return sum.ToString();  
}  
return "";
```

```
}
```

**[VB.NET]**

```
' Implement your Method.

Public Function ComputeSumPosNums(args As String) As String
    Dim model As GridFormulaCellModel =
        Me.gridControl1.CellModels("FormulaCell")

    If Not (model Is Nothing) Then
        Dim engine As GridFormulaEngine = model.Engine
        Dim sum As Double = 0.0
        Dim d As Double
        Dim s1 As String

        ' Loop through arguments and sum up the positive values.
        Dim r As String
        For Each r In args.Split(New Char() {",", "c"})

            ' Cell Range.
            If r.IndexOf(":") > -1 Then
                Dim s As String
                For Each s In engine.GetCellsFromArgs(r)

                    ' s is a cell line a21 or c3...
                    Try
                        s1 = engine.GetValueFromArg(s)
                    Catch ex As Exception
                        Return ex.Message
                    End Try
                    If s1 <> "" Then

                        ' Add only if positive.
                        If Double.TryParse(s1, NumberStyles.Number, Nothing,
                            d) And d > 0 Then
                            sum += d
                        End If
                    End If
                Next s
            Else
                Try
                    s1 = engine.GetValueFromArg(r)
                Catch ex As Exception
                    Return ex.Message
                End Try
            End If
        Next r
    End If
End Function
```

```
If s1 <> "" Then
    If Double.TryParse(s1, NumberStyles.Number, Nothing, d)
        And d > 0 Then
            sum += d
        End If
    End If
Next r
Return sum.ToString()
End If
Return ""
End Function
```

3. The last step is to actually add your formula to the library. You should do this after the grid has been created, say in a Form.Load event handler.

[C#]

```
GridFormulaCellModel cellModel =
this.gridControl1.CellModels["FormulaCell"] as GridFormulaCellModel;

// Add a formula named SumPosNums to the Library.
cellModel.Engine.AddFunction("SumPosNums", new
GridFormulaEngine.LibraryFunction(ComputeSumPosNums));
```

[VB .NET]

```
Dim cellModel As GridFormulaCellModel =
Me.gridControl1.CellModels("FormulaCell")

' Add a formula named SumPosNums to the Library.
cellModel.Engine.AddFunction("SumPosNums", New
GridFormulaEngine.LibraryFunction(AddressOf ComputeSumPosNums))
```

#### 4.1.4.6.6 Function Reference Section

In this section, the library functions that are shipped in the Essential Calculate library are discussed. The arguments required by each of these functions are listed in bold text. Optional arguments are listed in a normal text.

Following functions are discussed under this section:

#### [4.1.4.6.6.1 ABS](#)

Returns the absolute value of a number. The absolute value of a non-negative number is the number itself. The absolute value of a negative number is -1 times the number.

#### **Syntax**

**ABS(*number*)**, where:

**number** is the real number for which you want the absolute value.

#### [4.1.4.6.6.2 ACCRINT](#)

The ACCRINT function returns the accrued interest for a security that pays periodic interest.

#### **Syntax:**

**ACCRINT(*issue*, *first\_interest*, *settlement*, *rate*, *par*, *frequency*, [*basis*], [*calc\_method*])**

where:

- *issue* - security's issue date.
- *first\_interest* - security's first interest date.
- *settlement* - security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer.
- *rate* - security's annual coupon rate.
- *par* - security's par value.
- *frequency* - number of coupon payments per year.

#### **Remarks:**

#VALUE - occurs if *issue*, *first\_interest*, or *settlement* is not a valid date

#NUM! - occurs if *rate* ≤ 0 or if *par* ≤ 0

#NUM! - occurs if *frequency* is any number other than 1, 2, or 4

#NUM! - occurs if *basis* < 0 or if *basis* > 4

#NUM! - occurs if *issue* ≥ *settlement*

ACCRINT is calculated as follows:

$$ACCRINT = par \times \frac{rate}{frequency} \times \sum_{j=1}^{NCF} \frac{A_j}{NL_j}$$

where:

$A_i$  is the number of accrued days for the  $i$ th quasi-coupon period within an odd period.

$N_C$  is the number of quasi-coupon periods that fit in an odd period. If this number contains a fraction, raise it to the next whole number.

$N_{Li}$  is the normal length in days of the  $i$ th quasi-coupon period within an odd period.

#### [4.1.4.6.6.3 ACOS](#)

Returns the inverse cosine of a number. Inverse cosine is also referred to as arccosine. The arccosine is the angle whose cosine is the given number. The returned angle is given in radians in the range of 0 to pi.

#### Syntax

**ACOS(*number*)**, where:

**number** is the cosine of the angle that you want and must be between -1 and 1.

#### [4.1.4.6.6.4 ACOSH](#)

Returns the inverse hyperbolic cosine of a number. The number must be greater than or equal to 1. The inverse hyperbolic cosine is the value whose hyperbolic cosine is the given number.

#### Syntax

**ACOSH(*number*)**, where:

**number** is any real number that is greater than or equal to 1.

#### [4.1.4.6.6.5 ACOT](#)

The ACOT function retrieves the principal value of the inverse cotangent or arcotangent of a number.

#### Syntax:

**ACOT(*number*)** where:

- *number* is the cotangent of the angle you need.

**Remarks:**

#VALUE! - occurs if the number is a non-numeric value.

The returned angle is given in radians in the range of 0 (zero) to pi.

[\*\*4.1.4.6.6.6 ACOTH\*\*](#)

The ACOTH function retrieves the inverse hyperbolic cotangent of a value.

**Syntax:**

**ACOTH(number)** where:

- number is the cotangent of the angle you need.

**Remarks:**

#NUM! - occurs if number is less than one.

#VALUE! - occurs if absolute value of number is less than one.

This mathematical equation is used:

$$\coth(N) = \frac{1}{2} \ln\left(\frac{x+1}{x-1}\right)$$

[\*\*4.1.4.6.6.7 Acsch\*\*](#)

The AcscH function computes the inverse hyperbolic cosecant of its argument.

**Syntax:**

**x = acsch(y)** where:

- x is a complex or real array
- y is a complex or real array

[\*\*4.1.4.6.6.8 ADDRESS\*\*](#)

The ADDRESS function returns the address of a cell in a worksheet given specified row and column numbers.

**Syntax**

**ADDRESS(row\_num, column\_num, [abs\_num], [a1], [sheet\_text])**

**row\_num:** A numeric value that specifies the row number.

**column\_num:** A numeric value that specifies the column number

**abs\_num:** Optional. A numeric value that specifies the type of reference to return.

**A1:** A logical value that specifies the A1 or R1C1 reference style.

**Example**

FORMULA	RESULT
=ADDRESS(2,3,2, FALSE)	R2C[3].
=ADDRESS(2,3,1, FALSE, "[Book1]Sync1")	[Book1] Sync1!R2C3

**4.1.4.6.6.9 AND**

Returns True if all the arguments have a logical value of True and returns False if at least one argument is False.

**Syntax**

**AND(*logical1, logical2, ...*)**, where:

**logical1, logical2, ...** are multiple conditions you want to test for True or False.

**Remarks**

- The arguments must evaluate to logical values (True or False).
- If an argument does not evaluate to True or False, those values are ignored.
- There must be at least one value in the argument list.

**4.1.4.6.10 ARABIC**

A Roman numeral has been converted into an Arabic numeral.

**Syntax:**

**ARABIC( *text* )** where *text* is a string.

**Remarks:**

#VALUE! - occurs if *text* is not a valid value.

#VALUE! - occurs if *text* is not a valid Roman numeral.

Value zero occurs if an empty string is given as an input.

**4.1.4.6.11 AsecH**

The AsecH function computes the element-wise inverse hyperbolic secant of the argument.

**Syntax:**

**x = asech(y)** where:

- x is a complex or real array.
- y is a complex or real array.

#### [4.1.4.6.6.12 ASIN](#)

Returns the inverse sine of a number. Inverse sine is also referred to as arcsine. The arcsine is the angle whose sine is the given number. The returned angle is given in radians in the range from -pi/2 to +pi/2.

#### Syntax

**ASIN(number)**, where:

**number** is the sine of the angle that you want and must be between -1 and 1.

#### [4.1.4.6.6.13 ASINH](#)

Returns the inverse hyperbolic sine of a number. The inverse hyperbolic sine is the value whose hyperbolic sine is the given number, so ASINH(SINH(number)) equals number.

#### Syntax

**ASINH(number)**, where:

**number** is any real number.

#### [4.1.4.6.6.14 ATAN](#)

Returns the inverse tangent of a number. Inverse tangent is also known as arctangent. The arctangent is the angle whose tangent is a number. The returned angle is given in radians in the range from -pi/2 to +pi/2.

#### Syntax

**ATAN(number)**, where:

**number** is the tangent of the angle that you want.

#### [4.1.4.6.15 ATAN2](#)

Returns the inverse tangent of the specified x- and y-coordinates. The arctangent is the angle from the x-axis to a line containing the origin (0, 0) and the point (x\_num, y\_num). The angle is given in radians between -pi and pi, excluding -pi.

#### Syntax

**ATAN2(x\_num,y\_num)**, where:

**x\_num** is the X coordinate of the point.

**y\_num** is the Y coordinate of the point.

#### Remarks

- A positive result represents a counterclockwise angle from the x-axis; a negative result represents a clockwise angle.
- ATAN2(a,b) equals ATAN(b/a), except that a can equal 0 in ATAN2.

#### [4.1.4.6.16 ATANH](#)

Returns the inverse hyperbolic tangent of a number. Number must be strictly between -1 and 1. The inverse hyperbolic tangent is the value whose hyperbolic tangent is number, so ATANH(TANH(number)) equals the given number.

#### Syntax

**ATANH(number)**, where:

**number** is any real number that is between 1 and -1.

#### [4.1.4.6.17 AVEDEV](#)

Returns the average of the absolute mean deviations of data points. AVEDEV is a measure of the variability in a data set.

#### Syntax

**AVEDEV(number1, number2, ...), where:**

**number1, number2, ...** are arguments for which, you want the average of the absolute deviations. You can also use a single array or a reference to an array instead of arguments separated by commas.

### **Remarks**

- The arguments must either be numbers or names, arrays or references that contain numbers.
- If an array or reference argument contains text, logical values or empty cells, those values are ignored; however, cells with a zero value are included.
- The equation for average deviation is,

$$\frac{1}{n} \sum |x - \bar{x}|$$

- where  $\bar{x}$  is the arithmetic mean of the data.

#### [4.1.4.6.6.18 AVERAGE](#)

Returns the average (arithmetic mean) of the arguments.

### **Syntax**

**AVERAGE(number1, number2, ...), where:**

**number1, number2, ...** are numeric arguments for which you want the average.

### **Remarks**

- The arguments must either be numbers or names, arrays or references that contain numbers.
- If an array or reference argument contains text, logical values or empty cells, those values are ignored; however, cells with a zero value are included.

#### [4.1.4.6.6.19 AVERAGEA](#)

Calculates the average (arithmetic mean) of the values in the list of arguments. In addition to numbers and text logical values such as True and False are also included in the calculation.

## Syntax

**AVERAGEA(value1, value2, ...)**, where:

**value1, value2, ...** are cells, ranges of cells, or values for which you want the average.

## Remarks

- The arguments must be numbers, names, arrays, or references.
- Array or reference arguments that contain text evaluate as 0 (zero). If the calculation should not include text values in the average, then use the AVERAGE function.
- Arguments that contain True evaluate as 1; arguments that contain False evaluate as 0 (zero).

### 4.1.4.6.6.20 AVERAGEIF

The AVERAGEIF function finds the average of values in a given array that satisfy a given criteria, and returns the average value of the corresponding values in a second given array.

## Syntax

=AVERAGEIF(range, criteria, average\_range)

range: Array of values to be tested against the given criteria.

criteria: The condition to be tested in each of the values of the given range.

average\_range: Numeric values to be evaluated against the criteria and averaged.

## Notes

- If range is blank or a text value, AVERAGEIF returns the #DIV/0! error value.
- If a cell in criteria is empty, AVERAGEIF treats it as a 0 value.
- If no cells in the range meet the criteria, AVERAGEIF returns the #DIV/0! error value.

## Example

### Input Table

	A	B
1	Earning	Tax
2	100000	3000

<b>3</b>	200000	6000
<b>4</b>	300000	7500
<b>5</b>	400000	9000

FORMULA	RESULT
=AVERAGEIF(B2:B5,"<7000")	4500
=AVERAGEIF(A2:A5,">250000")	350000

#### 4.1.4.6.6.21 AVERAGEIFS

The AVERAGEIFS function finds the average of values in a given array that satisfy a set of given criteria.

#### Syntax

= AVERAGEIFS( average\_range, criteria\_range1, criteria1, [criteria\_range2, criteria2], ... )

average\_range: Specific set of values to be averaged if the criteria range meets the provided criteria.

criteria\_range1: Array of values to be tested against the given criteria.

criteria1: The condition to be tested on each of the values of the given range.

#### Notes

- If average\_range is blank or a text value, AVERAGEIFS returns the #DIV/0! error value.
- If a cell in a criteria range is empty, AVERAGEIFS treats it as a 0 value.
- If cells in average\_range cannot be translated into numbers, AVERAGEIFS returns the #DIV/0! error value.

#### Example

Input Table

	A	B	C
<b>1</b>	Earning	Tax	other
<b>2</b>	100000	3000	100
<b>3</b>	200000	6000	200
<b>4</b>	300000	7500	300
<b>5</b>	400000	9000	500

FORMULA	RESULT
AVERAGEIFS(C2:C5, B2:B5, ">7000", B2:B5, "<10000")	400

#### 4.1.4.6.6.22 AVG

Returns the average (arithmetic mean) of the arguments.

#### Syntax

**AVG(number1, number2, ...)**, where:

**number1, number2, ...** are numeric arguments for which, you want the average.

#### Remarks

- This method is the same as AVERAGE and is included for compatibility purposes.

#### 4.1.4.6.6.23 BASE

A number has been converted into a text representation with the given radix (base).

#### Syntax:

**BASE(Number, Radix [Min\_length])** where:

- Number is the value that you want to convert.
- Radix is the base radix that you want to convert the number into.
- Min\_length is the minimum length of the returned string. Min\_length is optional.

#### Remarks:

#NUM! - occurs if Number, Radix, or Min\_length are outside the minimum or maximum constraints.

#VALUE! - occurs if Number is a non-numeric value.

#### 4.1.4.6.6.24 BigMul

The BigMul function gives the full value of multiplying two 32-bit numbers.

#### Syntax:

**Math.BigMul(x,y);** where:

- x is the first number to multiply
- y is the second nr to multiply.

#### [4.1.4.6.6.25 BINOMDIST](#)

Returns the individual term binomial distribution probability.

### Syntax

**BINOMDIST(number\_s, trials, probability\_s, cumulative)**, where:

**number\_s** is the number of successes in trials.

**trials** is the number of independent trials.

**probability\_s** is the probability of success on each trial.

**Cumulative** is a logical value that determines the form of the function. If cumulative is True, then BINOMDIST returns the cumulative distribution which, is the probability that there are at most number\_s successes; if False, it returns the probability that there are exactly number\_s successes.

### Remarks

- Number\_s and trials are truncated to integers.
- Number\_s should be  $\geq 0$  and  $\leq$  trials.
- Probability\_s should be  $\geq 0$  and  $\leq 1$ .
- The binomial probability mass function is,

$$b(x,n,p) = \binom{n}{x} p^x (1-p)^{n-x}$$

where:

$$\binom{n}{x}$$

is COMBIN(n,x).

- The cumulative binomial distribution is,

$$B(x,n,p) = \sum_{y=0}^x b(y,n,p)$$

#### 4.1.4.6.6.26 Binom.Inv

The Binom.Inv function returns the smallest value for which the cumulative binomial distribution is greater than or equal to a criterion value.

##### Syntax:

**BINOM.INV(trials,probability\_s,alpha)** where:

- trials is the number of Bernoulli trials.
- probability\_s is the probability of a success on each trial.
- alpha is the criterion value.

##### Remarks:

#NUM! - occurs if trials is less than zero.

#VALUE! - occurs if trials is non-numeric.

#NUM! - occurs if probability\_s is less than zero.

#NUM! - occurs if probability\_s is greater than one.

#VALUE! - occurs if probability\_s is non-numeric.

#NUM! - occurs if alpha is less than zero.

#NUM! - occurs if alpha is greater than one.

#VALUE! - occurs if alpha is non-numeric.

#### 4.1.4.6.6.27 CEILING

Returns number rounded up, away from zero, to the nearest multiple of significance. For example, if you want to avoid using pennies in your prices and your product is priced at \$4.82, use the formula =CEILING(4.82,0.05) to round prices up to the nearest nickel.

##### Syntax

**CEILING(number, significance)**, where:

**number** is the value you want to round off.

**significance** is the multiple to which you want to round.

## Remarks

- Both values must be numeric.
- Regardless of the sign of a number, a value is rounded up when adjusted away from zero. If the number is an exact multiple of significance, no rounding occurs.

### [4.1.4.6.6.28 CEILING.MATH](#)

The CEILING.MATH function rounds a number up to the nearest multiple of significance.

#### Syntax:

**CEILING(number, [significance], [mode])** where:

- number must be less than 9.99E+307 and greater than -2.229E-308.
- significance must be the multiple to which the number is to be rounded.
- mode is for negative numbers, it controls whether the number is rounded toward or away from zero.

### [4.1.4.6.6.29 CHIDIST](#)

Returns the one-tailed probability of the chi-squared ( $\chi^2$ ) distribution. The  $\chi^2$  distribution is associated with a  $\chi^2$  test.

#### Syntax

**CHIDIST(x, degrees\_freedom)**, where:

**x** is the value at which you want to evaluate the distribution.

**degrees\_freedom** is the number of degrees of freedom.

## Remarks

- Both arguments should be numeric.
- **degrees\_freedom**  $\geq 1$  and  $< 10^{10}$ .
- CHIDIST is calculated as  $\text{CHIDIST} = P(X > x)$ , where  $X$  is a  $\chi^2$  random variable.

### [4.1.4.6.6.30 CHIINV](#)

Returns the inverse of the one-tailed probability of the chi-squared ( $\chi^2$ ) distribution. If probability = CHIDIST(x,...), then CHIINV(probability,...) = x. Use this function to compare observed results with expected ones in order to decide whether your original hypothesis is valid.

## Syntax

**CHIINV(probability, degrees\_freedom)**, where:

**probability** is a probability associated with the chi-squared distribution.

**degrees\_freedom** is the number of degrees of freedom.

## Remarks

- Probability must be  $\geq 0$  and  $\leq 1$ .
- degrees\_freedom  $\geq 1$  and  $= 10^{10}$ .

Given a value for probability, CHIINV seeks the value x such that CHIDIST(x, degrees\_freedom) = probability. Thus, precision of CHIINV depends on precision of CHIDIST. CHIINV uses an iterative search technique.

### 4.1.4.6.6.31 CHI- TEST

Returns the test for independence. CHITEST returns the value from the chi-squared (c2) distribution for the statistic and the appropriate degrees of freedom.

## Syntax

**CHITEST(actual\_range, expected\_range)**, where:

**actual\_range** is the range of data that contains observations to test against expected values.

**expected\_range** is the range of data that contains the ratio of the product of row totals and column totals to the grand total.

## Remarks

- The  $\chi^2$  test first calculates a  $\chi^2$  statistic using the formula,

$$\chi^2 = \sum_{i=1}^r \sum_{j=1}^c \frac{(A_{ij} - E_{ij})^2}{E_{ij}}$$

where:

$A_{ij}$  = actual frequency in the i-th row, j-th column  
 $E_{ij}$  = expected frequency in the i-th row, j-th column  
 $r$  = number of rows  
 $c$  = number of columns  
A low value of  $\chi^2$  is an indicator of independence.

The use of CHITEST is most appropriate when  $E_{ij}$ 's are not too small. Some statisticians suggest that each  $E_{ij}$  should be greater than or equal to 5.

#### [4.1.4.6.32 CLEAN](#)

The CLEAN function is used to remove the non-printable characters from the given text, represented by numbers 0 to 31 of the 7-bit ASCII code.

##### Syntax

=Clean(Text)

**Text:** Required. String or text from which to remove nonprintable characters.

##### Example

FORMULA	RESULT
=Clean(Syncfusion)	Syncfusion
= Clean("Text*")	Text

#### [4.1.4.6.33 COMBIN](#)

Returns the number of combinations for a given number of items. Use COMBIN to determine the total possible number of groups for a given number of items.

##### Syntax

**COMBIN(number, number\_chosen)**, where:

**number** is the number of items.

**number\_chosen** is the number of items in each combination.

## Remarks

- Numeric arguments are truncated to integers.
- A combination is any set or subset of items, regardless of their internal order. Combinations are distinct from permutations where the internal order is significant.
- The number of combinations is as follows, where number = n and number\_chosen = k,

$$\binom{n}{k} = \frac{P_{k,n}}{k!} = \frac{n!}{k!(n-k)!}$$

where:

$$P_{k,n} = \frac{n!}{(n-k)!}$$

### 4.1.4.6.34 COMBINA

For a given number of items, the COMBINA function returns the number of combinations (with repetitions).

#### Syntax:

**COMBINA(number1, number2)** where:

- number 1 is greater than equal to zero and greater than equal to number2
- number 2 is greater than equal to zero.

#### Remarks:

#NUM! - occurs if either value is outside of its constraints

#VALUE! - occurs if either value is non-numeric.

The following equation is used:

$$\binom{N+M-1}{N-1}$$

### 4.1.4.6.35 CONCATENATE

Joins several text strings into one text string.

#### Syntax

**CONCATENATE (text1, text2, ...), where:**

**text1, text2, ...** are text items to be joined into a single text item. The text items can be text strings, numbers, or single-cell references.

**Remarks**

- The "&" operator can be used instead of CONCATENATE to join text items.

[4.1.4.6.36 CONFIDENCE](#)

Returns a value that you can use to construct a confidence interval about a population mean. The confidence interval is a range of values. In your sample, mean  $x$  is at the center of this range and the range is  $x \pm \text{CONFIDENCE}$ . For example, if  $x$  is the sample mean of delivery times for products ordered through the mail,  $x \pm \text{CONFIDENCE}$  is a range of population means.

**Syntax**

**CONFIDENCE(alpha, standard\_dev, size), where:**

**alpha** is the significance level used to compute the confidence level. The confidence level equals  $100*(1 - \text{alpha})\%$ , or in other words, an alpha of 0.05 indicates a 95 percent confidence level.

**standard\_dev** is the population standard deviation for the data range and is assumed to be known.

**size** is the sample size.

**Remarks**

- All arguments must be non-numeric.
- Alpha must be  $> 0$  and  $< 1$ .
- Standard\_dev must be  $> 0$ .
- Size must be  $\geq 1$ .

[4.1.4.6.37 CONFIDENCE.T](#)

Using a student's distribution this function retrieves the confidence interval for a population mean.

**Syntax:**

**CONFIDENCE.T(alpha,standard\_dev,size) where:**

- alpha is the significance level used to compute the confidence level.

- standard\_dev is the population standard deviation for the data range and is assumed to be known.
- size is the sample size.

**Remarks:**

#VALUE! - occurs if any argument is non-numeric.

#NUM! - occurs if alpha is less than or equal to zero or if alpha is greater than or equal to zero.

#NUM! - occurs if standard\_dev is less than or equal to zero.

#DIV/0! - occurs if the size is equal to one.

[4.1.4.6.38 CORREL](#)

Returns the correlation coefficient of the array1 and array2 cell ranges.

**Syntax**

**CORREL(array1, array2)**, where:

**array1** is a cell range of values.

**array2** is the second cell range of values.

**Remarks**

- array1 and array2 must have the same number of data points.
- The equation for the correlation coefficient is,

$$\text{Correl}(X,Y) = \frac{\sum(x - \bar{x})(y - \bar{y})}{\sqrt{\sum(x - \bar{x})^2 \sum(y - \bar{y})^2}}$$

where x-bar and y-bar are the sample means AVERAGE(array1) and AVERAGE(array2).

[4.1.4.6.39 COS](#)

Returns the cosine of the given angle.

## Syntax

**COS(number)**, where:

**number** is the angle in radians for which you want the cosine.

### [4.1.4.6.6.40 COSH](#)

Returns the hyperbolic cosine of a number.

## Syntax

**COSH(number)**, where:

**number** is any real number for which you want to find the hyperbolic cosine.

## Remarks

The formula for the hyperbolic cosine is,

$$\cosh(z) = \frac{e^z + e^{-z}}{2}$$

### [4.1.4.6.6.41 COT](#)

The COT function returns the cotangent of an angle specified in radians.

## Syntax:

**COT(number)** where:

- **number** – the angle radians for which you want the secant.

#NUM! - occurs if the number is outside of its constraints.

#VALUE! - occurs if the number is a non-numeric value.

#### 4.1.4.6.6.42 COTH

The COTH function returns the hyperbolic cotangent of a hyperbolic angle.

##### Syntax:

**COTH(number)** where:

- number – the angle radians for which you want the secant.

##### Remarks:

#NUM! - occurs if the number is outside of its constraints.

#VALUE! - occurs if the number is a non-numeric value.

The following equation is used:

$$\coth(N) = \frac{1}{\tanh(N)} = \frac{\cosh(N)}{\sinh(N)} = \frac{e^N + e^{-N}}{e^N - e^{-N}}$$

#### 4.1.4.6.6.43 COUNT

Counts the number of items in a list that contains numbers.

##### Syntax

**COUNT(value1, value2, ...),** where:

**value1, value2, ...** are arguments that can contain or refer to a variety of different types of data but, only numbers are counted.

##### Remarks

- Arguments that are numbers, dates or text representations of numbers are counted; arguments that are error values or text that cannot be translated into numbers are ignored.
- If an argument is an array or reference, only numbers in that array or reference are counted. Empty cells, logical values, text or error values in the array or reference are ignored.

#### 4.1.4.6.6.44 COUNTA

Counts the number of cells that are not empty.

## Syntax

**COUNTA(value1, value2, ...)**, where:

**value1, value2, ...** are arguments representing the values you want to count. In this case, a value is any type of information, excluding empty cells.

### [4.1.4.6.6.45 COUNTBLANK](#)

Counts empty cells in a specified range of cells.

## Syntax

**COUNTBLANK(range)**, where:

**range** is the range from which you want to count the blank cells.

## Remark

- Cells with formulas that return "" (empty text) are also counted. Cells with zero values are not counted.

### [4.1.4.6.6.46 COUNTIF](#)

Counts the number of cells within a range that meet the given criteria.

## Syntax

**COUNTIF(range, criteria)**, where:

**range** is the range of cells from which you want to count cells.

**criteria** is the criteria in the form of a number, expression or text that defines which cells will be counted. For example, the criteria can be expressed as ">32".

## Remark

- If and SumIf are other library functions that can be used to conditionally compute values.

### 4.1.4.6.6.47 COVAR

Returns covariance, the average of the products of deviations for each data point pair.

## Syntax

**COVAR(array1, array2)**, where:

**array1** is the first cell range of numbers.

**array2** is the second cell range of numbers.

## Remarks

- The arguments must either be numbers or be names, arrays or references that contain numbers.
- array1 and array2 must have the same number of data points.
- The covariance is,

$$Cov(X,Y) = \frac{\sum(x - \bar{x})(y - \bar{y})}{n}$$

where X is array1, Y is array2, x-bar and y-bar are the sample means AVERAGE(array1) and AVERAGE(array2) and n is the sample size.

### 4.1.4.6.6.48 COVARIANCE.P

The COVARIANCE.P function retrieves population covariance, the average of the products of deviations for each data point pair in two data sets.

## Syntax:

**COVARIANCE.P(array1, array2)** where:

- array1 is the first cell range of integers.
- array2 is the second cell range of integers.

**Remarks:**

#N/A - occurs if arguments have a different number of data points.

The covariance is:

$$\text{Cov}(X, Y) = \frac{\sum (x - \bar{x})(y - \bar{y})}{n}$$

where x and y are the sample means, AVERAGE(array1) and AVERAGE(array2), and n is the sample size.

**4.1.4.6.49 COVARIANCE.S**

The COVARIANCE.S function returns the sample covariance, the average of the products of deviations for each data point pair in two data sets.

**Syntax:**

**COVARIANCE.S(array1,array2)** where:

- array1 is the first cell range of integers.
- array2 is the second cell range of integers.

**Remarks:**

#N/A - occurs when values are different number of data points.

#DIV/0! - occurs if either array1 or array2 is empty or contain only one data point each.

**4.1.4.6.50 CRITBINOM**

Returns the smallest value for which, the cumulative binomial distribution is greater than or equal to a criterion value.

**Syntax**

**CRITBINOM(trials, probability\_s, alpha)**, where:

**trials** is the number of Bernoulli trials.

**probability\_s** is the probability of a success on each trial.

**alpha** is the criterion value.

**Remarks**

- Trials must be  $\geq 0$ .
- Probability\_s must be  $\geq 0$  and  $\leq 1$ .
- Alpha must be  $\geq 0$  and  $\leq 1$ .

#### [4.1.4.6.6.51 CSC](#)

The CSC function returns the cosecant of an angle specified in radians.

**Syntax:**

**CSC(number)** where:

- number – the angle radians for which you want the secant.

**Remarks:**

#NUM! - occurs if the number is outside of its constraints.

#VALUE! - occurs if the number is a non-numeric value.

#### [4.1.4.6.6.52 CSCH](#)

The CSCH function returns the hyperbolic cosecant of an angle specified in radians.

**Syntax:**

**CSCH(number)** where:

- number – the angle radians for which you want the secant.

**Remarks:**

#NUM! - occurs if the number is outside of its constraints.

#VALUE! - occurs if the number is a non-numeric value.

#### [4.1.4.6.6.53 CUMIPMT](#)

Returns the Macaulay duration for an assumed par value of \$100. Duration is defined as the weighted average of the present value of the cash flows and is used as a measure of a bond price's response to changes in yield.

**Syntax:**

**DURATION(settlement,maturity,coupon,yld,frequency,basis)** where:

- Settlement - security's settlement date.
- Maturity - security's maturity date.
- Coupon – annual coupon rate
- Yld – security's annual yield.
- Frequency – number of coupon payments per year.

- Basis – type of day count basis.

**Remarks:**

#VALUE! – occurs if settlement or maturity is not a valid date.

#NUM! – occurs if coupon < 0 or yld < 0

# NUM! – occurs if frequency number is other than 1,2 and 4

# NUM! – occurs basis is less than 0 and greater than 4

# NUM! – occurs if settlement is >= Maturity

#### *4.1.4.6.6.54 CUMPRINC*

The CUMPRINC function returns the cumulative principal paid on a loan between the start\_period and end\_period.

**Syntax:**

**CUMPRINC(rate, nper, pv, start\_period, end\_period, type)** where:

- Rate – the interest rate.
- Nper - total number of payment periods
- pv - The present value.
- start\_period - first period in calculation. Brgind with one.
- end\_period - last period in calculation.
- type - timing of the payment.

**Remarks:**

#NUM! - occurs if rate,nper or pv is less than or equals zero.

#NUM! - occurs if start\_period or end\_period is less than one

#NUM! - occurs if start\_period is greater than end\_period.

#NUM! - occurs if type is any number other than 0 or 1

#### *4.1.4.6.6.55 DATE*

Returns the sequential serial number that represents a particular date.

**Syntax**

**DATE(year, month, day)**, where:

**year** can be one to four digits. Year is interpreted based on 1900.

- If a year is between 0 (zero) and 1899 (inclusive), the value is added to 1900 to calculate the year. For example, DATE(102,11,12) returns November 12, 2002 (1900+102).
- If a year is between 1900 and 9999 (inclusive), the value is used as is, for example, DATE(2002,11,12) returns November 12, 2002.  
**month** is a number representing the month of the year.  
**day** is a number representing the day of the month.

#### **Remark**

- Dates are stored as sequential serial numbers so that they can be used in calculations. By default, January 1, 1900 is serial number 1 and November 12, 2002 is serial number 37572 because it is 37572 days after January 1, 1900.

#### [4.1.4.6.6.56 DATEVALUE](#)

Returns the serial number of the date represented by the date\_text.

#### **Syntax**

**DATEVALUE(date\_text)**, where:

**date\_text** is the text that represents a date as a formatted string. For example, "11/12/2002" or "12-Nov-2002" are text strings within quotation marks that represent dates. If the year portion of the date\_text is omitted, DATEVALUE uses the current year from your computer's built-in clock. The time information in the date\_text is ignored.

#### **Remarks**

- Dates are stored as sequential serial numbers so that they can be used in calculations. By default, January 1, 1900 is serial number 1, and November 12, 2002 is serial number 37572 because it is 37572 days after January 1, 1900.
- Most functions automatically convert date values to serial numbers.

#### [4.1.4.6.6.57 DAVERAGE](#)

The DAVERAGE function finds the average values in a column of a list or database that matches the conditions that have been specified.

##### **Syntax:**

**DAVERAGE(database,field,criteria)** where:

- database is the range of cells that makes up the list or database
- field indicates which column is used in the function
- criteria is the range of cells that contains the conditions that you specify.

#### [4.1.4.6.6.58 DAY](#)

Returns the day of a date, represented by a serial number. The day is given as an integer ranging from 1 to 31.

##### **Syntax**

**DAY(serial\_number)**, where:

**serial\_number** is the date of the day you are trying to find. Dates should be entered by using the DATE function or as results of other formulas or functions. For example, use DATE(2002,4,23) for the 23rd day of April, 2002.

#### [4.1.4.6.6.59 DAYS360](#)

Returns the number of days between two dates based on a 360-day year (twelve 30-day months) which, is used in some accounting calculations.

##### **Syntax**

**DAYS360(start\_date, end\_date, method)**, where:

**start\_date** and **end\_date** are the two dates between which you want to know the number of days. If start\_date occurs after end\_date, DAYS360 returns a negative number. Dates should be entered by using the DATE function or as results of other formulas or functions.

**method** is a logical value that specifies whether to use the U.S. or European method in the calculation.

If method is:

- **False or omitted** – The calculation uses the U.S. (NASD) method. If the starting date is the 31st of a month, it becomes equal to the 30th of the same month. If the ending date is the 31st of a month and the starting date is earlier than the 30th of a month, the ending date becomes equal to the 1st of the next month; otherwise the ending date becomes equal to the 30th of the same month.
- **True** – The calculation uses the European method. Starting dates and ending dates that occur on the 31st of a month become equal to the 30th of the same month.

#### [4.1.4.6.6.60 DB](#)

Returns the depreciation of an asset for a specified period using the fixed-declining balance method.

#### Syntax

**DB(cost, salvage, life, period, month)**, where:

**cost** is the initial cost of the asset.

**salvage** is the value at the end of the depreciation (sometimes called the salvage value of the asset).

**life** is the number of periods over which the asset is being depreciated (sometimes called the useful life of the asset).

**period** is the period for which you want to calculate the depreciation. Period must use the same units as life.

**month** is the number of months in the first year. If month is omitted, it is assumed to be 12.

#### Remarks

- The fixed-declining balance method computes the depreciation at a fixed rate. DB uses the following formulas to calculate the depreciation for a period,

$$(\text{cost} - \text{total depreciation from prior periods}) * \text{rate}$$

where  $\text{rate} = 1 - ((\text{salvage} / \text{cost}) ^ (1 / \text{life}))$ , rounded to three decimal places.

- Depreciation for the first and last periods is a special case. For the first period, DB uses this formula,

$$\text{cost} * \text{rate} * \text{month} / 12$$

- For the last period, DB uses this formula,

$$((\text{cost} - \text{total depreciation from prior periods}) * \text{rate} * (12 - \text{month})) / 12$$

#### [4.1.4.6.6.61 DCOUNT](#)

The DCOUNT function counts the number of cells that contain numbers in a column of a list or database which matches the conditions that is been specified.

##### **Syntax:**

**DCOUNT(database, field,criteria )** where:

- database is the range of cells that makes up the list or database
- field indicates which column is used in the function
- criteria is the range of cells that contains the conditions that you specify.

#### [4.1.4.6.6.62 DCOUNTA](#)

The DCOUNTA function counts the number of nonblank cells in a column of a list or database that matches the conditions that have been specified.

##### **Syntax:**

**DCOUNTA(database, field, criteria)** where:

- database is the range of cells that makes up the list or database.
- field indicates which column is used in the function.
- criteria is the range of cells that contains the conditions that you specify.

#### [4.1.4.6.6.63 DDB](#)

Returns the depreciation of an asset for a specified period using the double-declining balance method or some other method you specify.

##### **Syntax**

**DDB(cost, salvage, life, period, factor)**, where:

**cost** is the initial cost of the asset.

**salvage** is the value at the end of the depreciation (sometimes called the salvage value of the asset).

**life** is the number of periods over which the asset is being depreciated (sometimes called the useful life of the asset).

**period** is the period for which you want to calculate the depreciation. Period must use the same units as life.

**factor** is the rate at which the balance declines. If factor is omitted, it is assumed to be 2 (the double-declining balance method).



**Note:** All five arguments must be positive numbers.

### Remarks

- The double-declining balance method computes the depreciation at an accelerated rate. Depreciation is highest in the first period and decreases in successive periods. DDB uses the following formula to calculate depreciation for a period,

$$((\text{cost}-\text{salvage}) - \text{total depreciation from prior periods}) * (\text{factor}/\text{life})$$

#### *4.1.4.6.64 DECIMAL*

A text representation of a number in a given base has been converted into a decimal number.

### Syntax:

**DECIMAL(text, radix)** where:

- text is a string.
- radix is an integer.

### Remarks:

#NUM! or #VALUE! - occurs if text or radix is outside the constraints.

#### *4.1.4.6.65 DEGREES*

Converts radians into degrees.

### Syntax

**DEGREES(angle)**, where:

**angle** is the angle in radians that you want to convert.

#### [4.1.4.6.66 DEVSQ](#)

Returns the sum of squares of deviations of data points from their sample mean.

#### Syntax

**DEVSQ(number1, number2, ...)**, where:

number1, number2, ... are arguments for which you want to calculate the sum of squared deviations. You can also use a single array or a reference to an array instead of arguments separated by commas.

#### Remarks

- The arguments must be numbers or names, arrays or references that contain numbers.
- The equation for the sum of squared deviations is,

$$DEVSQ = \sum(x - \bar{x})^2$$

#### [4.1.4.6.67 DGET](#)

The DGET function extracts a single value from a column of a list or database which matches the conditions that have been specified.

#### Syntax:

**DGET(database,field,criteria)** where:

- database is the range of cells that makes up the list or database.
- field indicates which column is used in the function.
- criteria is the range of cells that contains the conditions that you specify.

#### [4.1.4.6.68 Disc](#)

The Disc function returns the discount rate for a security.

#### Syntax:

**Disc(Arg1, Arg2, Arg3, Arg4, Arg5)** where:

- Arg1 represents the security's settlement date.
- Arg2 represents the security's maturity date.
- Arg3 represents the security' price per \$100 face value.
- Arg4 represents the security's redemption value.
- Arg5 represents the type of day count basis to use.

#### [4.1.4.6.6.69 DivRem](#)

The DivRem function calculates the quotient of two numbers and also returns the remainder in an output parameter.

##### **Syntax:**

**Math.DivRem(a,b)** where:

a and b are integers.

#### [4.1.4.6.6.70 DMAX](#)

The DMAX function returns the largest number in a column of a list or database which matches the conditions that have been specified.

##### **Syntax:**

**DMAX(database,field,criteria)** where:

- database is the range of cells that makes up the list or database.
- field indicates which column is used in the function.
- criteria is the range of cells that contains the conditions that you specify.

#### [4.1.4.6.6.71 DMIN](#)

The DMIN function returns the smallest number in a column of a list or database which matches the conditions that have been specified.

##### **Syntax:**

**DMIN(database,field,criteria)** where:

- database is the range of cells that makes up the list or database.
- field indicates which column is used in the function.
- criteria is the range of cells that contains the conditions that you specify.

#### [4.1.4.6.6.72 Dollar](#)

The Dollar function converts a number to text, using currency format.

The format used is `$#,##0.00_);($#,##0.00)`.

##### **Syntax:**

**Dollar (number, decimal\_places)**

Where:

**number** is the number, which you want to convert to text.

**decimal\_places** is the number of decimal digits to be displayed. The value will be rounded off accordingly.

#### *4.1.4.6.6.73 DollarDe*

A dollar price expressed as a fraction will be converted into a dollar price expressed as a decimal number.

##### **Syntax:**

**DOLLARDE(object arg1, object arg2)** where:

- Arg1 is a number expressed as a fraction.
- Arg2 is an integer to use in the denominator of the fraction.

##### **Remarks:**

#NUM! - Occurs if the fraction is less than 0.

#DIV/0! - Occurs if the fraction is equal to 0.

#### *4.1.4.6.6.74 DollarFr*

A dollar price expressed as a decimal number will be converted into a dollar price expressed as a fraction.

##### **Syntax:**

**DOLLARFR (object arg1,object arg2)** where:

arg1 is a decimal number.

arg2 is an integer to use in the denominator of a fraction.

##### **Remarks:**

#NUM! - Occurs if the fraction is less than 0.

#DIV/0! - Occurs if the fraction is 0.

#### *4.1.4.6.6.75 DSTDEV*

The DSTDEV function estimates the standard deviation of a population based on a sample by using the numbers in a column of a list or database that matches the conditions that have been specified.

##### **Syntax:**

**DSTDEV(database, field, criteria)** where:

- database is the range of cells that makes up the list or database.
- field indicates which column is used in the function.
- criteria is the range of cells that contains the conditions that you specify.

#### [4.1.4.6.6.76 DSTDEVP](#)

The DSTDEVP function calculates the standard deviation of a population based on the entire population, using the numbers in a column of a list or database that match the conditions that have been specified.

##### **Syntax:**

**DSTDEVP(database,field,criteria)** where:

- database is the range of cells that makes up the list or database.
- field indicates which column is used in the function.

criteria is the range of cells that contains the conditions that you specify.

#### [4.1.4.6.6.77 DSUM](#)

The DSUM function adds the numbers in a field (column) of records in a list or database that matches the conditions that have been specified.

##### **Syntax:**

**DSUM(database, field, criteria)** where:

- database is the range of cells that makes up the list or database.
- field indicates which column is used in the function.
- criteria is the range of cells that contains the conditions that you specify.

#### [4.1.4.6.6.78 Duration](#)

Returns the annual duration of a security with periodic interest payments.

##### **Syntax:**

**Duration (settlement, maturity, coupon, yld, frequency, basis)** where:

- Settlement – a security's settlement date.
- Maturity – a security's maturity date.
- Coupon – the annual coupon rate.
- Yld – security's annual yield.
- Frequency – the number of coupon payments per year.
- Basis – the type of day count basis.

#### [4.1.4.6.6.79 DVAR](#)

The DVAR function estimates the variance of a population based on a sample by using the numbers in a column of a list or database which matches the conditions that is been specified.

##### **Syntax:**

**VAR(database,field,criteria)** where:

- database is the range of cells that makes up the list or database.
- field indicates which column is used in the function.
- criteria is the range of cells that contains the conditions that you specify.

#### [4.1.4.6.6.80 DVARP](#)

The DVARP function calculates the variance of a population based on the entire population by using the numbers in a column of a list or database which matches the conditions that have been specified.

##### **Syntax:**

**DVARP(database,field,criteria)** where:

- database is the range of cells that makes up the list or database.
- field indicates which column is used in the function.
- criteria is the range of cells that contains the conditions that you specify.

#### [4.1.4.6.6.81 EVEN](#)

Returns the number rounded up to the nearest even integer.

##### **Syntax**

**EVEN(number)**, where:

**number** is the value that is to be rounded.

##### **Remarks**

- Regardless of the sign of the number a value is rounded up when adjusted away from zero. If the number is an even integer no rounding occurs.

#### [4.1.4.6.6.82 ERROR.TYPE](#)

The Error.Type function returns an integer for the given error value that denotes the type of the given error.

##### **Syntax:**

The syntax of the ERROR.TYPE function is

**= ERROR.TYPE(value)**

The given value is required.

Here is the return value of function:

Given Value	Return value of function
#NULL!	1
#DIV/0!	2
#VALUE!	3
#REF!	4
#NAME?	5
#NUM!	6
#N/A	7
#GETTING_DATA	8
Anything else	#N/A

**Example:**

FORMULA	RESULT
= ERROR.TYPE(#NULL!)	1
= ERROR.TYPE(even)	#NA

#### *4.1.4.6.6.83 EXP*

Returns e raised to the power of the given number.

**Syntax**

**EXP(number)**, where:

**number** is the exponent applied to the base e.

#### *4.1.4.6.6.84 EXPONDIST*

Returns the exponential distribution.

**Syntax**

**EXPONDIST(x, lambda, cumulative)**, where:

**x** is the value of the function.

**lambda** is the parameter value.

**cumulative** is a logical value that indicates which form of the exponential function is to be provided. If cumulative is True, EXPONDIST returns the cumulative distribution function; if False, it returns the probability density function.

### Remarks

The equation for the probability density function is,

$$f(x;\lambda) = \lambda e^{-\lambda x}$$

- The equation for the cumulative distribution function is,

$$F(x;\lambda) = 1 - e^{-\lambda x}$$

#NUM! - occurs if x is less than zero.

#VALUE! - occurs if x is non-numeric.

#VALUE! - occurs if lambda is non-numeric.

#NUM!- occurs if lambda is equal to or less than zero.

#### [4.1.4.6.85 FACT](#)

Returns the factorial of a number. The factorial of a number is the product of all positive integers <= the given number.

### Syntax

**FACT(number)**, where:

**number** is the non-negative number for which you want the factorial of. If the number is not an integer, it is truncated.

#### [4.1.4.6.86 FACTDOUBLE](#)

The FACTDOUBLE function returns the double factorial of a given value. The given value must be an integer value.

### Syntax

The syntax of the FACTDOUBLE function is

**= FACTDOUBLE (number).**

number – Required.

### Remarks:

#NUM! - If the number is less than zero (0).

#VALUE! - Occurs if any of the given arguments are non-numeric

### Example:

FORMULA	RESULT
= FACTDOUBLE (6)	48
= FACTDOUBLE (-2)	#NUM!

#### [4.1.4.6.6.87 False](#)

The False function returns a logical value when the given string value is false.

### Syntax:

**False(stringvalue)**

where:

**stringvalue** is to provide any text value or empty string.

#### [4.1.4.6.6.88 FDIST](#)

Returns the F probability distribution.

### Syntax

**FDIST(x, degrees\_freedom1, degrees\_freedom2),** where:

**x** is the value at which to evaluate the function.

**degrees\_freedom1** is the numerator degrees of freedom.

**degrees\_freedom2** is the denominator degrees of freedom.

## Remarks

- All arguments must be numeric.
- X must be  $\geq 0$ .
- Both **degrees\_freedom1** and **degrees\_freedom2** must be  $\geq 1$  and  $< 10^{10}$ .
- FDIST is calculated as  $FDIST = P(F > x)$ , where F is a random variable that has an F distribution with **degrees\_freedom1** and **degrees\_freedom2** degrees of freedom.

### [4.1.4.6.6.89 Finv](#)

The Finv function returns the inverse of the F probability distribution. If  $p = FDIST(x, \dots)$ , then  $FINV(p, \dots) = x$ .

Using F distribution, you can compare the degree of variability for two data sets.

#### Syntax:

**FINV(probability,deg\_freedom1,deg\_freedom2)**

The FINV function syntax has the following three arguments (Argument is a value that provides information to an action, an event, a method, a property, a function, or a procedure):

**Probability** is a probability associated with the F cumulative distribution.

**Deg\_freedom1** is the numerator degrees of freedom.

**Deg\_freedom2** is the denominator degrees of freedom.

#### Remarks:

#NUM! - occurs if probability is equal to or less than zero.

#NUM! - occurs if probability is equal to or greater than 1.

#VALUE! - occurs if probability is non-numeric.

#NUM! - occurs if **deg\_freedom1** is less than one.

#VALUE! - occurs if **deg\_freedom1** is non-numeric.

#NUM! - occurs if **deg\_freedom2** is less than one.

#VALUE! - occurs if **deg\_freedom2** is non-numeric.

#### *4.1.4.6.6.90 F.Inv.Rt*

The F.Inv.Rt function returns the inverse of the F probability distribution.

##### **Syntax:**

**F.INV.RT(probability,deg\_freedom1,deg\_freedom2)** where:

- probability is a probability that corresponds to the normal distribution.
- deg\_freedom1 is the numerator degrees of freedom.
- deg\_freedom2 is the denominator degrees of freedom.

##### **Remarks:**

#NUM! - occurs if probability is equal to or less than zero.

#NUM! - occurs if probability is equal to or greater than 1.

#VALUE! - occurs if probability is non-numeric.

#NUM! - occurs if deg\_freedom1 is less than one.

#VALUE! - occurs if deg\_freedom1 is non-numeric.

#NUM! - occurs if deg\_freedom2 is less than one.

#VALUE! - occurs if deg\_freedom2 is non-numeric.

#### *4.1.4.6.6.91 FISHER*

Returns the Fisher transformation at x. This transformation produces a function that is normally distributed rather than skewed.

##### **Syntax**

**FISHER(x)**, where:

x is a numeric value for which you want the transformation.

##### **Remarks**

- X must be > -1 and < 1.
- The equation for the Fisher transformation is,

$$z' = \frac{1}{2} \ln\left(\frac{1+x}{1-x}\right)$$

#### [4.1.4.6.6.92 FISHERINV](#)

Returns the inverse of the Fisher transformation. If  $y = \text{FISHER}(x)$ , then  $\text{FISHERINV}(y) = x$ .

#### Syntax

**FISHERINV(y)**, where:

**y** is the value for which you want to perform the inverse of the transformation.

#### Remarks

- The equation for the inverse of the Fisher transformation is,

$$x = \frac{e^{2y}-1}{e^{2y}+1}$$

#### [4.1.4.6.6.93 Fixed](#)

The **Fixed** function rounds off the given value to a specified number of decimal places and returns the value in text format.

#### Syntax:

**Fixed (number, decimal\_places, no\_commas)**

where:

**number** is the number, which you want to round off.

**decimal\_places** is the number of decimal places you want to display in the result.

**no\_commas** is a logical value. It will display commas when it is set to FALSE and does not display commas when it is set to TRUE.

#### [4.1.4.6.6.94 FLOOR](#)

Rounds off the given number down, toward zero, to the nearest multiple of significance.

#### Syntax

**FLOOR(number, significance)**, where:

number is the numeric value that you want to round off.

significance is the multiple to which you want to round the number off.

### Remarks

- Number and significance must have the same sign.
- Regardless of the sign of the number, a value is rounded down when adjusted away from zero. If a number is an exact multiple of significance, no rounding occurs.

#### 4.1.4.6.6.95 FORECAST

Calculates a future value by using existing values using a linear regression. The predicted value is a y-value for a given x-value.

### Syntax

**FORECAST(x, known\_ys, known\_xs)**, where:

**x** is the data point for which you want to predict a value.

**known\_ys** is the dependent array or range of data.

**known\_xs** is the independent array or range of data.

### Remarks

- The equation for FORECAST is  $a+bx$ ,

where:

$$a = \bar{y} - b \bar{x}$$

and

$$b = \frac{\sum(x - \bar{x})(y - \bar{y})}{\sum(x - \bar{x})^2}$$

and

x-bar and y-bar are the sample means AVERAGE(known\_xs) and AVERAGE(known\_ys).

#### [4.1.4.6.96 FV](#)

Returns the future value of an investment based on periodic, constant payments and interest rate.

#### Syntax

**FV(rate, nper, pmt, pv, type)**, where:

**rate** is the interest rate per period.

**nper** is the total number of payment periods in an annuity.

**pmt** is the payment made each period; it cannot change over the life of the annuity. Typically, pmt contains principal and interest but, no other fees or taxes. If pmt is omitted, you must include the pv argument.

**pv** is the present value or lump-sum amount that a series of future payments is worth right now. If pv is omitted, it is assumed to be 0 (zero), and you must include the pmt argument.

**type** is the number 0 or 1 and indicates when payments are due. If type is omitted it is assumed to be 0. If type equals:

- 0 - Payments are due at the end of the period.
- 1 - Payments are due at the beginning of the period.



**Note:** For a more complete description of the arguments in FV, see PV.

#### Remarks

- Make sure that you are consistent about the units you use for specifying rate and nper. If you make monthly payments for a four-year loan at 12 percent annual interest, use 12%/12 for rate and 4\*12 for nper. If you make annual payments on the same loan, use 12% for rate and 4 for nper.
- For all the arguments, cash you pay out, such as deposits to savings, is represented by negative numbers; cash you receive, such as dividend checks, is represented by positive numbers.

#### [4.1.4.6.97 Fvschedule](#)

After applying a series of compound interest rates, the Fvschedule method returns the future value of an initial principle.

**Syntax:**

**FVSchedule(arg1,arg2)** where:

- Arg1 is the present value.
- Arg2 is an array of interest rates to apply.

**Remarks:**

#VALUE! – occurs any other than numbers or blank cells.

**4.1.4.6.6.98 GAMMADIST**

Returns the gamma distribution.

**Syntax**

**GAMMADIST(x, alpha, beta, cumulative)**, where:

**x** is the value at which you want to evaluate the distribution.

**alpha** is a parameter to the distribution.

**beta** is a parameter to the distribution. If beta = 1, GAMMADIST returns the standard gamma distribution.

**cumulative** is a logical value that determines the form of the function. If cumulative is True, GAMMADIST returns the cumulative distribution function; if False, it returns the probability density function.

**Remarks**

- X must be  $\geq 0$ .
- Alpha and beta must be  $> 0$ .
- The equation for the gamma probability density function is,

$$f(x;\alpha,\beta) = \frac{1}{\beta^\alpha \Gamma(\alpha)} x^{\alpha-1} e^{-\frac{x}{\beta}}$$

The standard gamma probability density function is,

$$f(x;\alpha) = \frac{x^{\alpha-1} e^{-x}}{\Gamma(\alpha)}$$

- When alpha = 1, GAMMADIST returns the exponential distribution with,

$$\lambda = \frac{1}{\beta}$$

#NUM! - occurs if x is less than zero.

#VALUE! - occurs if x is non-numeric.

#VALUE! - occurs if alpha is non-numeric.

#NUM!- occurs if alpha is equal to or less than zero.

#VALUE! - occurs if beta is non-numeric.

#NUM! - occurs if beta is equal to or less than zero.

#### 4.1.4.6.6.99 GAMMALN

Returns the natural logarithm of the gamma function,  $\tilde{\Lambda}(x)$ .

#### Syntax

**GAMMALN(x)**, where:

x is the value for which you want to calculate GAMMALN.

#### Remarks

- x must be positive.
- GAMMALN is calculated as follows,

$$GAMMALN = LN(\Gamma(x))$$

where:

$$\Gamma(x) = \int_0^{\infty} e^{-u} u^{x-1} du$$

#### [4.1.4.6.6.100 GAMMAINV](#)

Returns the inverse of the gamma cumulative distribution. If  $p = \text{GAMMADIST}(x, \dots)$ , then  $\text{GAMMAINV}(p, \dots) = x$ .

#### Syntax

**GAMMAINV(probability, alpha, beta)**, where:

**probability** is the probability associated with the gamma distribution.

**alpha** is a parameter to the distribution.

**beta** is a parameter to the distribution.

#### Remarks

- Probability must be  $\geq 0$  and  $\leq 1$ .
- Alpha and beta must be positive.

Given a value for probability, GAMMAINV seeks value  $x$  such that  $\text{GAMMADIST}(x, \text{alpha}, \text{beta}, \text{True}) = \text{probability}$ . Thus, precision of GAMMAINV depends on the precision of GAMMADIST. GAMMAINV uses an iterative search technique.

#### [4.1.4.6.6.101 GCD](#)

The GCD function returns the greatest common divisor of two or more given values. The values must be a numeric value.

#### Syntax

The syntax of the GCD function is

**= GCD (number1, number2, ...)**

Number1 – Required.

If any value is not an integer, then it will be rounded down.

#### Remarks:

#NUM! - If the number is less than zero (0).

#VALUE! - Occurs if any of the given arguments are non-numeric.

#### Example:

FORMULA	RESULT
= GCD (5,3,2)	1
= GCD (-2)	#NUM!

#### 4.1.4.6.6.102 GEOMEAN

Returns the geometric mean of an array or range of positive data.

#### Syntax

**GEOMEAN(number1, number2, ...)**, where:

**number1, number2, ...** are arguments for which you want to calculate the mean.

#### Remarks

- The arguments must be either numbers or names, arrays or references that contain numbers.
- All values must be positive.
- The equation for the geometric mean is,

$$GM = \left( \prod_{i=1}^n y_i \right)^{\frac{1}{n}}$$

#### 4.1.4.6.6.103 Growth

This feature enables you to calculate predicted exponential growth using existing data. This calculates and returns an array of values used for the regression analysis. Growth enables you to perform a regression analysis.

Table 4: Method Table

Method	Description	Parameters	Type	Return Type	Reference links
Growth()	Calculates the Growth for an array of cells.	Known y's, Known x's, new_x's	Method	String	N/A

The following is the formula to calculate Growth for an array of cells in a column:

**[Syntax]**

=GROWTH(known\_y's, [known\_x's], [new\_x's],

**Known\_y's:** A set of y-values you already know in a relationship, where  $y = b \cdot m^x$ .

**Known\_x's:** An optional set of x-values that you may already know in the relationship, where  $y = b \cdot m^x$ .

**New\_x's:** New x-values for which you want GROWTH to return corresponding y-values.

**[Code]**

=Growth(B2:B7,A2:A7,C6:C7)

*4.1.4.6.6.104 HARMEAN*

Returns the harmonic mean of a data set. The harmonic mean is the reciprocal of the arithmetic mean of reciprocals.

**Syntax**

**HARMEAN(number1, number2, ...),** where:

**number1, number2, ...** are arguments for which you want to calculate the mean.

**Remarks**

- The arguments must be either numbers or names, arrays or references that contain numbers.
- All data values must be positive.
- The equation for the harmonic mean is,

$$H = \frac{n}{\sum \frac{1}{y_i}}$$

#### 4.1.4.6.105 HLOOKUP

Searches for a value in the top row of the array of values and then returns a value in the same column from a row you specify in the array. Use HLOOKUP when your comparison values are located in a row across the top of a table of data and you want to look down a specified number of rows. Use VLOOKUP when your comparison values are located in a column to the left of the data you want to find.

#### Syntax

**HLOOKUP(lookup\_value, table\_array, row\_index\_num, range\_lookup)**, where:

**lookup\_value** is the value to be found in the first row of the table. Lookup\_value can be a value, a reference or a text string.

**table\_array** is a table of information in which data is looked up. Use a reference to a range or a range name.

**row\_index\_num** is the row number in table\_array from which, the matching value will be returned. A row\_index\_num of 1 returns the first row value in table\_array, a row\_index\_num of 2 returns the second row value in table\_array and so on.

**range\_lookup** is a logical value that specifies whether you want HLOOKUP to find an exact match or an approximate match. If True or omitted, an approximate match is returned. In other words, if an exact match is not found, the next largest value that is less than the lookup\_value is returned. (This requires your lookup values to be sorted.) If False, HLOOKUP will find an exact match.

#### 4.1.4.6.106 HOUR

Returns the hour of a time value. The hour is given as an integer, ranging from 0 (12:00 A.M.) to 23 (11:00 P.M.).

#### Syntax

**HOUR(serial\_number)**, where:

**serial\_number** is the time that contains the hour you want to find. Times may be entered as text strings within quotation marks (for example, "6:00 PM"), as decimal numbers (for example, 0.75, which represents 6:00 PM), or as results of other formulas or functions (for example, TIMEVALUE("6:00 PM")).

#### [4.1.4.6.6.107 HYPGEOMDIST](#)

Returns the hypergeometric distribution. HYPGEOMDIST returns the probability of a given number of sample successes, given the sample size, population successes and population size.

#### Syntax

**HYPGEOMDIST(sample\_s, number\_sample, population\_s, number\_population)**, where:

**sample\_s** is the number of successes in the sample.

**number\_sample** is the size of the sample.

**population\_s** is the number of successes in the population.

**number\_population** is the population size.

#### Remarks

- All arguments are truncated to integers.
- **sample\_s** must be  $\geq 0$  less than both **number\_sample** and **population\_s**.
- **number\_sample** must be  $\geq 0$  and  $< \text{number\_population}$ .
- **population\_s** must be  $\geq 0$  and  $< \text{number\_population}$ .
- **number\_population** must be  $\geq 0$ .
- The equation for the hypergeometric distribution is,

$$P(X=x) = h(x; n, M, N) = \frac{\binom{M}{x} \binom{N-M}{n-x}}{\binom{N}{n}}$$

where:

**x** = **sample\_s**

**n** = **number\_sample**

M = population\_s

N = number\_population

#### [4.1.4.6.6.108 IEEERemainder](#)

IEEERemainder function returns the remainder operation on two arguments.

##### **Syntax:**

**IEEERemainder(d1, d2)** where:

- d1 is the divisor.
- d2 is the dividend.

#### [4.1.4.6.6.109 IF](#)

Returns one value if a condition you specify evaluates to True and another value if it evaluates to False.

Use IF to conduct conditional tests on values and formulas.

##### **Syntax**

**IF(logical\_test, value\_if\_true, value\_if\_false)**, where:

logical\_test is any value or expression that can be evaluated to True or False.

value\_if\_true is the value that is returned if a logical\_test is True.

value\_if\_false is the value that is returned if a logical\_test is False.

##### **Remarks**

- Countif and Sumif are additional methods that provide conditional calculations.

#### [4.1.4.6.6.110 IFERROR](#)

The IFERROR function tests if an initial given value (or expression) returns an error, and if so, this function returns a second given argument. Otherwise, the function returns the initial tested value.

##### **Syntax**

The syntax of the IFERROR function is

**= IFERROR (value, value\_error)**

value – Required. This is a value to check the error.

value\_error – Required. This value will be returned if the value has an error.

**Remarks:**

If the value\_error is an empty cell, then the function takes the error value as empty string.

**Example:**

FORMULA	RESULT
= IFERROR (200/55, “ERROR in DIVISION”)	3
= IFERROR (200/0, “ERROR in DIVISION”)	ERROR in DIVISION

[\*\*4.1.4.6.6.111 IFNA\*\*](#)

The IFNA function returns the value specified if the formula returns the #N/A error value; otherwise, it returns the result of the given formula.

**Syntax**

=IFNA (Formula\_value, value\_if\_na)

Formula\_value: Required. The argument that is checked for the #N/A error value.

value\_if\_na: Required. The value returned if the formula evaluates to the #N/A error value.

**Example**

FORMULA	RESULT
=IFNA("#N/A","Incorrect")	Incorrect
=IFNA(1602,"incorrect")	1602

[\*\*4.1.4.6.6.112 Index\*\*](#)

The Index function returns the exact cell value from the provided row index and column index from a specific range of cells.

**Syntax:**

**Index(range,row,col)**

where:

**range** is a string to mention the specific range.

**row** is the integer that indicates the specific row index.

**col** is the integer that indicates the specific column index.

#### [4.1.4.6.6.113 Indirect](#)

The Indirect function returns the reference as a string instead of providing the content or range within the cell.

##### **Syntax:**

**Indirect(content)**

where:

**content** is the string that provides the textual representation of the cell reference.

#### [4.1.4.6.6.114 INT](#)

Rounds a number down to the nearest integer.

##### **Syntax**

**INT(number)**, where:

**number** is the real number that you want to round down to an integer.

#### [4.1.4.6.6.115 INTRATE](#)

The Inrate function returns the interest rate for a fully invested security.

##### **Syntax:**

**INTRATE(settlement, maturity, investment, redemption, basis )** where:

- settlement is the security's settlement date.
- maturity is the security's maturity date.
- investment is the amount invested in the security.
- redemption is the amount to be received at maturity.
- basis is the kind of day count basis to use.

##### **Remarks:**

#NUM! - occurs if settlement is greater than or equal to maturity.

#VALUE! - occurs if a valid date is not given for the settlement.

#VALUE! - occurs if a valid date is not given for maturity.

#NUM! - occurs if investment is less than or equal to zero.

#NUM! - occurs if redemption is less than or equal to zero.

#### [4.1.4.6.6.116 INTERCEPT](#)

Calculates the point at which, the least squares fit line will intersect the y-axis.

#### Syntax

**INTERCEPT(known\_y's, known\_x's)**, where:

**known\_y's** is the dependent set of observations or data.

**known\_x's** is the independent set of observations or data.

#### Remarks

- The equation for the intercept of the regression line, a, is,

$$a = \bar{y} - b \bar{x}$$

where the slope, b, is calculated as:

$$b = \frac{\sum(x - \bar{x})(y - \bar{y})}{\sum(x - \bar{x})^2}$$

and x-bar and y-bar are the sample means AVERAGE(known\_x's) and AVERAGE(known\_y's).

#### [4.1.4.6.6.117 IPMT](#)

Returns the interest payment for a given period for an investment based on periodic, constant payments and a constant interest rate.

## Syntax

**IPMT(rate, per, nper, pv, fv, type)**, where:

**rate** is the interest rate per period.

**per** is the period for which you want to find the interest and must be in the range 1 to nper.

**nper** is the total number of payment periods in an annuity.

**pv** is the present value or the lump-sum amount that a series of future payments is worth right now.

**fv** is the future value or a cash balance that you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0 (the future value of a loan, for example, is 0).

**type** is the number 0 or 1 and indicates when payments are due. If type is omitted, it is assumed to be 0. If type = 0, payments are made at the end of the period. If type is 1, payments are made at the beginning of the period.

## Remarks

- Make sure that you are consistent about the units you use for specifying rate and nper. If you make monthly payments on a four-year loan at 12 percent annual interest, use 12%/12 for rate and 4\*12 for nper. If you make annual payments on the same loan, use 12% for rate and 4 for nper.

### [4.1.4.6.6.118 IRR](#)

Returns the internal rate of return for a series of cash flows represented by the numbers in values. The cash flows must occur at regular intervals such as monthly or annually.

## Syntax

**IRR(values, guess)**, where:

**values** is an array or a reference to cells that contain numbers for which you want to calculate the internal rate of return.

- Values must contain at least one positive value and one negative value to calculate the internal rate of return.
- IRR uses the order of values to interpret the order of cash flows. Be sure to enter your payment and income values in the sequence you want.

- **guess** is a number that you guess is close to the result of IRR.
- An iterative technique is used for calculating IRR.
- In most cases, you do not need to provide a guess for the IRR calculation. If a guess is omitted, it is assumed to be 0.1 (10 percent).

#### [4.1.4.6.6.119 IsErr](#)

The IsErr function checks whether a value has an error.

##### **Syntax:**

**IsErr(value)**

where:

**value** is the value that you want to test for error. If the value has an error (except #N/A), this function will return TRUE else returns FALSE.

#### [4.1.4.6.6.120 ISERROR](#)

Returns True if the value is a string that starts with a #.

##### **Syntax**

**ISERROR(value)**, where:

**value** is the value that is to be tested.

#### [4.1.4.6.6.121 IsNA](#)

The IsNA function returns a boolean value after determining that the provided value is a #NA error value.

##### **Syntax:**

**IsNA(value)**

where:

**value** is the value, which the function will test.

#### [4.1.4.6.6.122 ISNUMBER](#)

Returns True if the value parses as a numeric value.

## Syntax

**ISNUMBER(value)**, where:

**value** is the value that is to be tested.

### [4.1.4.6.6.123 ISPMT](#)

Calculates the interest paid during a specific period of an investment.

## Syntax

**ISPMT(rate, per, nper, pv)**, where:

**rate** is the interest rate for the investment.

**per** is the period for which you want to find the interest and must be between 1 and nper.

**nper** is the total number of payment periods for the investment.

**pv** is the present value of the investment. For a loan, pv is the loan amount.

## Remarks

- Make sure that you are consistent about the units you use for specifying rate and nper. If you make monthly payments on a four-year loan at an annual interest rate of 12 percent, use 12%/12 for rate and 4\*12 for nper. If you make annual payments on the same loan, use 12% for rate and 4 for nper.

### [4.1.4.6.6.124 ISREF](#)

The ISREF function returns the logical value TRUE if the given value is a reference value; otherwise, the function returns FALSE.

## Syntax

=ISREF(given\_value)

**given\_value:** Required. The value that is to be tested. The value argument can be a blank (empty cell), error, logical value, text, number, or reference value, or a name referring to any of these.

**Example**

FORMULA	RESULT
=ISREF("Region1")	FALSE
=ISREF(=ISLOGICAL(TRUE))	TRUE

[\*\*4.1.4.6.6.125 IsText\*\*](#)

The IsText function returns a boolean value after determining that the provided value is a string.

**Syntax:**

**IsText(text)**

where:

**text** is the value you want to check whether it is a string or not.

[\*\*4.1.4.6.6.126 IsNonText\*\*](#)

The IsNonText function returns the boolean value after determining that the provided value is not a string.

**Syntax:**

**IsNonText(text)**

where:

**text** is the value you want to check whether it is a string or not.

[\*\*4.1.4.6.6.127 KURT\*\*](#)

Returns the kurtosis of a data set. Kurtosis characterizes the relative peakedness or flatness of a distribution compared with the normal distribution. Positive kurtosis indicates a relatively peaked distribution. Negative kurtosis indicates a relatively flat distribution.

**Syntax**

**KURT(number1, number2, ...), where:**

**number1, number2, ...** are arguments for which you want to calculate kurtosis. You can also use a single array or a reference to an array instead of arguments separated by commas.

## Remarks

- The arguments must be either numbers or names, arrays or references that contain numbers.
- If an array or reference argument contains text, logical values or empty cells, those values are ignored; however, cells with the value zero are included.
- If there are fewer than four data points or if the standard deviation of the sample equals zero, KURT returns the #DIV/0! error value.
- Kurtosis is defined as:

$$\left\{ \frac{n(n+1)}{(n+1)(n+2)(n+3)} \sum \left( \frac{x_i - \bar{x}}{s} \right)^4 \right\} - \frac{3(n+1)^2}{(n+2)(n+3)}$$

where s is the sample standard deviation.

### 4.1.4.6.128 LARGE

Returns the k-th largest value in a data set.

## Syntax

**LARGE(array, k)**, where:

**array** is the array or range of data for which you want to determine the k-th largest value.

**k** is the position (from the largest) in the array or cell range of data to return.

## Remarks

- If n is the number of data points in a range, then LARGE(array,1) returns the largest value, and LARGE(array,n) returns the smallest value.

### 4.1.4.6.129 LCM

The LCM function returns the least common multiple of two or more given values. The values must be numeric values.

## Syntax

The syntax of the LCM function is

**= LCM (number1, number2, ...)**

number1 – Required.

If any value is not an integer, then it will be rounded down.

**Remarks:**

#NUM! - If the number is less than zero (0).

#VALUE! - Occurs if any of the given arguments are non-numeric.

**Example:**

FORMULA	RESULT
= LCM (5,2)	10
= LCM (-2)	#NUM!

[4.1.4.6.6.130 LEFT](#)

LEFT returns the first character or characters in a text string, based on the number of characters you specify.

**Syntax**

**LEFT(text, num\_chars)**, where:

**text** is the text string that contains the characters which you want to extract.

**num\_chars** specifies the number of characters which you want **LEFT** to extract.

**Remarks**

- Num\_chars must be greater than or equal to zero.
- If num\_chars is greater than the length of text, LEFT returns all the text.
- If num\_chars is omitted, it is assumed to be 1.

[4.1.4.6.6.131 LN](#)

Returns the natural logarithm of a number. Natural logarithms are based on the constant e (2.718281828459...).

## Syntax

**LN(number)**, where:

**number** is the positive real number for which you want the natural logarithm.

## Remarks

- LN is the inverse of the EXP function.

### [4.1.4.6.6.132 LEN](#)

LEN returns the length of a text string, including spaces.

## Syntax

**Len(text)**, where:

**text** is the text string whose length is to be determined.

### [4.1.4.6.6.133 OG](#)

Returns the logarithm of a number to the base that you specify.

## Syntax

**LOG(number, base)**, where:

**number** is the positive real number for which you want the logarithm.

**base** is the base of the logarithm. If base is omitted, it is assumed to be 10.

#### [4.1.4.6.6.134 LOG10](#)

Returns the base-10 logarithm of a number.

#### Syntax

**LOG10(number)**, where:

**number** is the positive real number for which you want the base-10 logarithm.

#### [4.1.4.6.6.135 IsBlank](#)

The IsBlank function checks for blank or null values.

#### Syntax:

**IsBlank(value)**

where:

**value** is the value that you want to test. If the value is blank, this function will return TRUE. If the value is not blank, the function will return FALSE.

#### [4.1.4.6.6.136 ISEVEN](#)

The ISEVEN function returns TRUE if given number is an even number, and returns FALSE if the given number is odd.

#### Syntax:

The syntax of the ISEVEN function is

**=ISEVEN (value)**

The given value must be a numeric value. If it is non-integer value, the value is rounded down.

#### Remarks:

If the given value is nonnumeric, the ISEVEN function returns the '#VALUE!' error value.

#### Example:

FORMULA	RESULT
=ISEVEN(-1)	FALSE
=ISEVEN(2.5)	TRUE

=ISEVEN(5)	FALSE
------------	-------

#### 4.1.4.6.6.137 ISODD

The ISODD function returns TRUE if the given number is an odd number, and returns FALSE if the given number is even.

**Syntax:**

The syntax of the ISODD function is

**=ISODD (value)**

The given value must be a numeric value. If it is a non-integer value, the value is rounded down.

**Remarks:**

If the given value is nonnumeric, ISODD function returns the '#VALUE!' error value.

**Example:**

FORMULA	RESULT
=ISODD(-1)	TRUE
=ISODD(2.5)	FALSE
=ISODD(5)	TRUE

#### 4.1.4.6.6.138 Logest

This feature enables you to calculate predicted exponential growth using existing data. This calculates and returns an array of values used for the regression analysis. Logest calculates and returns an array of values that is used in regression analysis.

Table 5: Method Table

Method	Description	Parameters	Type	Return Type	Reference links
Logest()	Calculates Logest for an array of cells.	known_y's, known_x's, const, stats	<b>Method</b>	String	N/A

The following is the formula to calculate Logest for an array of cells in a column:

**[Syntax]**

=LOGEST(known\_y's, [known\_x's], [const], [stats])

Known\_y's : A set of y-values you already know in a relationship, where  $y = b \cdot m^x$ .

Known\_x's : An optional set of x-values that you may already know in a relationship, where  $y = b \cdot m^x$ .

Const : A logical value specifying whether to force the constant b to equal 1.

Stats : A logical value specifying whether to return additional regression statistics.

**[Code]**

= Logest(B2:B7,A2:A7,TRUE,FALSE)

*4.1.4.6.6.139 IsLogical*

The IsLogical function checks whether a value is a logical value and returns TRUE or FALSE.

**Syntax:**

**IsLogical(value)**

where:

**value** is the value that you want to check whether it is logical. If the value is TRUE or FALSE, this function will return TRUE. Otherwise, it will return FALSE.

*4.1.4.6.6.140 LOGINV*

Returns the inverse of the lognormal cumulative distribution function of x, where  $\ln(x)$  is normally distributed with parameters mean and standard\_dev. If  $p = LOGNORMDIST(x, \dots)$ , then  $LOGINV(p, \dots) = x$ .

## Syntax

**LOGINV(probability, mean, standard\_dev)**, where:

**probability** is the probability associated with the lognormal distribution.

**mean** is the mean of  $\ln(x)$ .

**standard\_dev** is the standard deviation of  $\ln(x)$ .

## Remarks

- Probability must be  $\geq 0$  and  $< 1$ .
- Standard\_dev must be positive.
- The inverse of the lognormal distribution function is,

$$\text{LOGINV}(p, \mu, \sigma) = e^{\{\mu + \sigma \times [\text{NORMSINV}(p)]\}}$$

### 4.1.4.6.6.141 LOGNORMDIST

Returns the cumulative lognormal distribution of  $x$ , where  $\ln(x)$  is normally distributed with parameters **mean** and **standard\_dev**.

## Syntax

**LOGNORMDIST(x, mean, standard\_dev)**, where:

**x** is the value at which the function can be evaluated.

**mean** is the mean of  $\ln(x)$ .

**standard\_dev** is the standard deviation of  $\ln(x)$ .

## Remarks

- Both **x** and **standard\_dev** must be positive.
- The equation for the lognormal cumulative distribution function is,

$$LOGNORMDIST(x, \mu, \sigma) = NORMSDIST\left(\frac{\ln(x) - \mu}{\sigma}\right)$$

#### 4.1.4.6.6.142 LOOKUP

The LOOKUP function returns a value either from a one-row or one-column range, or from an array. The LOOKUP function has two syntax forms: vector and array.

**Vector Form:** The vector form of LOOKUP looks in a one-row or one-column range for a value, and then returns a value from the same position in a second one-row or one-column range.

#### Syntax

=LOOKUP(lookup\_value, lookup\_vector, result\_vector)

**Array form:** The array form of LOOKUP looks in the first row or column of an array for the specified value, and then returns a value from the same position in the last row or column of the array.

#### Syntax

=LOOKUP(lookup\_value, array)

#### Notes

- If the LOOKUP function can't find the lookup\_value, the function matches the largest value in lookup\_vector that is less than or equal to lookup\_value.
- If lookup\_value is smaller than the smallest value in lookup\_vector, LOOKUP returns the #N/A error value.

#### Example

Input Table

	A	B	C
1	Earning	Tax	other
2	100000	3000	100
3	200000	6000	200
4	300000	7500	300
5	400000	9000	500

FORMULA	RESULT
=LOOKUP(6000,B2:B5,C2:C5)	200

=LOOKUP("C", {"a","b","c","d";1,2,3,4})	3
---	---

#### [4.1.4.6.6.143 Lower](#)

The Lower function converts all characters in a specified text string to lowercase. Characters in the string that are not text are not changed.

##### Syntax:

**Lower(text)**

where:

**text** is the string you want to convert to lowercase.

#### [4.1.4.6.6.144 MAX](#)

Returns the largest value in a set of values.

##### Syntax

**MAX(number1, number2, ...)**, where:

**number1, number2, ...** are numbers for which you want to find the maximum value.

#### [4.1.4.6.6.145 MAXA](#)

Returns the largest value in a list of arguments. Text and logical values such as True and False are compared as well as numbers.

##### Syntax

**MAXA(value1, value2, ...)**, where:

**value1, value2, ...** are values for which you want to find the largest value.

## Remarks

- You can specify arguments that are numbers, empty cells, logical values or text representations of numbers. Arguments that are error values cause errors. If the calculation does not include text or logical values, use the MAX worksheet function instead.
- If an argument is an array or reference, only values in that array or reference are used. Empty cells and text values in the array or reference are ignored.
- Arguments that contain True evaluate as 1; arguments that contain text or False evaluate as 0 (zero).
- If the arguments contain no values, MAXA returns 0 (zero).

### [4.1.4.6.146 MDETERM](#)

The MDETERM function retrieves the matrix determinant of an array.

#### Syntax:

**MDETERM(array)** where:

- array is a numeric array with an equal number of rows and columns.

#### Remarks:

#VALUE! - occurs if any cell in the array is empty or has text in it.

#VALUE! - occurs if the array does not have an equal number of rows and columns.

### [4.1.4.6.147 MEDIAN](#)

Returns the median of the given numbers. The median is the number in the middle of a set of numbers; that is, half the numbers have values that are greater than the median and half have values that are less.

#### Syntax

**MEDIAN(number1, number2, ...)**, where:

**number1, number2, ...** are numbers for which you want the median.

#### Remarks

- If there is an even number of numbers in the set, then MEDIAN calculates the average of the two numbers in the middle.

#### [4.1.4.6.6.148 MID](#)

MID returns a text segment of a character string. The parameters specify the starting position and the number of characters.

#### Syntax

**MID(text, start\_position, num\_chars)**, where:

**text** is the text containing the characters to extract.

**start** is the position of the first character in the text to extract.

**number** specifies the number of characters in the part of the text.

#### [4.1.4.6.6.149 MIN](#)

Returns the smallest number in a set of values.

#### Syntax

**MIN(number1, number2, ...)**, where:

**number1, number2, ...** are numbers for which you want to find the minimum value.

#### Remarks

- If an argument is an array or reference, only numbers in that array or reference are used. Empty cells, logical values or text in the array or reference are ignored. If logical values and text should not be ignored, use MINA.

#### [4.1.4.6.6.150 MINA](#)

Returns the smallest value in the list of arguments. Text and logical values such as True and False are compared as well as numbers.

#### Syntax

**MINA(value1,value2, ...), where:**

**value1, value2, ...** are values for which you want to find the smallest value.

### Remarks

- Arguments that contain True evaluate as 1; arguments that contain text or False evaluate as 0 (zero).

#### [4.1.4.6.151 MINUTE](#)

Returns the minutes of a time value. The minute is given as an integer, ranging from 0 to 59.

### Syntax

**MINUTE(serial\_number), where:**

**serial\_number** is the time that contains the minute you want to find. Times may be entered as text strings within quotation marks (for example, "6:00 PM"), as decimal numbers (for example, 0.75, which represents 6:00 PM), or as results of other formulas or functions (for example, TIMEVALUE("6:00 PM")).

### Remarks

- Time values are a portion of a date value and are represented by a decimal number (for example, 12:00 PM is represented as 0.5).

#### [4.1.4.6.152 MINVERSE](#)

The MINVERSE function retrieves the inverse matrix for the matrix stored in an array.

### Syntax:

**MINVERSE(array) where:**

- array is an numeric array with an equal number of rows and columns.

### Remarks:

#VALUE! - occurs if any cells in the array are empty or contain a string.

#VALUE! - occurs if the array does not have an equal number of rows and columns.

#### 4.1.4.6.6.153 MIRR

Returns the modified internal rate of return for a series of periodic cash flows.

#### Syntax

**MIRR(values, finance\_rate, reinvest\_rate)**, where:

**values** is an array or a reference to cells that contain numbers. These numbers represent a series of payments (negative values) and income (positive values) occurring at regular periods.

- Values must contain at least one positive value and one negative value to calculate the modified internal rate of return.
- **finance\_rate** is the interest rate you pay on the money used in the cash flows.
- **reinvest\_rate** is the interest rate you receive on the cash flows as you reinvest them.

#### Remarks

- MIRR uses the order of values to interpret the order of cash flows. Be sure to enter your payment and income values in the sequence you want and with the correct signs (positive values for cash received, negative values for cash paid).
- If n is the number of cash flows in values, frate is the **finance\_rate**, and rrate is the **reinvest\_rate**, then the formula for MIRR is,

$$\left( \frac{-NPV(rrate, values[positive]) * (1+rrate)^n}{NPV(frate, values[negative]) * (1+frate)} \right)^{\frac{1}{n-1}} - 1$$

#### 4.1.4.6.6.154 MMULT

The MMULT function returns the matrix product of two arrays. Both the arrays should have same number of columns and same number of rows.

#### Syntax:

**MMULT(a1,a2)** where:

- a1,a2 are the arrays that have to be multiplied.

#### Remarks:

#VALUE! - occurs if any cells in array are empty or contain string.

#VALUE! - occurs if the array does not have an equal number of rows and columns.

#### [4.1.4.6.6.155 MOD](#)

Returns the remainder after the number is divided by a divisor. The result has the same sign as the divisor.

#### Syntax

**MOD(number, divisor)**, where:

**number** is the number for which you want to find the remainder.

**divisor** is the value by which you want to divide the number.

#### Remarks

- The MOD function can be expressed in terms of the INT function,

$$\text{MOD}(n, d) = n - d * \text{INT}(n/d)$$

#### [4.1.4.6.6.156 MODE](#)

Returns the most frequently occurring or repetitive, value in an array or range of data.

#### Syntax

**MODE(number1, number2, ...)**, where:

**number1, number2, ...** are arguments for which you want to calculate the mode.

#### Remarks

- In a set of values, the mode is the most frequently occurring value.

#### [4.1.4.6.6.157 MONTH](#)

Returns the month of a date represented by a serial number. The month is given as an integer, ranging from 1 (January) to 12 (December).

#### Syntax

**MONTH(serial\_number)**, where:

**serial\_number** is the date of the month you are trying to find. Dates should be entered by using the DATE function or as results of other formulas or functions. For example, use DATE(2002,11,12) for the 12th day of Nov, 2002.

#### Remarks

- Dates are stored as sequential serial numbers so that they can be used in calculations. By default, January 1, 1900 is serial number 1 and January 1, 2008 is serial number 39448 because it is 39,448 days after January 1, 1900.

#### [4.1.4.6.6.158 MROUND](#)

The MROUND function rounds a given number up or down to the nearest multiple of a given number.

#### Syntax

The syntax of the MROUND function is

**= MROUND (number, multiple)**

Number – The value to be rounded. This value is required.

Multiple – This value is required.

#### Remarks:

The number must be greater than or equal to half the value of multiple.

#NUM! - Occurs if the number and multiple have different signs.

#VALUE! - Occurs if any of the given arguments are non-numeric.

#### Example:

FORMULA	RESULT
---------	--------

=MROUND(10,2.6)	8
=MROUND(-10,-2.6)	-8
=MROUND(10,-2)	#NUM!

•

#### 4.1.4.6.6.159 Multinomial

The MULTINOMIAL function returns the ratio of the factorial of a sum of values to the product of factorials.

#### Syntax

The syntax of the MULTINOMIAL function is

**=MULTINOMIAL(number1, (number2), ...)**

#### Remarks:

#NUM! - Occurs if any of the supplied arguments are less than 0.

#VALUE! - Occurs if any of the supplied arguments are non-numeric.

#### Example:

FORMULA	RESULT
=MULTINOMIAL(2, 3, 4)	1260

#### 4.1.4.6.6.160 MUNIT

The MUNIT function retrieves the unit matrix for the particular dimension that has been specified.

#### Syntax:

**MUNIT(dimension) where:**

- Dimension is an integer specifying the dimension of the unit matrix that you want to return.

#### Remarks:

#VALUE! - occurs if the dimension is a value that is equal to or smaller than zero.

#### [4.1.4.6.6.161 N](#)

The N function converts the given value into a numeric value.

##### **Syntax:**

The syntax of the N function is

**=N (value)**

A value is required.

Numeric values are converted as numeric values. A date value is converted as a serial number. The Logic operator TRUE returns a value of 1. The other values are returned as 0.

##### **Example:**

FORMULA	RESULT
=N(7)	7
=N("EVEN")	0
=N(1/1/2008)	39448
=N(TRUE)	1

#### [4.1.4.6.6.162 NA](#)

The NA function returns the #N/A error. This error message is produced when a formula is unable to find a value that it needs. This error message denotes 'value not available.'

##### **Syntax:**

The syntax of NA function is

**=NA()**

The NA function syntax has no arguments.

##### **Remarks:**

The function doesn't have any arguments.

##### **Example:**

FORMULA	RESULT
=NA()	#NA

#### 4.1.4.6.6.163 NEGBINOMDIST

Returns the negative binomial distribution. NEGBINOMDIST returns the probability that there will be number\_f failures before the number\_s-th success, when the constant probability of a success is probability\_s.

#### Syntax

**NEGBINOMDIST(number\_f, number\_s, probability\_s)**, where:

**number\_f** is the number of failures.

**number\_s** is the threshold number of successes.

**probability\_s** is the probability of a success.

#### Remarks

- **number\_s** must be  $\geq 1$ .
- **probability\_s** must be  $\geq 0$  and  $\leq 1$ .
- **number\_f** must be  $\geq 0$ .
- The equation for the negative binomial distribution is,

$$nb(x;r;p) = \binom{x+r-1}{r-1} p^r (1-p)^x$$

where x is number\_f, r is number\_s and p is probability\_s.

#### 4.1.4.6.6.164 NETWORKDAYS

The NETWORKDAYS function is used to calculate the number of whole work days between two given dates. This includes all weekdays from Monday to Friday, but excludes a supplied list of holidays.

#### Syntax

= NETWORKDAYS( start\_date, end\_date, [holidays] )

**start\_date**: The start of the period to find the working days

**end\_date**: The end of the period to find the working days.

[holidays]: An optional argument, which specifies an array of dates that are not to be counted as working days.

## Notes

- If any argument is not a valid date, NETWORKDAYS returns the #VALUE! error value.

## Example

FORMULA	RESULT
=NETWORKDAYS(DATE(2012,10,1),DATE(2013,3,1))	110

### 4.1.4.6.6.165 NORMDIST

Returns the normal distribution for the specified mean and standard deviation.

## Syntax

**NORMDIST(x, mean, standard\_dev, cumulative)**, where:

**x** is the value for which you want the distribution.

**mean** is the arithmetic mean of the distribution.

**standard\_dev** is the standard deviation of the distribution.

**cumulative** is a logical value that determines the form of the function. If cumulative is True, NORMDIST returns the cumulative distribution function; if False, it returns the probability mass function.

## Remarks

- Standard\_dev must be > 0.
- The equation for the normal density function (cumulative = False) is,

$$f(x, \mu, \sigma) = \frac{e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\sqrt{2\pi} \sigma}$$

- When cumulative = True, the formula is the integral from negative infinity to x of the given formula.
- #VALUE! - occurs if mean is non-numeric.
- #VALUE! - occurs if standard\_dev is non-numeric.
- #NUM! - occurs if standard\_dev is equal to or less than zero.

#### [4.1.4.6.166 NormsDist](#)

The NormsDist function returns the probability that the observed value of a standard normal random variable will be less than or equal to the parameter.

#### Syntax:

**NormsDist(value)**

where:

**value** is a numeric value that checks with the random variable.

#### [4.1.4.6.167 NORMINV](#)

Returns the inverse of the normal cumulative distribution for the specified mean and standard deviation.

#### Syntax

**NORMINV(probability, mean, standard\_dev)**, where:

**NORM.INV(probability,mean,standard\_dev)** where:

- probability is a probability that corresponds to the normal distribution.
- mean is the arithmetic mean of the distribution.
- standard\_dev is the standard deviation of the distribution.

#### Remarks:

#NUM! - if probability is equal to or less than zero.

#NUM! - if probability is equal to or greater than 1.

#VALUE! - if probability is non-numeric.

#VALUE! - if mean is non-numeric.

#NUM! - if standard\_dev is equal to or less than zero.

#VALUE! - if standard\_dev is non-numeric.

Given a value for probability, NORMINV seeks value x such that NORMDIST(x, mean, standard\_dev, True) = probability. NORMINV uses an iterative search technique.

#### [4.1.4.6.168 NormsInv](#)

The NormsInv function returns the standard normal random variable that has Mean 0 and Standard Deviation 1

**Syntax:**

**NormsDist(value)**

where:

**value** is the probability of the standard deviation.

#### [4.1.4.6.169 NORM.S.DIST](#)

The NORM.S.DIST function returns the standard normal distribution.

**Syntax:**

**NORM.S.DIST(z, cumulative)** where:

**z** is the value for which you want the distribution.

**cumulative** is a logical value that determines the form of the function.

**Remarks:**

The equation for the standard normal density function is:

$$f(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}}$$

#### [4.1.4.6.170 NORM.S.INV](#)

The NORM.S.INV function returns the inverse of the standard normal cumulative distribution.

**Syntax:**

**NORM.S.INV(probability)** where:

- probability is a probability that corresponds to the normal distribution.

#### [4.1.4.6.6.171 NOT](#)

Reverses the value of its argument.

#### Syntax

**NOT(logical)**, where:

**logical** is a value or expression that can be evaluated to True or False.

#### [4.1.4.6.6.172 NOW](#)

Returns the serial number of the current date and time.

#### Syntax

**NOW()**

#### Remarks

- Dates are stored as sequential serial numbers so that they can be used in calculations. By default, January 1, 1900 is serial number 1 and January 1, 2008 is serial number 39448 because it is 39,448 days after January 1, 1900.
- Numbers to the right of the decimal point in the serial number represent the time; numbers to the left represent the date. For example, the serial number .5 represents the time 12:00 noon.

#### [4.1.4.6.6.173 NPER](#)

Returns the number of periods for an investment based on periodic, constant payments and a constant interest rate.

#### Syntax

**NPER(rate, pmt, pv, fv, type)**, where:

**rate** is the interest rate per period.

**pmt** is the payment made each period; it cannot change over the life of the annuity.

**pv** is the present value or the lump-sum amount that a series of future payments is worth right now.

**fv** is the future value or a cash balance that you want to attain after the last payment is made. If **fv** is omitted, it is assumed to be 0 (the future value of a loan, for example, is 0).

**type** is the number 0 or 1 and indicates when payments are due. If **type** equals:

- 0 - Payments are due at the end of the period.
- 1 - Payments are due at the beginning of the period.

#### [4.1.4.6.6.174 NPV](#)

Calculates the net present value of an investment by using a discount rate and a series of future payments (negative values) and income (positive values).

#### Syntax

**NPV(rate, value1, value2, ...)**, where:

**rate** is the rate of discount over the length of one period.

**value1, value2, ...** are arguments representing the payments and income.

- Value1, value2, ... must be equally spaced in time and occur at the end of each period.
- NPV uses the order of value1, value2, ... to interpret the order of cash flows. Be sure to enter your payment and income values in the correct sequence.

#### Remarks

- The NPV investment begins one period before the date of the value1 cash flow and ends with the last cash flow in the list. The NPV calculation is based on future cash flows. If your first cash flow occurs at the beginning of the first period, the first value must be added to the NPV result, not included in the value arguments.
- If n is the number of cash flows in the list of values, the formula for NPV is,

$$NPV = \sum_{i=1}^n \frac{values_i}{(1+rate)^i}$$

#### [4.1.4.6.6.175 ODD](#)

Returns the number rounded up to the nearest odd integer.

#### Syntax

**ODD(number)**, where:

**number** is the value to be rounded off.

#### Remarks

- Regardless of the sign of a number, a value is rounded up when adjusted away from zero. If the number is an odd integer, no rounding occurs.

#### [4.1.4.6.6.176 OR](#)

Returns True if any argument is True; returns False if all arguments are False.

#### Syntax

**OR(logical1, logical2, ...)**, where:

**logical1, logical2, ...** are conditions you want to test that can be either True or False.

#### Remarks

- The arguments must evaluate to logical values such as True or False or in arrays or references that contain logical values.

#### [4.1.4.6.6.177 PEARSON](#)

Returns the Pearson product moment correlation coefficient, r, a dimensionless index that ranges from -1.0 to 1.0 inclusive and reflects the extent of a linear relationship between two data sets.

## Syntax

**PEARSON(array1, array2)**, where:

**array1** is a set of independent values.

**array2** is a set of dependent values.

## Remarks

- The arguments must be either numbers or names, array constants or references that contain numbers.
- The formula for the Pearson product moment correlation coefficient, r, is,

$$r = \frac{\sum(x - \bar{x})(y - \bar{y})}{\sqrt{\sum(x - \bar{x})^2 \sum(y - \bar{y})^2}}$$

where x-bar and y-bar are the sample means AVERAGE(array1) and AVERAGE(array2).

### 4.1.4.6.6.178 PERCENTILE

Returns the k-th percentile of values in a range.

## Syntax

**PERCENTILE(array, k)**, where:

**array** is the array or range of data that defines relative standing.

**k** is the percentile value in the range 0..1, inclusive.

## Remarks

- k must be >=0 and <= 1.
- If k is not a multiple of 1/(n - 1), PERCENTILE interpolates to determine the value at the k-th percentile.

#### [4.1.4.6.6.179 PERCENTILE.EXC](#)

The PERCENTILE.EXC function returns the k-th percentile of values in a range, where k is in the range of 0 to 1, exclusively.

##### **Syntax:**

**PERCENTILE.EXC(array, k)** where:

- array is the range of data that defines relative standing.
- k is the percentile value in the range of 0 to 1.

##### **Remarks:**

#NUM! - occurs if k is equal to or less than zero.

#NUM! - occurs if the array is empty.

#NUM! - occurs if k is equal to or greater than 1.

#VALUE! - occurs if k is non-numeric.

#### [4.1.4.6.6.180 PERCENTILE.INC](#)

The PERCENTILE.INC function returns the k-th percentile of values in a range, where k is in the range 0 to 1.

##### **Syntax:**

**PERCENTILE.INC(array,k)** where:

- array is the range of data that defines relative standing.
- k is the percentile value in the range 0 to 1.

##### **Remarks:**

#NUM! - occurs if k is equal to or less than zero.

#NUM! - occurs if array is empty.

#NUM! - occurs if k is equal to or greater than 1.

#VALUE! - occurs if k is non-numeric.

#### [4.1.4.6.6.181 PERCENTRANK](#)

Returns the rank of a value in a data set as a percentage of the data set.

##### **Syntax**

**PERCENTRANK(array, x, significance)**, where:

**array** is the range of data with numeric values that defines relative standing.

**x** is the value for which you want to know the rank.

**significance** is an optional value that identifies the number of significant digits for the returned percentage value. If omitted, PERCENTRANK uses three digits (0.xxx).

## Remarks

- Significance must be  $\geq 1$ .
- If **x** does not match one of the values in the array, PERCENTRANK interpolates to return the correct percentage rank.

### [4.1.4.6.6.182 PERCENTRANK.EXC](#)

The PERCENTRANK.EXC function returns the rank of a value in a data set as a percentage (0 to 1, exclusively) of the data set.

#### Syntax:

**PERCENTRANK.EXC(array, x, significance )** where:

- **array** is the range of data that defines relative standing.
- **x** is value for which you want to know the rank.
- **significance** is an optional value that identifies the number of significant digits for the returned percentage value.

#### Remarks:

#NUM! - occurs if this argument is empty.

#NUM! - occurs if the argument is less than one.

### [4.1.4.6.6.183 PERMUT](#)

Returns the number of permutations for a given number of objects that can be selected from a number of objects.

#### Syntax

**PERMUT(number, number\_chosen)**, where:

**number** is an integer that describes the number of objects.

**number\_chosen** is an integer that describes the number of objects in each permutation.

## Remarks

- Number must be > 0 and number\_chosen must be >= 0.
- Number must be >= number\_chosen.
- The equation for the number of permutations is,

$$P_{k,n} = \frac{n!}{(n-k)!}$$

### 4.1.4.6.184 PERMUTATIONA

The PERMUTATIONA function returns the number of permutations for a given number of objects (with repetitions) that can be selected from the total number of objects.

#### Syntax:

**PERMUTATIONA(number, number-chosen)** where:

- number is an integer that describes the total number of objects.
- number-chosen is an integer that describes the number of objects in each permutation.

#### Remarks:

#VALUE! - occurs if numeric arguments use data types that are non-numeric.

#NUM! - occurs if numeric arguments are values that are not valid.

Both arguments are truncated to integers.

### 4.1.4.6.185 PI

Returns the number 3.14159265358979, the mathematical constant pi, accurate to 15 digits.

#### Syntax

**PI()**

### 4.1.4.6.186 PMT

Calculates the payment for a loan based on constant payments and a constant interest rate.

## Syntax

**PMT(rate, nper, pv, fv, type)**, where:

**rate** is the interest rate for the loan.

**nper** is the total number of payments for the loan.

**pv** is the present value or the total amount that a series of future payments is worth now; also known as the principal.

**fv** is the future value or a cash balance you want to attain after the last payment is made. If **fv** is omitted, it is assumed to be 0 (zero), that is, the future value of a loan is 0.

**type** is the number 0 (zero) or 1 and indicates when payments are due. If **type** equals:

- 0 - Payments are due at the end of the period.
- 1 - Payments are due at the beginning of the period.

## Remarks

- The payment returned by PMT includes principal and interest but no taxes, reserve payments or fees sometimes associated with loans.
- Make sure that you are consistent about the units you use for specifying **rate** and **nper**. If you make monthly payments on a four-year loan at an annual interest rate of 12 percent, use 12%/12 for **rate** and 4\*12 for **nper**. If you make annual payments on the same loan, use 12 percent for **rate** and 4 for **nper**.

### 4.1.4.6.6.187 POISSON

Returns the Poisson distribution.

## Syntax

**POISSON(x, mean, cumulative)**, where:

**x** is the number of events.

**mean** is the expected numeric value.

**cumulative** is a logical value that determines the form of the probability distribution returned. If **cumulative** is True, **POISSON** returns the cumulative Poisson probability that the number of random events occurring will be between zero and **x** inclusive; if False, it returns the Poisson probability mass function that the number of events occurring will be exactly **x**.

## Remarks

- X must be  $\geq 0$ .
- Mean must be  $> 0$ .
- POISSON is calculated as follows:

For cumulative = False,

$$POISSON = \frac{e^{-\lambda} \lambda^x}{x!}$$

For cumulative = True,

$$CUMPOISSON = \sum_{k=0}^x \frac{e^{-\lambda} \lambda^k}{k!}$$

### 4.1.4.6.188 Pow

The **Pow** function returns the number raised to the specified power.

#### Syntax:

**POW(number, power)**

where:

**number** is the base number. It can be any real number.

**power** is the exponent to which, the base number is raised.

### 4.1.4.6.189 POWER

Returns the result of a number raised to a power.

#### Syntax

**POWER(number, power)**, where:

**number** is the base number. It can be any real number.

**power** is the exponent to which the base number is raised.

#### [4.1.4.6.6.190 PPMT](#)

Returns the payment on the principal for a given period, for an investment based on periodic, constant payments and a constant interest rate.

#### Syntax

**PPMT(rate, per, nper, pv, fv, type)**, where:

**rate** is the interest rate per period.

**per** specifies the period and must be in the range of 1 to nper.

**nper** is the total number of payment periods in an annuity.

**pv** is the present value—the total amount that a series of future payments is worth now.

**fv** is the future value or a cash balance that you may want to attain after the last payment is made. If fv is omitted, it is assumed to be 0 (zero), that is, the future value of a loan is 0.

**type** is the number 0 or 1 and indicates when payments are due. If type equals:

- 0 - Payments are due at the end of the period.
- 1 - Payments are due at the beginning of the period.

#### Remarks

- Make sure that you are consistent about the units you use for specifying rate and nper. If you make monthly payments on a four-year loan at 12 percent annual interest, use 12%/12 for rate and 4\*12 for nper. If you make annual payments on the same loan, use 12% for rate and 4 for nper.

#### [4.1.4.6.6.191 PROB](#)

Returns the probability whose values are in a range that is between two limits. If upper\_limit is not supplied, returns the probability that values in x\_range are equal to lower\_limit.

#### Syntax

**PROB(x\_range, prob\_range, lower\_limit, upper\_limit)**, where:

**x\_range** is the range of numeric values of x with which there are associated probabilities.

**prob\_range** is a set of probabilities associated with values in **x\_range**.

**lower\_limit** is the lower bound on the value for which you want a probability.

**upper\_limit** is the optional upper bound on the value for which you want a probability.

## Remarks

- Any value in **prob\_range** must be > 0 and < 1.
- If **upper\_limit** is omitted, PROB returns the probability of being equal to **lower\_limit**.

### [4.1.4.6.6.192 PRODUCT](#)

Multiplies all the numbers given as arguments and returns the product.

## Syntax

**PRODUCT(number1, number2, ...)**, where:

**number1, number2, ...** are numbers that you want to multiply.

### [4.1.4.6.6.193 PV](#)

Returns the present value of an investment. The present value is the total amount that a series of future payments is worth now.

## Syntax

**PV(rate, nper, pmt, fv, type)**, where:

**rate** is the interest rate per period. For example, if you obtain an automobile loan at a 10% annual interest rate and make monthly payments, your interest rate per month is 10%/12 or 0.83%. You would enter 10%/12 or 0.83% or 0.0083, into the formula as the rate.

**nper** is the total number of payment periods in an annuity. For example, if you get a four-year car loan and make monthly payments, your loan has 4\*12 (or 48) periods. You would enter 48 into the formula for **nper**.

**pmt** is the payment made for each period and cannot change over the life of the annuity. Typically, pmt includes principal and interest but, no other fees or taxes. For example, the monthly payments on a \$10,000, four-year car loan at 12 percent are \$263.33. You will have to enter -263.33 into the formula as the pmt. If pmt is omitted, you must include the fv argument.

**fv** is the future value or a cash balance that you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0 (the future value of a loan, for example, is 0). For example, if you want to save \$50,000 to pay for a special project in 18 years, then \$50,000 is the future value. You could then make a conservative guess at an interest rate and determine how much you must save each month. If fv is omitted, you must include the pmt argument.

**type** is the number 0 or 1 and indicates when payments are due. If type equals:

- 0 - Payments are due at the end of the period.
- 1 - Payments are due at the beginning of the period.

### Remarks

- Make sure that you are consistent about the units you use for specifying rate and nper. If you make monthly payments on a four-year loan at 12 percent annual interest, use 12%/12 for rate and 4\*12 for nper. If you make annual payments on the same loan, use 12% for rate and 4 for nper.
- In annuity functions, the cash you pay out such as a deposit to savings is represented by a negative number; the cash you receive such as a dividend check is represented by a positive number.
- One financial argument is solved for in terms of the others. If rate is not 0, then,

$$pv * (1+rate)^{nper} + pmt(1+rate*type) * \left( \frac{(1+rate)^{nper} - 1}{rate} \right) + fv = 0$$

If rate is 0, then,

$$(pmt * nper) + pv + fv = 0$$

#### [4.1.4.6.6.194 QUARTILE](#)

Returns the quartile of a data set.

### Syntax

QUARTILE(array, quart), where:

array is the array or cell range of numeric values for which you want the quartile value.

quart indicates which value to return.

Quartile	Value Returned
0	Minimum value
1	First quartile (25th percentile)
2	Median value (50th percentile)
3	Third quartile (75th percentile)
4	Maximum value

#### 4.1.4.6.6.195 QUOTIENT

The QUOTIENT function returns the integer portion of a division between two given numbers. The returned value will be integer value.

#### Syntax

The syntax of the QUOTIENT function is

**=QUOTIENT (numerator, denominator)**

Numerator – Required.

Denominator – Required.

#### Remarks:

#VALUE! - Occurs if any of the given arguments are non-numeric.

#### Example:

FORMULA	RESULT
= QUOTIENT (10,3)	3

= QUOTIENT (-20,6)	-3
--------------------	----

#### [4.1.4.6.6.196 RADIANS](#)

Converts degrees to radians.

#### Syntax

**RADIANS(angle)**, where:

**angle** is an angle in degrees that you want to convert.

#### [4.1.4.6.6.197 RAND](#)

Returns an evenly distributed random number greater than or equal to 0 and less than 1.

#### Syntax

**RAND()**

#### [4.1.4.6.6.198 RANDBETWEEN](#)

The RANDBETWEEN function returns a random number that is between the given ranges. This function returns a new random number each time in recalculation.

#### Syntax

The syntax of the RANDBETWEEN Function is

**=RANDBETWEEN (start\_num, end\_num)**

start\_num – Required. This is the smallest integer.

end\_num – Required. This is the largest integer.

#### Remarks:

#NUM! - Occurs if the end\_num value is larger than start\_num value.

#VALUE! - Occurs if any of the given arguments are non-numeric.

**Example:**

FORMULA	RESULT
= RANDBETWEEN(10,20)	12. The value is generated randomly between the given values.

[4.1.4.6.6.199 RANK](#)

Returns the rank of a number in a list of numbers. The rank of a number is its size relative to other values in a list. (If you were to sort the list, the rank of the number would be its position)

**Syntax**

**RANK(number, ref, order)**, where:

**number** is the number whose rank you want to find.

**ref** is an array of or a reference to a list of numbers.

**order** is a number specifying how to rank numbers.

- If the order is 0 (zero) or omitted, the number is ranked as if ref were a list sorted in descending order.
- If the order is any nonzero value, the number is ranked as if ref were a list sorted in ascending order.

**Remark**

- RANK gives duplicate numbers of the same rank. However, the presence of duplicate numbers will affect the ranks of subsequent numbers.

[4.1.4.6.6.200 RATE](#)

Returns the interest rate per period of an annuity. RATE is calculated by iteration and may not converge to a unique solution.

**Syntax**

**RATE(nper, pmt, pv, fv, type, guess)**, where:

**nper** is the total number of payment periods in an annuity.

**pmt** is the payment made for each period and cannot change over the life of the annuity.

Typically, pmt includes the principal and interest but, no other fees or taxes. If pmt is omitted, you must include the fv argument.

**pv** is the present value—the total amount that a series of future payments is worth now.

**fv** is the future value or a cash balance that you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0 (the future value of a loan, for example, is 0).

**type** is the number 0 or 1 and indicates when payments are due. If type equals:

- **0** - Payments are due at the end of the period.
- **1** - Payments are due at the beginning of the period.
- **guess** is your guess for what the rate will be.
- If you omit guess, it is assumed to be 10 percent.
- If RATE does not converge, try different values for guess. RATE usually converges if guess is between 0 and 1.

#### *4.1.4.6.6.201 RECEIVED*

The Received function returns the amount received at maturity for a fully invested security.

**Syntax:**

**RECEIVED(settlement,maturity,investment,discount, basis)** where:

- Maturity - security's maturity date.
- Settlement - security's settlement date.
- Discount - security's discount rate.
- Basis - type of day count basis.

**Remarks:**

#NUM! - occurs if settlement  $\geq$  maturity

#VALUE! - occurs if valid date is not entered for maturity and settlement.

#NUM! - occurs if investment and discount is  $\leq 0$

RECEIVED is calculated as follows:

$$RECEIVED = \frac{investment}{1 - (discount \times \frac{DSM}{B})}$$

where:

- B is the number of days in year, depending on the year basis.
- DSM is the number of days from issue to maturity.

#### [4.1.4.6.6.202 RIGHT](#)

RIGHT returns the last character or characters in a text string, based on the number of characters you specify.

#### Syntax

**RIGHT(text, num\_chars)**, where:

**text** is the text string containing the characters you want to extract.

**num\_chars** specifies the number of characters you want RIGHT to extract.

#### Remarks

- **Num\_chars** must be **greater than or equal to zero**.
- If **num\_chars** is greater than the length of **text**, **RIGHT** returns all the **text**.
- If **num\_chars** is omitted, it is assumed to be 1.

#### [4.1.4.6.6.203 ROMAN](#)

The ROMAN function converts an Arabic number to a Roman numeral. This function returns a text string depicting the Roman numeral form of the given number.

#### Syntax

The syntax of the ROMAN function is

**=ROMAN( number, (form) )**

number – Required.

Form – Optional, this value will specify the style of the Roman numeral.

If number is not an integer, then it will be rounded down.

Form	Type
0 or omitted	Classic.
1	More concise. See example below.
2	More concise. See example below.
3	More concise. See example below.
4	Simplified.

#### Remarks:

#VALUE! - Occurs if any of the given values is non-numeric, or for values less than 0 and greater than 3999.

**Example:**

FORMULA	RESULT
= ROMAN (499,0)	CDXCIX
= ROMAN (499,1)	ID
=ROMAN(-100)	#VALUE!

#### [4.1.4.6.6.204 ROUND](#)

Rounds a number to a specified number of digits.

**Syntax**

**ROUND(number, num\_digits)**, where:

**number** is the number you want to round off.

**num\_digits** specifies the number of digits you want to round off.

**Remarks**

- If num\_digits > 0, then number is rounded off to the specified number of decimal places.
- If num\_digits = 0, then number is rounded off to the nearest integer.
- If num\_digits < 0, then number is rounded off to the left of the decimal point.

#### [4.1.4.6.6.205 ROUNDDOWN](#)

Rounds a number down towards zero.

**Syntax**

**ROUNDDOWN(number, num\_digits)**, where:

**number** is any real number that you want rounded down.

**Num\_digits** is the number of digits to which you want to round a number.

### Remark

- **ROUNDDOWN** behaves like **ROUND**, except that it always rounds a number down.

#### *4.1.4.6.6.206 ROUNDUP*

Rounds a number up away from 0 (zero).

### Syntax

**ROUNDUP(number, num\_digits)**, where:

**number** is any real number that you want rounded up.

**num\_digits** is the number of digits to which you want to round a number.

### Remarks

- **ROUNDUP** behaves like **ROUND**, except that it always rounds a number up.

#### *4.1.4.6.6.207 RSQ*

Returns the square of the Pearson product moment correlation coefficient through data points in **known\_y's** and **known\_x's**.

### Syntax

**RSQ(known\_y's, known\_x's)**, where:

**known\_y's** is an array or range of data points.

**known\_x's** is an array or range of data points.

### Remarks

- The equation for the Pearson product moment correlation coefficient is,

$$r = \frac{\sum(x - \bar{x})(y - \bar{y})}{\sqrt{\sum(x - \bar{x})^2 \sum(y - \bar{y})^2}}$$

where:

x-bar and y-bar are the sample means AVERAGE(known\_x's) and AVERAGE(known\_y's).

RSQ returns r2 which, is the square of this correlation coefficient.

#### 4.1.4.6.6.208 SEARCH

The SEARCH function returns the location of a substring in a string. This function is NOT case-sensitive.

#### Syntax

`SEARCH(substring, string, [start_position] )`

substring: Required. The text to be found.

string: Required. The text in which to search for the value of the substring.

start\_num: Optional. The starting position for searching in the string.

#### Notes

- If the value of find\_text is not found, the #VALUE! error value is returned.
- If the start\_num argument is omitted, it is assumed to be 1.
- If start\_num is not greater than 0, or is greater than the length of the string argument, the #VALUE! error value is returned.

#### Example

FORMULA	RESULT
=SEARCH("base","database")	5

#### 4.1.4.6.6.209 SEC

The SEC function returns the secant of an angle.

#### Syntax:

`SEC(number)` where:

- number - angle radians for which you want the secant

#### Remarks:

#NUM! - occurs if the number is outside of its constraints.

#VALUE! - occurs if number is a non-numeric value.

#### [4.1.4.6.6.210 SECH](#)

The SECH function returns the hyperbolic secant of an angle.

##### **Syntax:**

**SECH(number)** where:

- number - angle radians for which you want the secant

##### **REMARKS:**

#NUM! - occurs if the number is outside of its constraints.

#VALUE! - occurs if the number is a non-numeric value.

#### [4.1.4.6.6.211 SECOND](#)

Returns the seconds of a time value. The second is given as an integer in the range 0 (zero) to 59.

##### **Syntax**

**SECOND(serial\_number)**, where:

**serial\_number** is the time that contains the seconds you want to find.

##### **Remarks**

- Time values are a portion of a date value and are represented by a decimal number (for example, 12:00 PM is represented as 0.5 because it is half of a day).

#### [4.1.4.6.6.212 SIGN](#)

Determines the sign of a number. Returns 1 if the number is positive, zero (0) if the number is 0 and -1 if the number is negative.

##### **Syntax**

**SIGN(number)**, where:

**number** is any real number.

#### [4.1.4.6.6.213 SIN](#)

Returns the sine of the given angle.

#### **Syntax**

**SIN(number)**, where:

**number** is the angle in radians for which you want the sine.

#### [4.1.4.6.6.214 SINH](#)

Returns the hyperbolic sine of a number.

#### **Syntax**

**SINH(number)**, where:

**number** is any real number.

#### **Remarks**

- The formula for the hyperbolic sine is,

$$\sinh(z) = \frac{e^z - e^{-z}}{2}$$

#### [4.1.4.6.6.215 SKEW](#)

Returns the skewness of a distribution. Skewness characterizes the degree of asymmetry of a distribution around its mean.

## Syntax

**SKEW(number1, number2, ...)**, where:

**number1, number2 ...** are arguments for which you want to calculate the skewness. You can also use a single array or a reference to an array instead of arguments separated by commas.

## Remarks

- The equation for skewness is defined as follows.

$$\frac{n}{(n-1)(n-2)} \sum \left( \frac{x_i - \bar{x}}{s} \right)^3$$

### [4.1.4.6.6.216 SKEW.P.](#)

Returns the skewness of a distribution based on a population: a characterization of the degree of asymmetry of a distribution around its mean.

## Syntax:

**SKEW.P(number 1, [number 2],...)** where:

- Number 1 is required, and subsequent numbers are optional. Number 2,... up to 254 are numbers or names, arrays, or references that contain numbers for which you want the population skewness, can be used.

## Remarks:

#NUM! - occurs if arguments do not have a valid value.

#VALUE! - occurs if arguments do not have valid data types.

#DIV/0! - occurs if there are fewer than three data points, or the sample standard deviation is zero.

### [4.1.4.6.6.217 SLN](#)

Returns the straight-line depreciation of an asset for one period.

## Syntax

**SLN(cost, salvage, life)**, where:

**cost** is the **initial cost** of the asset.

**salvage** is the **value at the end of the depreciation** (sometimes called the salvage value of the asset).

**life** is the number of periods over which the asset is depreciated (the **useful life** of the asset).

#### *4.1.4.6.6.218 SLOPE*

Returns the slope of the linear regression line through data points in known\_y's and known\_x's. The slope is the rate of change along the regression line.

#### **Syntax**

**SLOPE(known\_y's, known\_x's)**, where:

**known\_y's** is an array or cell range of **numeric dependent data points**.

**known\_x's** is the set of **independent data points**.

#### **Remarks**

- The equation for the slope of the regression line is,

$$b = \frac{\sum(x - \bar{x})(y - \bar{y})}{\sum(x - \bar{x})^2}$$

where x-bar and y-bar are the sample means AVERAGE(known\_x's) and AVERAGE(known\_y's).

#### *4.1.4.6.6.219 SMALL*

Returns the k-th smallest value in a data set.

#### **Syntax**

**SMALL(array, k)**, where:

**array** is an **array** or range of numerical data for which you want to determine the k-th smallest value.

**k** is the **position** (from the smallest) in the array or range of data to return.

#### [4.1.4.6.6.220 SQRT](#)

Returns a positive square root.

#### Syntax

**SQRT(number)**, where:

**number** is the **number** for which you want the square root.

#### Remarks

- Number must be  $\geq 0$ .

#### [4.1.4.6.6.221 SQRTPI](#)

The SQRTPI function returns the square root of a given number multiplied by  $\pi$ . Here  $\pi$  is the constant math value.

#### Syntax

The syntax of the SQRTPI function is

**=SQRTPI (number)**

number – Required.

#### Remarks:

#NUM! - If the number is less than zero (0).

#VALUE! - Occurs if any of the given arguments are non-numeric.

#### Example:

FORMULA	RESULT
= SQRTPI (2)	2.506628

= SQRTPI (-2)	#NUM!
---------------	-------

#### [4.1.4.6.6.222 STANDARDIZE](#)

Returns a normalized value from a distribution characterized by mean and standard\_dev.

#### Syntax

**STANDARDIZE(x,mean, standard\_dev)**, where:

**x** is the **value** that you want to normalize.

**mean** is the **arithmetic mean** of the distribution.

**standard\_dev** is the **standard deviation** of the distribution.

#### Remarks

- standard\_dev must be > 0.
- The equation for the normalized value is,

$$Z = \frac{X - \mu}{\sigma}$$

#### [4.1.4.6.6.223 STDEV](#)

Estimates the standard deviation based on a sample. The standard deviation is a measure of how widely values are dispersed from the average value (the mean).

#### Syntax

**STDEV(number1, number2, ...)**, where:

**number1, number2, ...** are **number arguments** corresponding to a sample of a population.

#### Remarks

- STDEV assumes that its arguments are a sample of the population. If your data represents the entire population, then compute the standard deviation using STDEVP.
- STDEV uses the following formula,

$$\sqrt{\frac{\sum(x - \bar{x})^2}{(n-1)}}$$

where  $\bar{x}$  is the sample mean **AVERAGE(number1,number2,...)** and  $n$  is the sample size.

#### 4.1.4.6.6.224 STDEVA

Estimates standard deviation based on a sample. The standard deviation is a measure of how widely values are dispersed from the average value (the mean). Text and logical values such as True and False are also included in the calculation.

#### Syntax

**STDEVA(value1, value2, ...)**, where:

**value1, value2, ...** are values corresponding to a sample of a population. You can also use a single array or a reference to an array instead of arguments separated by commas.

#### Remarks

- Arguments that contain True evaluate as 1; arguments that contain text or False evaluate as 0 (zero).
- STDEVA uses the following formula,

$$\sqrt{\frac{\sum(x - \bar{x})^2}{(n-1)}}$$

where  $\bar{x}$  is the sample mean **AVERAGE(value1,value2,...)** and  $n$  is the **sample size**.

#### 4.1.4.6.6.225 STDEVP

Calculates standard deviation based on the entire population given as arguments.

#### Syntax

**STDEVP(number1, number2, ...)**, where:

**number1, number2, ...** are **1 to 30 number arguments** corresponding to a population. You can also use a single array or a reference to an array instead of arguments separated by commas.

### Remarks

- STDEVP assumes that its arguments are the entire population. If your data represents a sample of the population, then compute the standard deviation using STDEV.
- STDEVP uses the following formula:

$$\sqrt{\frac{\sum(x - \bar{x})^2}{n}}$$

where **x** is the sample **mean AVERAGE(number1,number2,...)** and **n** is the **sample size**.

### [4.1.4.6.6.226 STDEVPA](#)

Calculates the standard deviation based on the entire population given as arguments, including text and logical values.

### Syntax

**STDEVPA(value1, value2, ...)**, where:

**value1, value2, ...** are values corresponding to a population. You can also use a single array or a reference to an array instead of arguments separated by commas.

### Remarks

- Arguments that contain **True** evaluate as **1**; arguments that contain **text or False** evaluate as **0** (zero).
- STDEVPA uses the following formula,

$$\sqrt{\frac{\sum(x - \bar{x})^2}{n}}$$

where **x-bar** is the sample **mean AVERAGE(value1,value2,...)** and **n** is the **sample size**.

#### [4.1.4.6.6.227 STDEV.S](#)

Based on a sample, the STDEV.S function estimates standard deviation.

##### **Syntax:**

**STDEV.S(number1,[number2],...])** where:

- number1 is the first number argument corresponding to a population.
- Number2, ... are the arguments 2 to 254 corresponding to a population.

##### **Remarks:**

- Arguments can either be numbers or names, arrays, or references that contain numbers.
- The standard deviation is calculated using the "n" method.
- Arguments that are error values or text that cannot be translated into numbers cause errors.

#### [4.1.4.6.6.228 STEYX](#)

Returns the standard error of the predicted y-value for each x in the regression.

##### **Syntax**

**STEYX(known\_y's, known\_x's)**, where:

**known\_y's** is an array or range of dependent data points.

**known\_x's** is an array or range of independent data points.

##### **Remarks**

- The equation for the standard error of the predicted y is,

$$\sqrt{\frac{1}{(n-2)} \left[ \sum(y - \bar{y})^2 - \frac{\left[ \sum(x - \bar{x})(y - \bar{y}) \right]^2}{\sum(x - \bar{x})^2} \right]}$$

where **x-bar** and **y-bar** are the sample means **AVERAGE(known\_x's)** and **AVERAGE(known\_y's)** and **n** is the **sample size**.

#### [4.1.4.6.6.229 SUBSTITUTE](#)

Substitutes new\_text for old\_text in a text string. Use SUBSTITUTE when you want to replace specific text in a text string; use REPLACE when you want to replace any text that occurs in a specific location in a text string.

## Syntax

**SUBSTITUTE(text, old\_text, new\_text, instance\_num)**, where:

- **Text** is the text or the reference to a cell containing text for which you want to substitute characters.
- **Old\_text** is the text you want to replace.
- **New\_text** is the text you want to replace old\_text with.
- **Instance\_num** specifies which occurrence of old\_text you want to replace with new\_text. If you specify instance\_num, only that instance of old\_text is replaced. Otherwise, every occurrence of old\_text in text is changed to new\_text.

## Example

The example may be easier to understand if you copy it to a blank worksheet.

	A	
1	Data	
2	Sales Data	
3	Quarter 1, 2008	
4	Quarter 1, 2011	
	Formula	Description (Result)
	=SUBSTITUTE(A2, "Sales", "Cost")	Substitutes Cost for Sales (Cost Data).
	=SUBSTITUTE(A3, "1", "2", 1)	Substitutes first instance of "1" with "2" (Quarter 2, 2008).
	=SUBSTITUTE(A4, "1", "2", 3)	Substitutes third instance of "1" with "2" (Quarter 1, 2012).

### 4.1.4.6.6.230 SUBTOTAL

The SUBTOTAL function returns a subtotal in a list. Once the subtotal list is created, you can modify it by editing the SUBTOTAL function.

**Syntax:**

The syntax of the SUBTOTAL function is

**=SUBTOTAL (function\_Number, ref1, (ref2),...)**

A function\_Number is required. This specifies which function to use in calculating subtotals within a list. Here is the list of functions supported by Syncfusion:

FUNCTION NUMBER	FUNCTION NAME
1	AVERAGE
2	COUNT
3	COUNTA
4	MAX
5	MIN
6	PRODUCT
7	STDEV
8	STDEVP
9	SUM
10	VAR

**Ref1** The first named range which is used for the subtotal. This value is required.

**Ref2** This value is optional.

**Remarks:**

- If the subtotal function has any nested subtotal functions, then the nested subtotal is ignored for double counting.

**4.1.4.6.6.231 Sum**

The **Sum** function adds all numbers in a range of cells and returns the result.

**Syntax:**

**Sum(number1, number2, ... number\_n)**

where:

**number1** is the first number, **number2** is the second and **number\_n** is the nth number to be added together.

#### [4.1.4.6.6.232 SUMIF](#)

Adds the cells specified by a given criteria.

#### Syntax

**SUMIF(range, criteria, sum\_range)**, where:

**range** is the range of cells you want evaluated.

**criteria** is the criteria in the form of a number, expression or text that defines which, cells will be added. For example, criteria can be expressed as ">32" or some other logical expression.

**Sum\_range** are the actual cells to sum.

#### Remarks

- The cells in sum\_range are summed only if their corresponding cells in range match the criteria.
- If sum\_range is omitted, the cells in range are summed.

#### [4.1.4.6.6.233 SUMIFS](#)

The SUMIFS function sums the values in a given array that satisfy a set of given criteria.

#### Syntax:

=SUMIFS(sum\_range, criteria\_range1, criteria1, [criteria\_range2, criteria2], ...)

**criteria\_range1**: Array of values to be tested against the given criteria.

**criteria1**: The condition to be tested on each of the values of given range.

**sum\_range**: The range of values to be summed if the associated criteria range meets the specified criteria.

#### Notes

- Cells in the sum\_range argument that contain TRUE evaluate to 1; cells in sum\_range that contain FALSE evaluate to 0 (zero).

#### Example

##### Input Table

	A	B	C
1	Earning	Tax	other
2	100000	3000	100
3	200000	6000	200
4	300000	7500	300
5	400000	9000	500

FORMULA	RESULT
SUMIFS(C2:C5, B2:B5, ">7000", B2:B5, "<10000")	800

#### 4.1.4.6.6.234 SUMPRODUCT

Multiplies corresponding components in the given arrays and returns the sum of those products.

##### Syntax

**SUMPRODUCT(array1, array2, array3, ...)**, where:

**array1, array2, array3, ...** are 2 to 30 arrays whose components you will want to multiply and then add.

##### Remarks

- The array arguments must have the same dimensions.
- SUMPRODUCT treats array entries that are not numeric as if they were zeros.

#### 4.1.4.6.6.235 SUMSQ

Returns the sum of the squares of the arguments.

##### Syntax

**SUMSQ(number1, number2, ...)**, where:

**number1, number2, ...** are arguments for which you want the sum of the squares. You can also use a single array or a reference to an array instead of arguments separated by commas.

#### [4.1.4.6.6.236 SUMX2MY2](#)

Returns the sum of the difference of squares of corresponding values in two arrays.

#### Syntax

**SUMX2MY2(array\_x, array\_y)**, where:

**array\_x** is the first array or range of values.

**array\_y** is the second array or range of values.

#### Remarks

- If an array or reference argument contains text, logical values or empty cells, those values are ignored; however, cells with the value zero are included.
- The equation for the sum of the difference of squares is,

$$SUMX2MY2 = \sum (x^2 - y^2)$$

#### [4.1.4.6.6.237 SUMX2PY2](#)

Returns the sum of the sum of squares of corresponding values in two arrays. The sum of the sum of squares is a common term in many statistical calculations.

#### Syntax

**SUMX2PY2(array\_x, array\_y)**, where:

**array\_x** is the first array or range of values.

**array\_y** is the second array or range of values.

## Remarks

- If an array or reference argument contains text, logical values or empty cells, those values are ignored; however, cells with the value zero are included.
- The equation for the sum of squares is,

$$SUMX2PY2 = \sum(x^2+y^2)$$

### 4.1.4.6.6.238 SUMXMY2

Returns the sum of squares of differences of corresponding values in two arrays.

## Syntax

**SUMXMY2(array\_x, array\_y)**, where:

**array\_x** is the first array or range of values.

**array\_y** is the second array or range of values.

## Remarks

- If an array or reference argument contains text, logical values or empty cells, those values are ignored; however, cells with the value zero are included.
- The equation for the sum of squared differences is,

$$SUMXMY2 = \sum(x-y)^2$$

### 4.1.4.6.6.239 SYD

Returns the sum-of-years' digits depreciation of an asset for a specified period.

## Syntax

**SYD(cost, salvage, life, per)**, where:

**cost** is the initial cost of the asset.

**salvage** is the value at the end of the depreciation (sometimes called the salvage value of the asset).

**life** is the number of periods over which the asset is depreciated (sometimes called the useful life of the asset).

**per** is the period and must use the same units as life.

## Remarks

- SYD is calculated as follows,

$$\frac{(cost - salvage) * (life - per + 1) * 2}{(life)(life + 1)}$$

### 4.1.4.6.6.240 T

The T function tests whether the given value is text or not. If the given value is text, then it returns the given text. Otherwise, the function returns an empty text string.

## Syntax

The syntax of the T function is

**=T( value )**

value – Required. This is a value to be checked.

## Remarks:

If the value is not a number or logical value, then the function returns an empty string.

## Example:

FORMULA	RESULT
=T("SYNCFUSION")	SYNCFUSION
=T(TRUE)	
=T(10)	

#### [4.1.4.6.6.241 TAN](#)

Returns the tangent of the given angle.

#### Syntax

**TAN(number)**, where:

**number** is the angle in radians for which, you want the tangent.

#### [4.1.4.6.6.242 TANH](#)

Returns the hyperbolic tangent of a number.

#### Syntax

**TANH(number)**, where:

**number** is any real number.

#### Remarks

- The formula for the hyperbolic tangent is,

$$\tanh(z) = \frac{\sinh(z)}{\cosh(z)}$$

#### [4.1.4.6.6.243 TEXT](#)

Converts a value to text in a specific number format.

#### Syntax

**TEXT(value, format\_text)**, where:

**value** is a numeric value a formula that evaluates to a numeric value or a reference to a cell containing a numeric value.

**format\_text** is a number format in text form in the Category box on the Number tab in the Format Cells dialog box.

#### [4.1.4.6.6.244 TIME](#)

Returns the decimal number for a particular time.

The decimal number returned by TIME is a value ranging from 0 (zero) to 0.99999999, representing the times from 0:00:00 (12:00:00 A.M.) to 23:59:59 (11:59:59 P.M.).

#### Syntax

**TIME(hour, minute, second)**, where:

**hour** is a number from 0 (zero) to 23 representing the hour.

**minute** is a number from 0 to 59 representing the minute.

**second** is a number from 0 to 59 representing the second.

#### [4.1.4.6.6.245 TIMEVALUE](#)

Returns the decimal number of the time represented by a text string. The decimal number is a value ranging from 0 (zero) to 0.99999999, representing the times from 0:00:00 (12:00:00 A.M.) to 23:59:59 (11:59:59 P.M.).

#### Syntax

**TIMEVALUE(time\_text)**, where:

**time\_text** is a text string that represents a time as a formatted string; for example, "6:45 PM" and "18:45" text strings within quotation marks that represent time.

#### Remarks

- Date information in time\_text is ignored.

#### [4.1.4.6.6.246 TODAY](#)

Returns the serial number of the current date. The serial number is the number of days since Jan 1, 1900.

#### Syntax

**TODAY()**

#### Remarks

- Dates are stored as sequential serial numbers so that they can be used in calculations. By default, January 1, 1900 is serial number 1 and January 1, 2008 is serial number 39448 because it is 39,448 days after January 1, 1900.

#### [4.1.4.6.6.247 Trim](#)

The Trim function returns a text value with the leading and trailing spaces removed.

#### Syntax:

**Trim(text)**

where:

**text** is the text value for which you want to remove the leading and the trailing spaces.

#### [4.1.4.6.6.248 TRIMMEAN](#)

Returns the mean of the interior of a data set. TRIMMEAN calculates the mean taken by excluding a percentage of data points from the top and bottom tails of a data set.

#### Syntax

**TRIMMEAN(array, percent)**, where:

**array** is the array or range of values to trim and average.

**percent** is the fractional number of data points to exclude from the calculation. For example, if percent = 0.2, 4 points are trimmed from a data set of 20 points ( $20 \times 0.2$ ): 2 from the top and 2 from the bottom of the set.

#### Remarks

- Percent must be  $\geq 0$  and  $\leq 1$ .
- TRIMMEAN rounds off the number of excluded data points down to the nearest multiple of 2. If percent = 0.1, 10 percent of 30 data points equals 3 points. For symmetry, TRIMMEAN excludes a single value from the top and bottom of the data set.

#### [4.1.4.6.6.249 TRUNC](#)

The Trunc function truncates a supplied number to a specified number of decimal places.

#### Syntax

**TRUNC(number, num\_digits)**, where:

**number** is the number you want to truncate.

**num\_digits** is an optional argument that specifies the number of decimal places to truncate the supplied number to. The default value is 0.

#### Remarks

- TRUNC and INT are similar in that both return integers. TRUNC removes the fractional part of the number. INT rounds numbers down to the nearest integer based on the value of the fractional part of the number. INT and TRUNC are different only when using negative numbers: TRUNC(-4.3) returns -4 but, INT(-4.3) returns -5 because -5 is the lower number.

#### [4.1.4.6.6.250 True](#)

The **True** function returns the logical value when the given sting value is true.

#### Syntax:

**True(stringvalue)**

where:

**stringvalue** is to provide any text value or empty string.

#### [4.1.4.6.6.251 Upper](#)

The Upper function converts all characters in a text string to uppercase.

#### Syntax:

## **Upper(text)**

where:

**text** is the string you want to convert to uppercase.

### [4.1.4.6.6.252 VALUE](#)

Converts a text string that represents a number to a number.

## **Syntax**

**VALUE(text)**, where:

**text** is the text enclosed in quotation marks or a reference to a cell containing the text you want to convert.

## **Remarks**

- Text can be in any of the constant number, date or time formats recognized by Essential Calculate.
- You do not generally need to use the VALUE function in a formula, as the text is automatically converted to numbers as necessary.

### [4.1.4.6.6.253 VAR](#)

Estimates variance based on a sample.

## **Syntax**

**VAR(number1, number2, ...)**, where:

**number1, number2, ...** are arguments corresponding to a sample of a population.

## **Remarks**

- VAR assumes that its arguments are a sample of the population. If your data represents the entire population, then compute the variance using VARP.

- Logical values such as True, False and text are ignored. If logical values and text must not be ignored, use the VARA worksheet function.
- VAR uses the following formula,

$$\frac{\sum(x - \bar{x})^2}{(n-1)}$$

where  $\bar{x}$  is the sample mean `AVERAGE(number1,number2,...)` and  $n$  is the sample size.

#### [4.1.4.6.6.254 VARA](#)

Estimates variance based on a sample. In addition to numbers and text, logical values such as True and False are included in the calculation.

#### Syntax

**VARA(value1, value2, ...)**, where:

**value1, value2, ...** are value arguments corresponding to a sample of a population.

#### Remarks

- VARA assumes that its arguments are a sample of the population. If your data represents the entire population, you must compute the variance using VARPA.
- Arguments that contain True evaluate as 1; arguments that contain text or False evaluate as 0 (zero). If the calculation must not include text or logical values, use the VAR worksheet function instead.
- VARA uses the following formula,

$$\frac{\sum(x - \bar{x})^2}{(n-1)}$$

where  $\bar{x}$  is the sample mean `AVERAGE(value1,value2,...)` and  $n$  is the sample size.

#### [4.1.4.6.6.255 VARP](#)

Calculates variance based on the entire population.

## Syntax

**VARP(number1, number2, ...)**, where:

**number1, number2, ...** are number arguments corresponding to a population.

## Remarks

- VARP assumes that its arguments are the entire population. If your data represents a sample of the population, then compute the variance using VAR.
- The equation for VARP is,

$$\frac{\sum(x - \bar{x})^2}{n}$$

where  $\bar{x}$  is the sample mean **AVERAGE(number1,number2,...)** and  $n$  is the sample size.

### [4.1.4.6.6.256 VARPA](#)

Calculates variance based on the entire population. In addition to numbers and text, logical values such as True and False are also included in the calculation.

## Syntax

**VARPA(value1, value2, ...)**, where:

**value1, value2, ...** are arguments corresponding to a population.

## Remarks

- VARPA assumes that its arguments are the entire population. If your data represents a sample of the population, you must compute the variance using VARA.
- Arguments that contain True evaluate as 1; arguments that contain text or False evaluate as 0 (zero). If the calculation does not include text or logical values, use the VARP worksheet function instead.
- The equation for VARPA is,

$$\frac{\sum(x - \bar{x})^2}{n}$$

where x is the sample mean AVERAGE(value1,value2,...) and n is the sample size.

#### 4.1.4.6.6.257 VDB

Returns the depreciation of an asset for any period you specify, including partial periods, using the double-declining balance method or some other method you specify. VDB stands for variable declining balance.

#### Syntax

**VDB(cost, salvage, life, start\_period, end\_period, factor, no\_switch)**, where:

**cost** is the initial cost of the asset.

**salvage** is the value at the end of the depreciation (sometimes called the salvage value of the asset).

**life** is the number of periods over which the asset is depreciated (sometimes called the useful life of the asset).

**start\_period** is the starting period for which you want to calculate the depreciation. start\_period must use the same units as life.

**end\_period** is the ending period for which you want to calculate the depreciation. end\_period must use the same units as life.

**factor** is the rate at which, the balance declines. If factor is omitted, it is assumed to be 2 (the double-declining balance method).

**no\_switch** is a logical value specifying whether to switch to straight-line depreciation when depreciation is greater than the declining balance calculation.

- If no\_switch is True, straight-line depreciation is not used even when the depreciation is greater than the declining balance calculation.
- If no\_switch is False or omitted, straight-line depreciation is used when depreciation is greater than the declining balance calculation.

All arguments except no\_switch must be positive numbers.

#### [4.1.4.6.6.258 VLOOKUP](#)

Searches for a value in the left most column of a table and then returns a value in the same row from a column you specify in the table. Use VLOOKUP instead of HLOOKUP when your comparison values are located in a column to the left of the data you want to find.

The V in VLOOKUP stands for "Vertical."

#### Syntax

**VLOOKUP(lookup\_value, table\_array, col\_index\_num, range\_lookup)**, where:

**lookup\_value** is the value to be found in the first column of the array. Lookup\_value can be a value, a reference or a text string.

**table\_array** is the table of information in which data is looked up. Use a reference to a range or a range name.

- If range\_lookup is True, the values in the first column of the table\_array must be placed in ascending order: ..., -2, -1, 0, 1, 2, ..., A-Z, False, True; otherwise VLOOKUP may not give the correct value. If range\_lookup is False, table\_array does not need to be sorted.
- The values in the first column of the table\_array can be text, numbers or logical values.
- Uppercase and lowercase text are equivalent.

**col\_index\_num** is the column number in the table\_array from which the matching value must be returned. A col\_index\_num of 1 returns the value in the first column of the table\_array; a col\_index\_num of 2 returns the value in the second column of the table\_array, and so on.

**range\_lookup** is a logical value that specifies whether you want VLOOKUP to find an exact match or an approximate match. If True or omitted, an approximate match is returned. In other words, if an exact match is not found, the next largest value that is less than the lookup\_value is returned.

#### Remarks

- If VLOOKUP can't find a lookup\_value and the range\_lookup is True, it uses the largest value that is less than or equal to the lookup\_value.

#### [4.1.4.6.6.259 WEEKDAY](#)

Returns the day of the week corresponding to a date. The day is given as an integer, ranging from 1 (Sunday) to 7 (Saturday) by default.

#### Syntax

**WEEKDAY(serial\_number, return\_type)**, where:

**serial\_number** is a sequential number that represents the date of the day you are trying to find. Dates should be entered by using the DATE function or as results of other formulas or functions. For example, use DATE(2008,5,23) for the 23rd day of May 2008.

**return\_type** is a number that determines the type of return value.

<b>Return_type is</b>	<b>Number Returned</b>
1 or omitted	Numbers 1 (Sunday) through 7 (Saturday).
2	Numbers 1 (Monday) through 7 (Sunday).
3	Numbers 0 (Monday) through 6 (Sunday).

### **Remarks**

- Dates are stored as sequential serial numbers so that they can be used in calculations. By default, January 1, 1900 is serial number 1 and January 1, 2008 is serial number 39448 because it is 39,448 days after January 1, 1900.

#### [\*\*4.1.4.6.6.260 WEIBULL\*\*](#)

Returns the Weibull distribution.

### **Syntax**

**WEIBULL(x, alpha, beta, cumulative)**, where:

**x** is the value at which to evaluate the function.

**alpha** is a parameter to the distribution.

**beta** is a parameter to the distribution.

**cumulative** determines the form of the function.

### **Remarks**

- X must be  $\geq 0$ .
- Alpha and beta must  $> 0$ .

- The equation for the Weibull cumulative distribution function is,

$$F(x; \alpha, \beta) = 1 - e^{-(\frac{x}{\beta})^\alpha}$$

- The equation for the Weibull probability density function is,

$$f(x; \alpha, \beta) = \frac{\alpha}{\beta} x^{\alpha-1} e^{-(\frac{x}{\beta})^\alpha}$$

- When alpha = 1, WEIBULL returns the exponential distribution with,

$$\lambda = \frac{1}{\beta}$$

#### [4.1.4.6.6.261 WEIBULL.DIST](#)

The WEIBULL.DIST function returns the Weibull distribution.

##### **Syntax:**

**WEIBULL.DIST(x,alpha,beta,cumulative)** where:

- x is the value at which to evaluate the function.
- alpha is a parameter of the distribution.
- beta is a parameter of the distribution.
- cumulative determines the form of the function.

##### **Remarks:**

#NUM! - occurs if x is less than zero.

#VALUE! - occurs if x is non-numeric.

#VALUE! - occurs if alpha is non-numeric.

#NUM! - occurs if alpha is equal to or less than zero.

#VALUE! - occurs if beta is non-numeric.

#NUM! - occurs if beta is equal to or less than zero.

#### [4.1.4.6.6.262 XIRR](#)

The XIRR function computes the internal rate-of-return for a schedule of possibly non-periodic cash flows.

**Syntax:**

**XIRR(cashflow, datelist, value)**

where:

**cashflow** is the range of cash flow.

**datelist** is the list of serial number of the corresponding date values.

**value** is an initial guess integer value which reflects in the result of the function.

[\*\*4.1.4.6.6.263 XOR\*\*](#)

The XOR function returns the exclusive OR for the given arguments.

**Syntax**

**XOR (logical\_value1, logical\_value2,...)**

**Logical\_value1:** Required. This can be either TRUE or FALSE, and can be logical values, arrays, or references.

**Notes**

If the given arguments do not have the logical values, XOR returns the #VALUE! error value.

**Example**

FORMULA	RESULT
= XOR( 5>0, 7<9 )	FALSE
=XOR(3>12, 4>6)	TRUE

[\*\*4.1.4.6.6.264 YEAR\*\*](#)

Returns the year corresponding to a date. The year is returned as an integer in the range 1900-9999.

**Syntax**

**YEAR(serial\_number)**, where:

**serial\_number** is the date of the year you want to find. Dates should be entered by using the DATE function or as results of other formulas or functions. For example, use DATE(2002,11,12) for the 12th day of November 2002.

## Remarks

- Dates are stored as sequential serial numbers so that they can be used in calculations. By default, January 1, 1900 is serial number 1 and January 1, 2008 is serial number 39448 because it is 39,448 days after January 1, 1900.

### [4.1.4.6.6.265 ZTEST](#)

Returns the one-tailed probability-value of a z-test.

## Syntax

**ZTEST(array, u0, sigma)**, where:

**array** is the array or range of data against which to test u0

**u0** is the value to test.

**sigma** is the population (known) standard deviation. If omitted, the sample standard deviation is used.

## Remarks

- ZTEST is calculated as follows when sigma is not omitted,

$$ZTEST(\text{array}, \mu_0) = 1 - NORMSDIST\left(\frac{(\bar{x} - \mu_0)\sqrt{n}}{\text{sigma}}\right)$$

- or when sigma is omitted,

$$ZTEST(\text{array}, \mu_0) = 1 - NORMSDIST\left(\frac{(\bar{x} - \mu_0)\sqrt{n}}{s}\right)$$

where x is the sample mean AVERAGE(array); s is the sample standard deviation STDEV(array); and n is the number of observations in the sample COUNT(array).

#### 4.1.4.6.7 Cross Sheet Reference

Essential Grid control supports cross sheet references. A formula cell can be defined with values from another grid by using cross sheet references. In this case, multiple grids can either be in worksheet format or multiple grids can be laid out in a form. The following screen shot shows multiple grids in worksheet format.

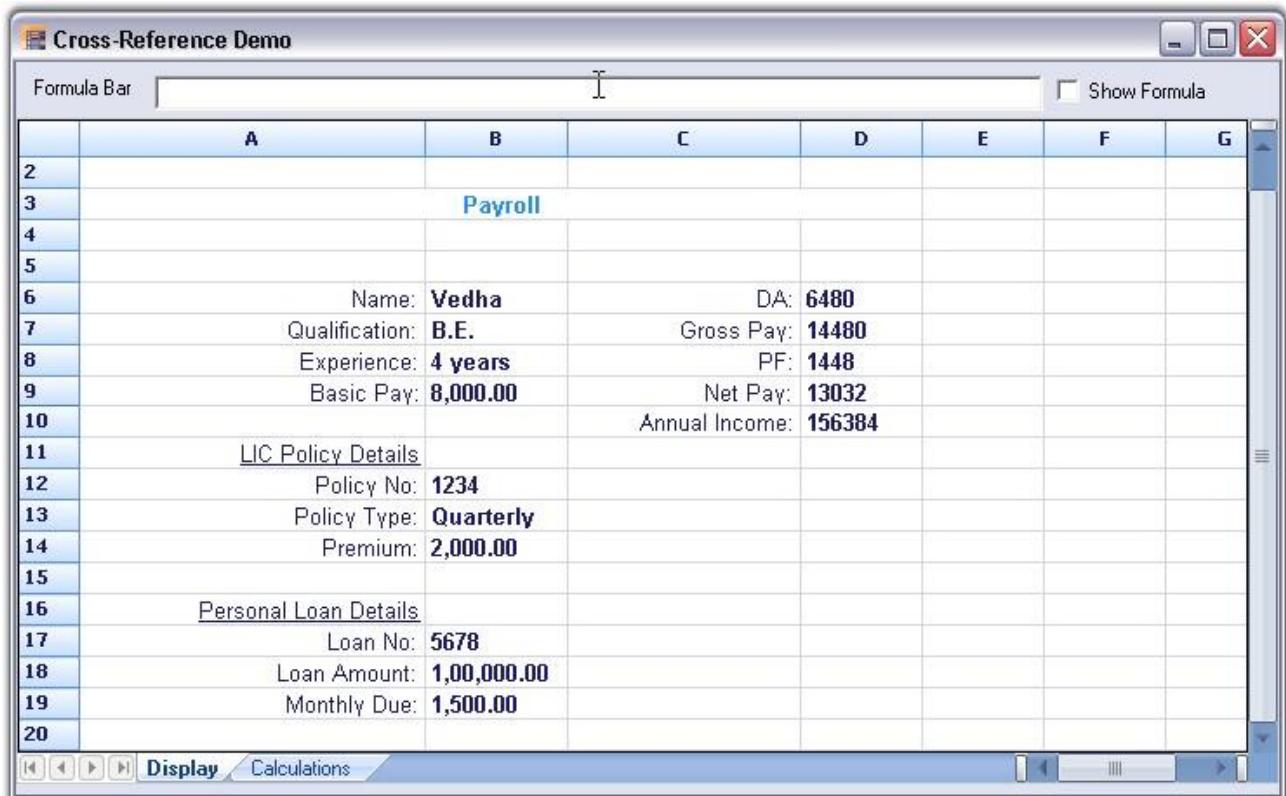


Figure 133: Multiple Grids in Worksheet Format

The following screen shot shows multiple grids laid out in the form.

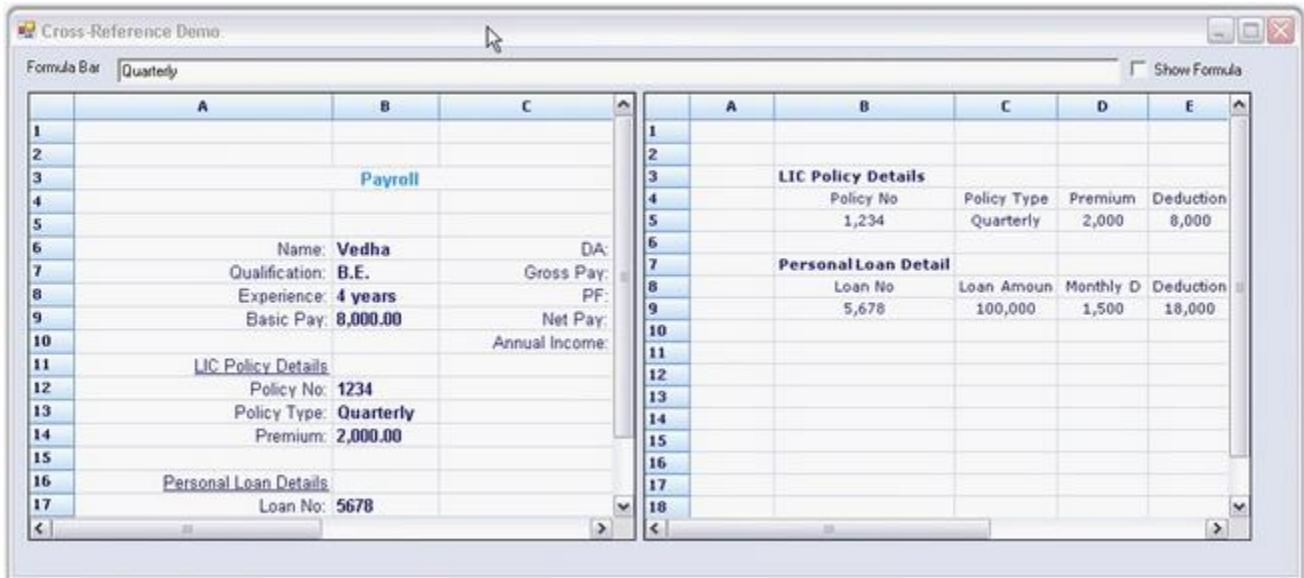


Figure 134: Form with Multiple Grids

## Example

The following example implements this feature with two sheets-'Display' and 'Calculations'.

- The 'Display' sheet accepts parameters such as Salary details, Insurance Policy details, Loan details, and then computes the annual income of the given employee.
- The 'Calculations' sheet computes tax deductions for an insurance policy and personal loan.

This process is explained in the following steps.

**Step 1**-Register the grid controls as shown in the following code so that they can be referenced by using a formula.

### [C#]

```
// Registering grid controls as separate sheets for cross-reference.
int sheetFamilyID = GridFormulaEngine.CreateSheetFamilyID();
GridFormulaEngine.RegisterGridAsSheet("Display", this.gridDisplay.Model,
sheetFamilyID);
GridFormulaEngine.RegisterGridAsSheet("Calculate", this.gridCalculations.Model,
sheetFamilyID);
```

### [VB .NET]

```
' Registering grid as separate sheets for cross-reference.  
Dim sheetFamilyID As Integer = GridFormulaEngine.CreateSheetFamilyID()  
GridFormulaEngine.RegisterGridAsSheet("Display", Me.gridDisplay.Model,  
sheetFamilyID)  
GridFormulaEngine.RegisterGridAsSheet("Calculate", Me.gridCalculations.Model,  
sheetFamilyID)
```

**Step 2**-Set values for the Display Grid.

**[C#]**

```
this.gridDisplay[12, 1].Text = "Policy Premium Amount";  
this.gridDisplay[12, 2].Text = "2,000";  
this.gridDisplay[13, 1].Text = "Personal Loan Monthly Due";  
this.gridDisplay[13, 2].Text = "1,500";
```

**[VB.NET]**

```
Me.gridDisplay(12, 1).Text = "Policy Premium Amount"  
Me.gridDisplay(12, 2).Text = "2,000"  
Me.gridDisplay(13, 1).Text = "Personal Loan Monthly Due"  
Me.gridDisplay(13, 2).Text = "1,500"
```

**Step 3**-Compute the values for the Calculations Grid by using the values in the Display Grid.

**[C#]**

```
// Cross Sheet References.  
this.gridCalculations[row, col].CellType = GridCellTypeName.FormulaCell;  
this.gridCalculations[row, col].Text = "=Display!B12";  
this.gridCalculations[row, col + 1].CellType = GridCellTypeName.FormulaCell;  
this.gridCalculations[row, col + 1].Text = "=Display!B13";
```

**[VB.NET]**

```
' Cross Sheet References.  
Me.gridCalculations(row, col).CellType = GridCellTypeName.FormulaCell  
Me.gridCalculations(row, col).Text = "=Display!B12"  
Me.gridCalculations(row, col + 1).CellType = GridCellTypeName.FormulaCell  
Me.gridCalculations(row, col + 1).Text = "=Display!B13"
```

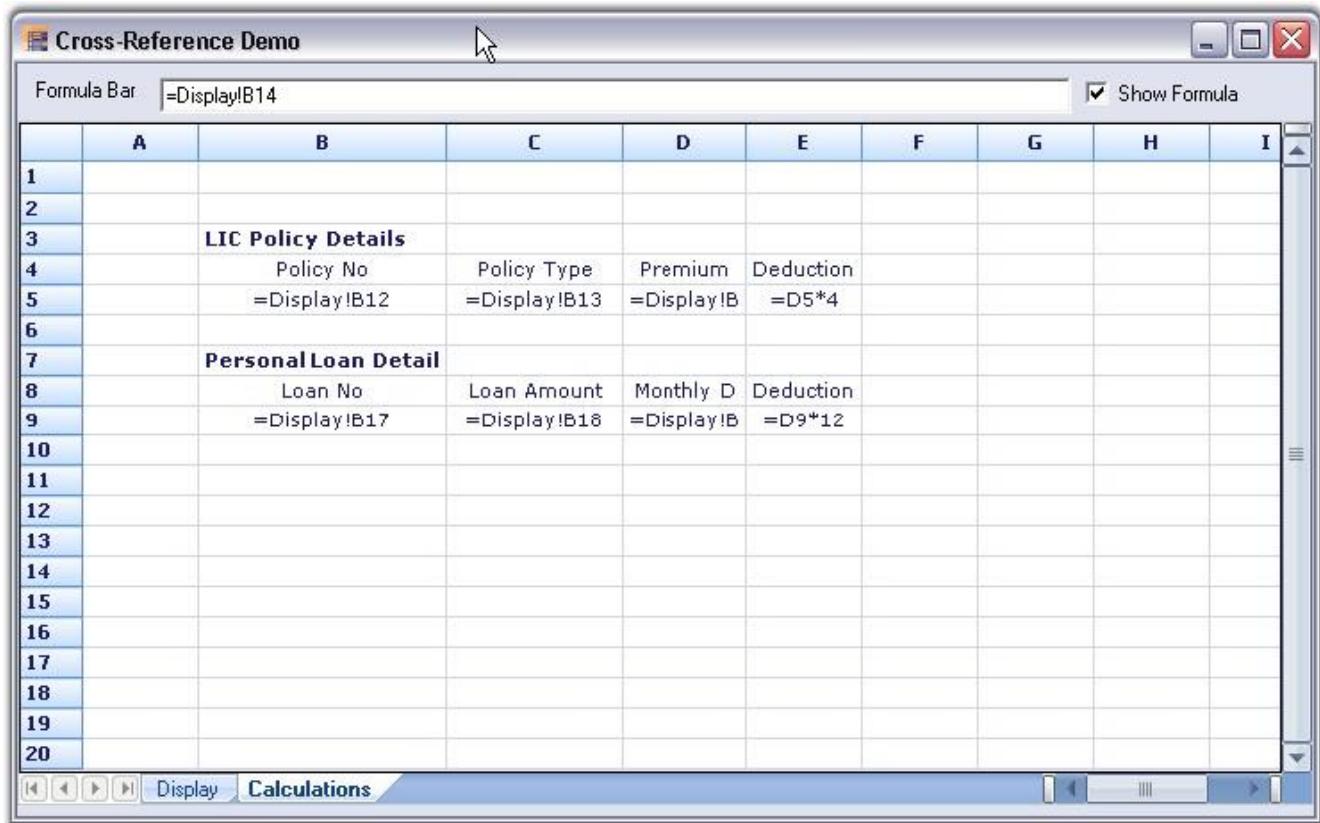


Figure 135: Cross Sheet References

After computing the values, the Display sheet displays the Annual Income and Calculations sheet displays the Tax Deductions.

A sample which demonstrates this feature is available in the following sample installation path.

**C:\Syncfusion\EssentialStudio\Version  
Number\Windows\Grid.Windows\Samples\2.0\Formula Support\Cross-Reference Demo**

#### 4.1.4.6.8 Named Ranges

Essential Grid supports named ranges along with Grid Formula Engine. Named ranges let the users to set up names for expressions or ranges, and then use these names in formulas. For example, if you name the range, "B4:B12" as "Expenses", you can use formulas like =Sum(Expenses) instead of =Sum(B4:B12).

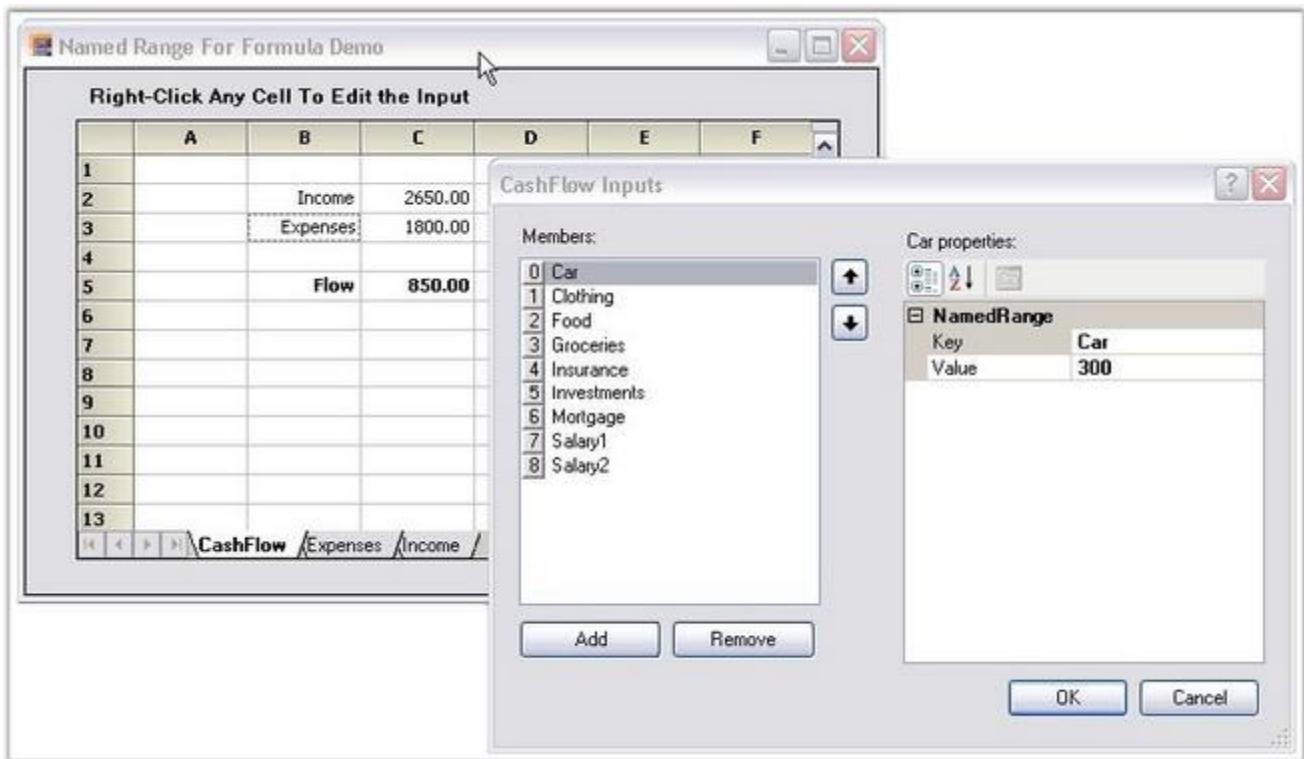


Figure 136: Named Ranges in Grid Control

### Example

The following example illustrates the creation and usage of named ranges in Grid controls.

You can call the **AddNamedRange** method by using an instance of Grid Formula Engine and pass two parameters-**Name** and **Value** to be set for the range.

#### [C#]

```
GridFormulaEngine engine;
this.engine =
((GridFormulaCellModel)gridCashFlow.Model.CellModels["FormulaCell"]).Engine;
this.engine.AddNamedRange("Car", "300");
```

#### [VB .NET]

```
Dim engine As GridFormulaEngine
Me.engine = CType(gridCashFlow.Model.CellModels("FormulaCell"),
GridFormulaCellModel)).Engine
Me.engine.AddNamedRange("Car", "300")
```

You can edit the named ranges by using the **NamedRange Collection Editor**. The following code uses the ShowNamedRangesDialog method to display the editor.

[C#]

```
GridFormulaNamedRangesEditHelper.ShowNamedRangesDialog(this.engine);
```

[VB .NET]

```
GridFormulaNamedRangesEditHelper.ShowNamedRangesDialog(Me.engine)
```

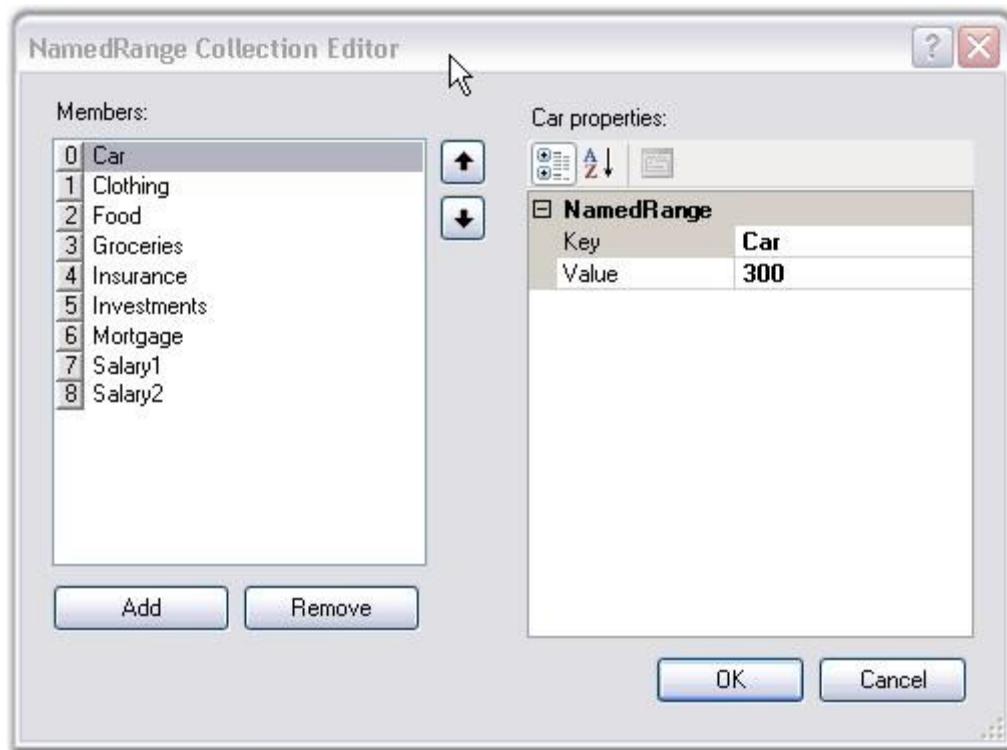


Figure 137: NamedRange Collection Editor

In the above dialog box, you will notice the named ranges (Members) are displayed in the left pane and their corresponding properties to the right pane (Properties). You can select a Named Range and edit its value as follows.

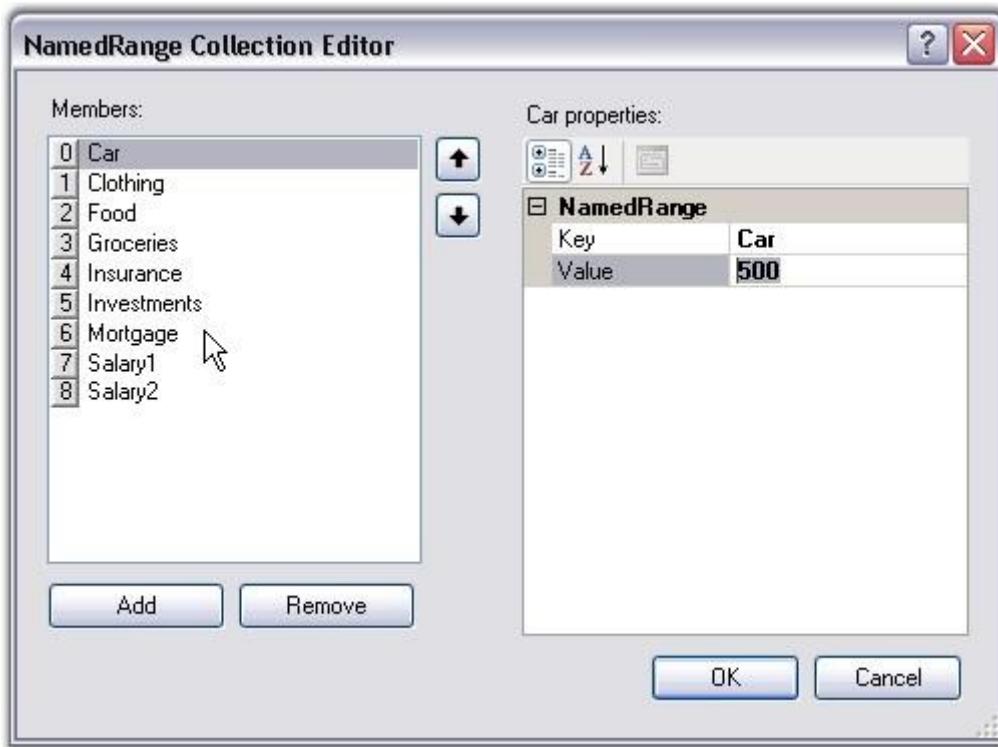


Figure 138: Editing Named Ranges

You can also edit the title of this editor by handling the `ShowingNamedRangesDialog` event. The following code example illustrates this.

[C#]

```
GridFormulaNamedRangesEditHelper.ShowingNamedRangesDialog += new
ControlEventHandler(helper_ShowingNamedRangesDialog);

// Event handler to change the title of NamedRange Collection Editor
// dialog box.
private void helper_ShowingNamedRangesDialog(object sender,
ControlEventArgs e)
{
    Form f = e.Control as Form;
    if(f != null)
    {
        // Set the title for the dialog box.
        f.Text = "CashFlow Inputs";
    }
}
```

[VB .NET]

```
Private GridFormulaNamedRangesEditHelper.ShowingNamedRangesDialog +=  
    New ControlEventHandler(AddressOf helper_ShowingNamedRangesDialog)  
  
    ' Event handler to change the title of NamedRange Collection Editor  
    dialog box.  
Private Sub helper_ShowingNamedRangesDialog(ByVal sender As Object,  
    ByVal e As ControlEventArgs)  
    Dim f As Form = TryCast(e.Control, Form)  
    If f IsNot Nothing Then  
  
        ' Set the title for the dialog box.  
        f.Text = "CashFlow Inputs"  
    End If  
End Sub
```

Custom Title for the ShowNamedRange dialog is set to "CashFlow Inputs" by using the above code.

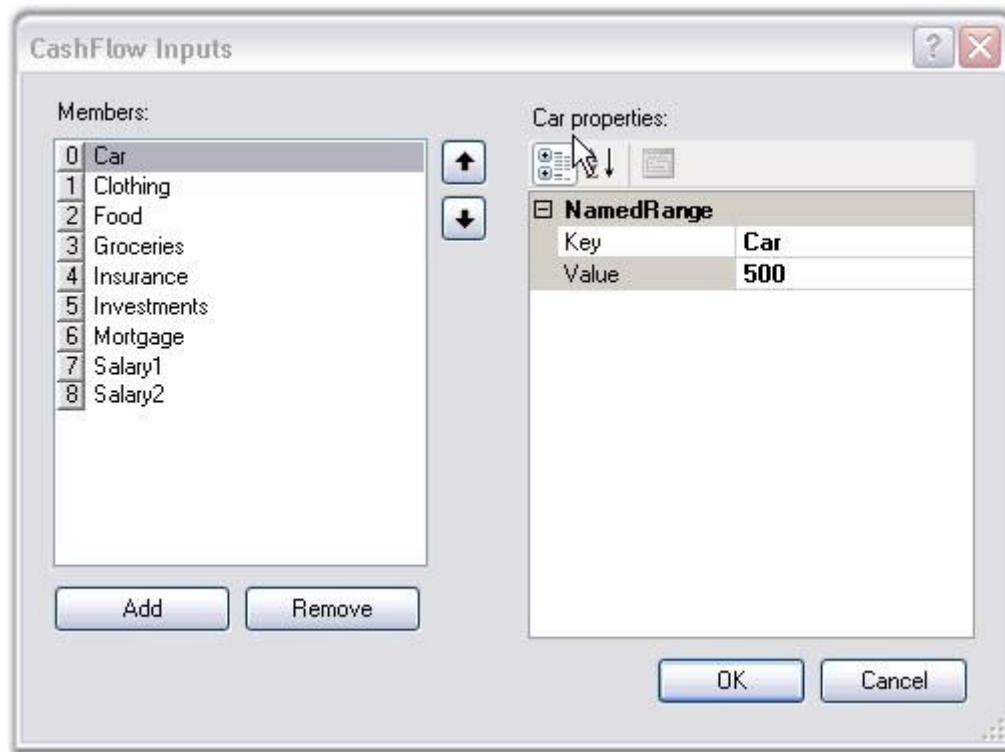


Figure 139: Custom Title set for the NamedRange Collection Editor

 **Note:** The following sample illustrates the use of Cross Sheet References and Named Ranges with the Grid Formula Engine.

<Install Location>\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Windows\Samples\2.0\Formula Support\Named Range For Formula Demo

#### 4.1.4.7 Populating a Grid Control

There are several ways of using Grid control to display data. For instance, you can move your data directly into an Essential Grid and let the underlying grid manage this data for you, or another way would be to set the datasource member of Essential Grid, thus binding the grid. Still another method is, to use the Essential Grid in a pure virtual mode, whereby you can handle certain events to provide data to the Essential Grid whenever it is in demand.

Out of these three ways, the latter two ways of providing data to an Essential Grid are discussed in more detail (in their own sections of) in this user's guide. In this section, you will learn how to get data stored directly into an Essential Grid's internal storage, so that the grid is able to maintain the data for you, and also how to populate a Grid control from a two-dimensional integer array.

##### 4.1.4.7.1 The Grid Control Indexer Technique

To place data into a grid, you need to loop through all the rows and columns, setting the **GridStyleInfo.CellValue** property for each cell. This technique is fine for small grids and is not a real drawback. It does have the overhead of firing events that are normally associated with a change in a cell's **GridStyleInfo** object. For larger grids, the overhead associated with such events is likely to be noticed by users only during population. One thing to note is that you may need to turn off the Grid's Undo/Redo support so that users cannot undo your initial population of the grid.

[C#]

```
// Turn off undo.  
this.gridControl1.CommandStack.Enabled = false;  
  
// Prevent the grid from redrawing for each change.  
this.gridControl1.BeginUpdate();
```

```
this.gridControl1.RowCount = this.numArrayRows;
this.gridControl1.ColCount = this.numArrayCols;

for (int i = 0; i < this.numArrayRows; ++i)
{
    for(int j = 0; j < this.numArrayCols; ++j)
        this.gridControl1[i+1, j+1].CellValue = this.intArray[i,j];
}
this.gridControl1.EndUpdate();
this.gridControl1.Refresh();
```

**[VB.NET]**

```
' Turn off undo.
Me.gridControl1.CommandStack.Enabled = False

' Prevent the grid from redrawing for each change.
Me.gridControl1.BeginUpdate()

Me.gridControl1.RowCount = Me.numArrayRows
Me.gridControl1.ColCount = Me.numArrayCols

Dim i As Integer
Dim j As Integer
For i = 1 To Me.numArrayRows
    For j = 1 To Me.numArrayCols
        Me.gridControl1(i+1, j+1).CellValue = Me.intArray(i, j)
    Next j
Next i

Me.gridControl1.EndUpdate()
Me.gridControl1.Refresh()
```

#### 4.1.4.7.2 The GridControl.PopulateValues Method

To overcome the cell-by-cell event overhead intrinsic, Grid control has a **PopulateValues** method. This method is a member of the **GridControl** class which is in the indexer population technique and which was discussed in the Grid Control Indexer Technique section. This method will allow you to pass in a range of cells and a data source of the type object. With this information, the method will then use techniques that will bypass the event overhead to retrieve data from the **datasource** object in a manner that depends upon the object type and will move it into the range.

The basic types that can be passed into the datasource parameter include `IListSource`, `Array` and `IList`. This includes things like a `DataTable` and `DataView`, since a `DataTable` implements `IListSource` and a `DataView` implements `IList`.

Note that when you use the `GridControl.PopulateValues` method, the data values are copied from your data source and placed into the Essential Grid object. Thus, this is an entirely different concept than binding a grid to a data source. In such cases, the data is not moved into the grid object, but instead is provided on demand from the data source to the grid. So, the grid never stores any values in a databound grid. When you use the `PopulateValues` method, the data is actually copied from the data source and is placed in the grid's internal storage.

**[C#]**

```
this.gridControl1.BeginUpdate();
this.gridControl1.RowCount = this.numArrayRows;
this.gridControl1.ColCount = this.numArrayCols;

// Call PopulateValues Method to move values from a given data source
// (this.initArray) into the Grid Range specified.
this.gridControl1.Model.PopulateValues(GridRangeInfo.Cells(1, 1,
this.numArrayRows, this.numArrayCols), this.intArray);
this.gridControl1.EndUpdate();
this.gridControl1.Refresh();
```

**[VB .NET]**

```
Me.gridControl1.BeginUpdate()
Me.gridControl1.RowCount = Me.numArrayRows
Me.gridControl1.ColCount = Me.numArrayCols

' Call PopulateValues Method to move values from a given data source
' (this.initArray) into the Grid Range specified.
Me.gridControl1.Model.PopulateValues(GridRangeInfo.Cells(1, 1,
Me.numArrayRows, Me.numArrayCols), Me.intArray)
Me.gridControl1.EndUpdate()
Me.gridControl1.Refresh()
```

To see a program sample that uses these techniques to populate a grid, look at the Grid Population Demo sample that ships with the product. It allows you to time the different population methods including the using of a grid virtually. For small grids, the three techniques are comparable. But, as you increase the grid size, the virtual grid timing values remain the same, whereas the other two methods vary depending upon the number of data points. In general, the population of a grid by using the indexer technique is about a factor of ten slower than using the `PopulateValues` method. And, depending upon the size of the grid, the virtual technique can be thousands of times quicker than the other techniques.

#### 4.1.4.7.3 Using the GridStyleInfo Class

By using the **GridRangeInfo** class and the properties of the **GridStyleInfo** class, you can write code illustrating how to enter values into a grid and how to affect the appearance of these displayed values. In the sections that follow, you will learn how to create a Grid object and place it on a form. Then you can set the style values by using the Grid classes discussed so far: **GridStyleInfo**, **GridRangeInfo** and **GridControl**.

##### 4.1.4.7.3.1 Setting GridStyleInfo Properties

There are several ways of setting the **GridStyleInfo** properties. If you want to set them for a particular cell, you need to use the row and column values as indexers on the **GridControl** object to retrieve the **GridStyleInfo** object that is associated with a particular cell. But, to change a **BaseStyle**, a **ColumnStyle** or a **RowStyle**, you will have to use different accessory methods to retrieve the style which is under consideration. In the code samples that follow, we will show you several ways of changing particular styles.



**Note:** *In this section, we are working with the cell-oriented Grid control which allows us to explicitly set individual cell and row properties. In the column-oriented Grid Data Bound Grid, explicitly setting individual cell and row properties is not supported. Instead, events are used to set these properties on demand. You can see samples in the Grid Data Bound Grid section.*

It comprises the following sections:

###### 4.1.4.7.3.1.1 Through Code

Given below is the code to help you create **GridStyleInfo** properties.

```
[C#]

// Add some text to cell (2,3).
gridControl1[2, 3].CellValue = "Essential";

// Create a GridStyleInfo.
GridStyleInfo style = new GridStyleInfo();
```

```
// Set some properties.  
style.BackColor = Color.Aquamarine;  
style.CellValue = "Grid";  
style.Font.Facename = "Verdana";  
style.Font.Size = 8.2f;  
style.Font.Bold = true;  
  
// Apply this style to several cells.  
this.gridControl1.ChangeCells(GridRangeInfo.Cells(3, 3, 4, 4), style);
```

**[VB.NET]**

```
' Add some text to cell (2,3).  
gridControl1(2, 3).CellValue = "Essential"  
  
' Create a GridStyleInfo.  
Dim style As GridStyleInfo  
style = New GridStyleInfo()  
  
' Set some properties.  
style.BackColor = Color.Aquamarine  
style.CellValue = "Grid"  
style.Font.Facename = "Verdana"  
style.Font.Size = 8.2!  
style.Font.Bold = True  
  
' Apply this style to several cells.  
Me.gridControl1.ChangeCells(GridRangeInfo.Cells(3, 3, 4, 4), style)
```

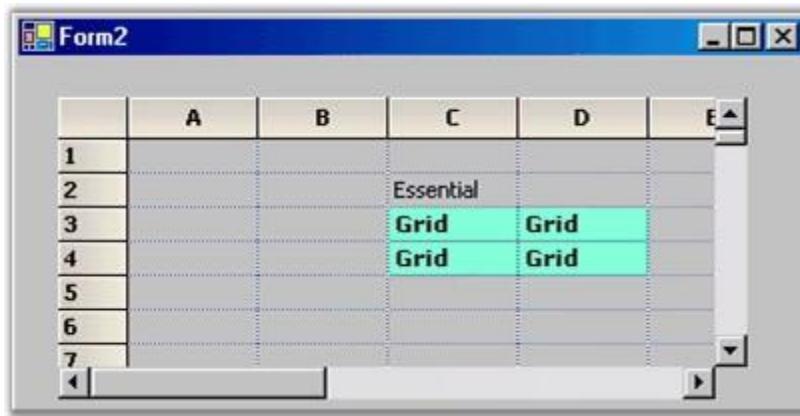


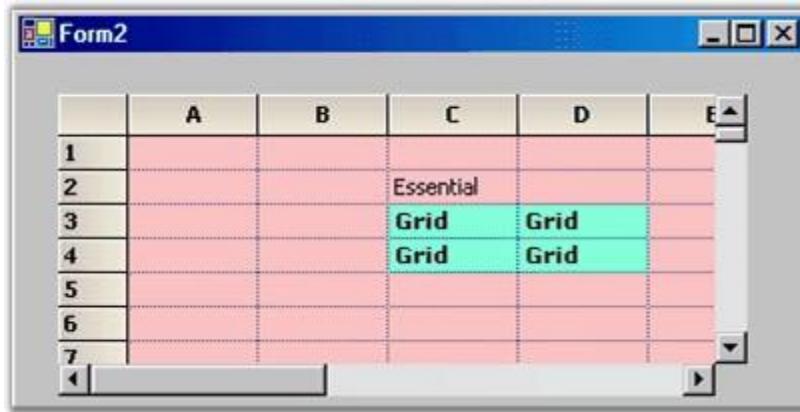
Figure 140: Result of the ChangeCells Call in the Previous Code Sample

**[C#]**

```
// Change the back color of the grid by using TableStyle.  
this.gridControl1.TableStyle.BackColor = Color.FromArgb(255, 192, 192);
```

**[VB.NET]**

```
' Change the back color of the grid using TableStyle.  
Me.gridControl1.TableStyle.BackColor = Color.FromArgb(255, 192, 192)
```



*Figure 141: Changing the TableStyle BackColor Property Adds a Rose Background for Cells Whose BackColor Was Not Explicitly Set*

**[C#]**

```
// Change a row style and a column style.  
this.gridControl1.RowStyle[3].TextColor = Color.Blue;  
this.gridControl1.RowStyle[3].CellValue = "Blue";  
this.gridControl1.ColumnStyle[3].TextColor = Color.Red;  
this.gridControl1.ColumnStyle[3].CellValue = "Red";
```

**[VB.NET]**

```
' Change a row style and a column style.  
Me.gridControl1.RowStyle(3).TextColor = Color.Blue  
Me.gridControl1.RowStyle(3).CellValue = "Blue"  
Me.gridControl1.ColumnStyle(3).TextColor = Color.Red  
Me.gridControl1.ColumnStyle(3).CellValue = "Red"
```

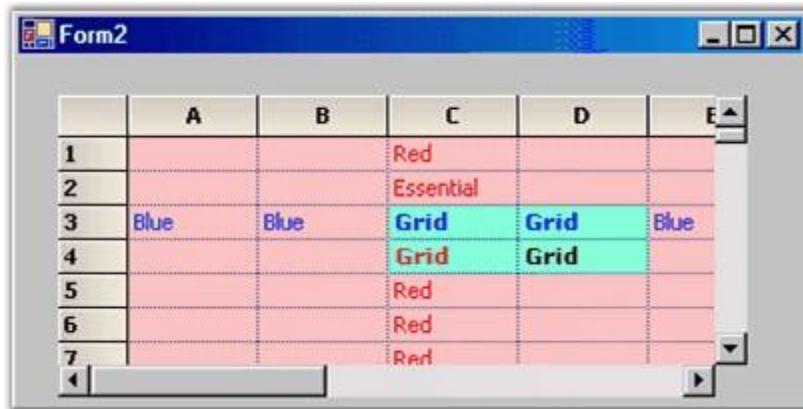


Figure 142: Row 3 with Blue Text and Column 3 With Red Text. Note the Row Attribute Takes Precedence Over the Column Attribute in Cell 3,3 as the TGxt is Blue

[C#]

```
// Change a base style, eg. Row Header.  
gridControl1.BaseStylesMap["Row Header"].StyleInfo.BackColor =  
Color.FromArgb(228, 255, 255);
```

[VB .NET]

```
' Change a base style, eg. Row Header.  
gridControl1.BaseStylesMap("Row Header").StyleInfo.BackColor =  
Color.FromArgb(228, 255, 255)
```



Figure 143: Row Headers No Longer Shaded Gray, Now Light Blue

#### 4.1.4.7.3.1.2 Through Designer

To edit cell styles from the designer you must select the grid on the design surface and then click the Toggle Interactive Mode verb shown at the bottom of the **PropertyGrid**. This will allow you to select the cells that are within the Grid control on the design surface. To change any of the style settings of the selected cells, you must first click the Cell Settings tool bar button at the top of the PropertyGrid. This will display the cell style settings that are within the PropertyGrid and will allow you to change them. The changes will affect the currently selected range or the current cell if no range is selected.

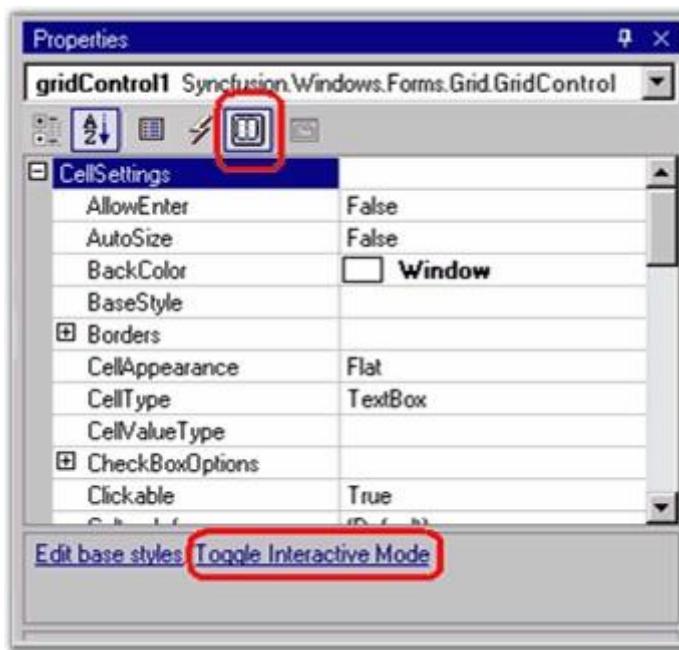


Figure 144: Toggle Interactive Mode Verb and CellSettings Button

#### 4.1.4.7.3.2 Creating a Grid Object

To add a Grid control to a form, you must create an instance of the Grid control, set the row and column count, then position it on your form.

It comprises the following sections:

#### 4.1.4.7.3.2.1 Through Designer

With the designer, you can drag-and-drop the control, size it, and then set a couple of properties.

- Drag a Grid control object from your toolbox and drop it on the form.
- Size and position it.
- Change the **RowCount** and **ColCount** values in the **PropertyGrid** for this control.

#### 4.1.4.7.3.2.2 Through Code

Given below is the code to help you create a grid.

##### [C#]

```
using Syncfusion.Windows.Forms.Grid;
        ....
// Create the Essential Grid.
private GridControl gridControl1;
        ....
this.gridControl1 = new GridControl();

// Set the number of rows and columns.
this.gridControl1.ColCount = 10;
this.gridControl1.RowCount = 100;

// Position it on the form.
this.gridControl1.Location = new System.Drawing.Point(20, 20);
this.gridControl1.Size = new System.Drawing.Size(344, 200);

// Add it to the forms' controls.
this.Controls.Add(this.gridControl1);
```

##### [VB.NET]

```
imports Syncfusion.Windows.Forms.Grid
        ....
' Create the Essential Grid.
Private WithEvents gridControl1 As GridControl
        ....
Me.gridControl1 = New GridControl()

' Set the number of rows and columns.
```

```
Me.gridControl1.ColCount = 10
Me.gridControl1.RowCount = 100

' Position it on the form.
Me.gridControl1.Location = New System.Drawing.Point(20, 15)
Me.gridControl1.Size = New System.Drawing.Size(344, 150)

' Add it to the forms' controls.
Me.Controls.Add(Me.gridControl1)
```

#### 4.1.4.8 Using the **ReadOnly** Attribute

There are several ways by which, you can prevent the user from making a change to the contents of a grid cell. If you set a cell's **GridStyleInfo.CellType** to "Static", the user will not be able to type in the cell. Another way to accomplish this is to use the Read-only attribute. This can be done on a grid-wide or cell-by-cell basis.

A Static cell will not allow the edit cursor to become visible. With a Read-only text box cell, the edit cursor may be visible. But, a Static cell can be pasted over or cleared by hitting the Delete key. Read-only cells cannot be pasted over or cleared. If you want a cell that will not show an edit cursor and which, cannot be pasted over or cleared, you must set the **CellType** to "Static" and also set the Read-only property to True.

##### 4.1.4.8.1 Setting **ReadOnly** Grid Wide

The property **GridControl.ReadOnly** or **GridDataBoundGrid.Model.ReadOnly** will allow you to set the **ReadOnly** behavior on a grid-wide basis.

###### [C#]

```
// Changes cannot be made to the Grid control.
this.gridControl1.ReadOnly = true;

// Changes cannot be made to the Grid Data Bound Grid.
this.gridDataBoundGrid1.ReadOnly = true;
```

###### [VB .NET]

```
' Changes cannot be made to the Grid control.  
Me.GridControl1.ReadOnly = True  
  
' Changes cannot be made to the Grid Data Bound Grid.  
Me.GridDataBoundGrid1.ReadOnly = True
```

#### 4.1.4.8.2 Setting ReadOnly Cell by Cell

To set the **ReadOnly** behavior on a cell-by-cell basis, you must use the cell's **GridStyleInfo** object which, has a **ReadOnly** property.

[C#]

```
// Changes cannot be made to the cell (1,1).  
this.gridControl1[1,1].ReadOnly = true;
```

[VB.NET]

```
' Changes cannot be made to the cell (1,1).  
Me.GridControl1(1,1).ReadOnly = True
```

#### 4.1.4.8.3 Making a Change to a ReadOnly Cell

If you set the **Read-Only** behavior, the user will be able to type in the cell and he will also not be able to change the cell's value programmatically. So, to make changes to a **Read-Only** cell, you must use the **GridControl.IgnoreReadOnly** property which, will allow you to change a **Read-Only** cell.

[C#]

```
this.gridControl1[1,1].ReadOnly = true;  
  
// Cell (1,1) has been set to Read-only.  
// To change its value, you need to use the IgnoreReadOnly property.  
this.gridControl1.IgnoreReadOnly = true;  
  
// Turn off Read-only checking.  
this.gridControl1[1,1].CellValue = 256;
```

```
// Now you can change the cell value.  
// Turn on Read-only checking.  
this.gridControl1.IgnoreReadOnly = false;
```

**[VB .NET]**

```
Me.GridControl1(1, 1).ReadOnly = True  
  
' Cell (1,1) has been set to Read-only.  
' To change its value, you need to use the IgnoreReadOnly property.  
Me.GridControl1.IgnoreReadOnly = True  
  
' Turn off Read-only checking.  
Me.GridControl1(1, 1).CellValue = 256  
  
' Now you can change the cell value.  
' Turn on Read-only checking.  
this.Me.GridControl1.IgnoreReadOnly = False
```

#### 4.1.4.9 Using Undo/Redo

Essential Grid supports Undo/Redo functionalities similar to the one achieved with MS-Office type application. To handle this functionality, a stack is maintained internally in Essential Grid, to save the changes handled, through which the following tasks can be accomplished by the users directly.

- Allows to control the stack-when to save/unsave the changes, and when to rollback changes
- Allows to create new transactions, and control each individual transaction(like cancelling, rollback) without affecting the others

The Undo/Redo architecture is extensible thereby allowing the users to derive the base class, and add some more requirements for the Essential Grid.

## See Also

### 4.1.4.9.1 The Basics

Essential Grid has a **GridCommandStack** class that implements support for the Undo/Redo commands in a grid. Depending upon the grid settings, as a user makes changes to the grid these changes will be tracked in stack structures which, will be found in the GridCommandStack class. This class has methods that will allow you to undo the last action, redo the last undone action and batch transactions so that a series of actions can be undone or redone in a single step.

The **CommandStack** property of the **GridControl** class will return a reference to the GridCommandStack object that is associated with a grid. It is through this property that you can access the Undo/Redo support in an Essential Grid. For example, you can use the enabled property of the CommandStack to control whether or not the grid is supporting an Undo/Redo at any given moment. Here are the code samples that show you some CommandStack properties.

#### [C#]

```
// Turn off the Undo buffer.  
this.gridControl1.CommandStack.Enabled = false;  
  
// Turn on the Undo buffer.  
this.gridControl1.CommandStack.Enabled = true;  
  
// Clear the Undo buffer.  
this.gridControl1.CommandStack.UndoStack.Clear();  
  
// Clear the Redo buffer.  
this.gridControl1.CommandStack.RedoStack.Clear();  
  
// Clear both the Undo and Redo buffers.  
this.gridControl1.CommandStack.Clear();
```

#### [VB .NET]

```
' Turn off the Undo buffer.  
Me.gridControl1.CommandStack.Enabled = False  
  
' Turn on the Undo buffer.  
Me.gridControl1.CommandStack.Enabled = True  
  
' Clear the Undo buffer.  
Me.gridControl1.CommandStack.UndoStack.Clear()
```

```
' Clear the Redo buffer.  
Me.gridControl1.CommandStack.RedoStack.Clear()  
  
' Clear both the Undo and Redo buffers.  
Me.gridControl1.CommandStack.Clear()
```

#### 4.1.4.9.2 Transactions

A transaction is a series of steps that should be treated as a single action in the Undo / Redo architecture. For example, you may have a record-oriented grid where you may want to group any changes in the current row as one transaction. This way when the user wants to undo the last change, all the changes in the row are undone. It is possible to group a series of actions into a single Undo / Redo step through the use of these three **GridCommandStack** methods: **BeginTrans**, **CommitTrans** and **RollBack**.

A call to **BeginTrans** will mark the start of a series of actions that are to be treated as a single Undo / Redo step. Once **BeginTrans** has begun, all the changes are marked as being a member of a single transaction until either **CommitTrans** or **RollBack** is called. **CommitTrans** signals a successful end to the transaction. A call to **RollBack** will cause all the changes in the current transaction to be undone and will end the transaction processing. A **RollBack** call will return the grid in the same state that it was in, immediately prior to the call to **BeginTrans**.

It is possible to nest transactions. If you are in the middle of a transaction, it is OK to call **BeginTrans** again. But, when such nested transactions are undone, they are treated as part of a single parent transaction.

#### 4.1.4.9.3 Derived Commands

The Undo / Redo architecture of the Essential Grid is complete as shipped with the product. If, for some reason, you need to handle special grid requirements that cannot be performed with the standard implementation, the Undo / Redo architecture is extensible. To extend it, you need to derive custom command classes from either the abstract class **SyncfusionCommand** or the abstract class **GridModelCommand**. In your derived class, you will need to add whatever members you need in order to track enough state information that will allow you to Undo / Redo the action that is being done. Then you have to implement an execute method and other abstract members of the base class. If you do a search in the Essential Grid source code for **GridModelCommand**, you will see many examples of the derived command classes.

Once you have your derived SyncfusionCommand class, whenever the action is taken, you will have to create a proper instance of your derived SyncfusionCommand class and add it to the **GridControl.CommandStack.UndoStack**. Thus when Essential Grid needs to undo this action, your command will be popped from the **UndoStack**, and its execute method will be called indicating that this action needs to be undone (Also, at this point Essential Grid will add this same instance to the RedoStack so that the action can later be redone if necessary).

#### 4.1.4.10 Serialization

The following serialization techniques are discussed in this section:

##### 4.1.4.10.1 Word Converter

Export to Word is one of the most common functionalities that are required in the .NET world. The Essential Grid control has in-built support for Word Export. Users can download the data from the Grid Control into a Word document for offline verification and/or computation. This can be achieved by making use of the **GridWordConverter** class. This section will walk you through the conversion of the contents of the grid to a word file as well as discuss the various converter options.

GridWordConverter class derives from GridWordConverterBase. It contains a number of methods that helps in exporting different components of the grid.

#### Properties

Here is a list of the properties offered by GridWordConverter. By setting these properties, you could be able to choose the elements you need to export.

Property	Description
ShowHeader	Specifies if header should be displayed.
ShowFooter	Indicates if footer should be displayed.

#### Method

GridWordConverter control provides a method called GridToWord. This is the method that does the conversion of grid contents to a word file. It accepts two parameters: grid to be converted and filename of the destination word document.

#### Syntax

**[C#]**

```
GridWordConverter converter = new GridWordConverter();
converter.GridToWord("Grid.doc", this.gridControl1);
```

**[VB .NET]**

```
Dim converter As GridWordConverter = New GridWordConverter()
converter.GridToWord("Grid.doc", Me.gridControl1)
```

## Events

DrawHeader and DrawFooter are the events offered by the GridWordConverter that aids in adding as well as customizing the header and footer in the destination word document.

## Sample Output

Below images depicts the conversion of grid content to a word file.

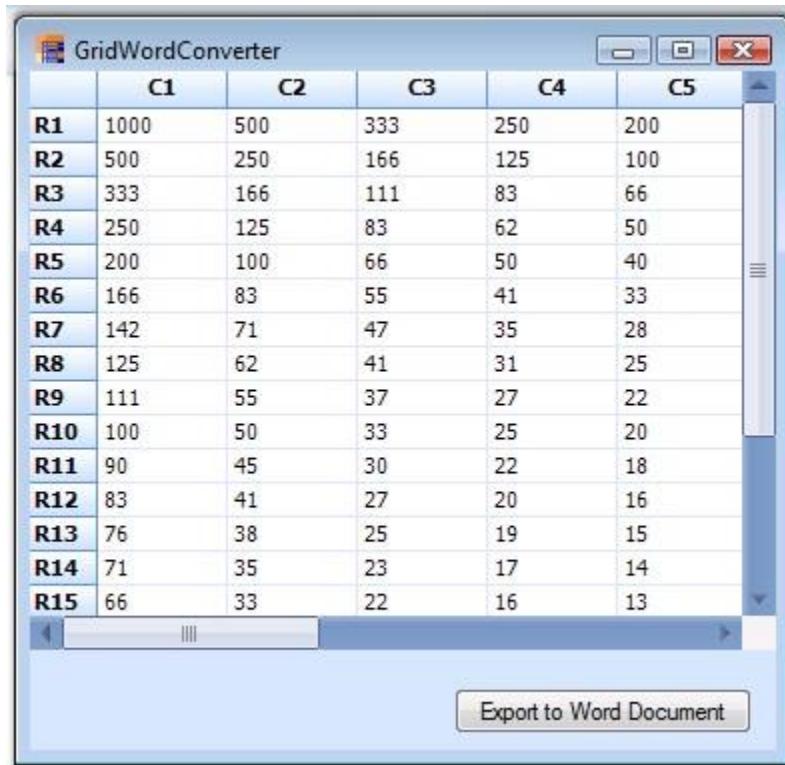
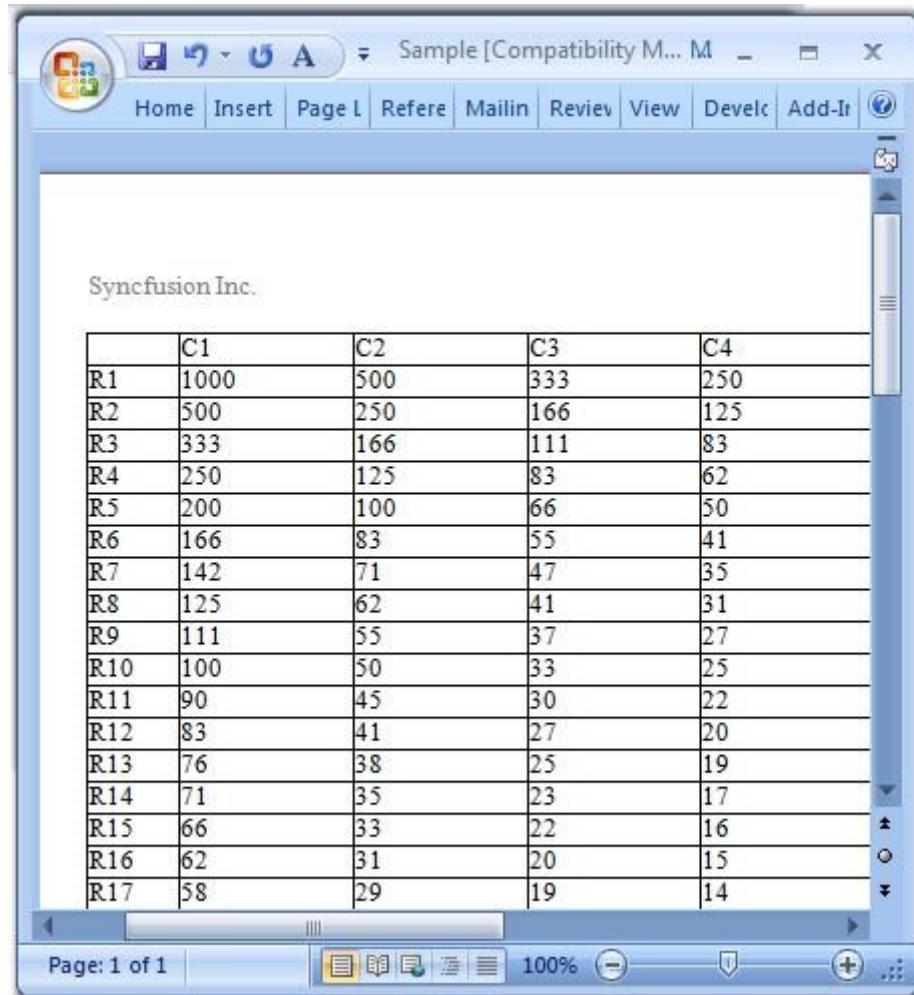


Figure 145: Grid to be Exported



*Figure 146: Grid Exported to a Word File*

A sample demonstrating this feature is available under the following sample installation path.

**<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Windows\Samples\2.0\Export\Word Converter Demo**

#### 4.1.4.10.2 Excel Export

Exporting data to an Excel spreadsheet is one of the most commonly preferred features in the .NET world. The Essential Grid control has in-built support for an Excel spreadsheet export. The class **GridExcelConverterControl** provides support for exporting data from a Grid control or Grid Data Bound Grid control to an Excel spreadsheet for verification and/or computation. This class automatically copies a grid's styles and formats to an Excel spreadsheet. The **GridExcelConverterControl** class is derived from the **GridExcelConverterBase** class. The XlsIO libraries are used to support the conversion of the grid contents to Excel.

For the control to function, the following dll files should be added along with the default dll files in the reference folder:

- Syncfusion.GridConverter.Base
- Syncfusion.XlsIO.Base

The **GridToExcel** method should be used to export the grid to an excel sheet. Following code example illustrates how to convert the content in Grid control to an Excel spreadsheet.

#### 1. Using C#

[C#]

```
Syncfusion.GridExcelConverter.GridExcelConverterControl gecc = new  
Syncfusion.GridExcelConverter.GridExcelConverterControl();  
gecc.GridToExcel(this.gridControl1.Model, @"C:\MyGC.xls");
```

#### 2. Using VB.NET

[VB .NET]

```
Dim gecc As Syncfusion.GridExcelConverter.GridExcelConverterControl =  
New Syncfusion.GridExcelConverter.GridExcelConverterControl()  
gecc.GridToExcel(Me.gridControl1.Model, "C:\MyGC.xls")
```

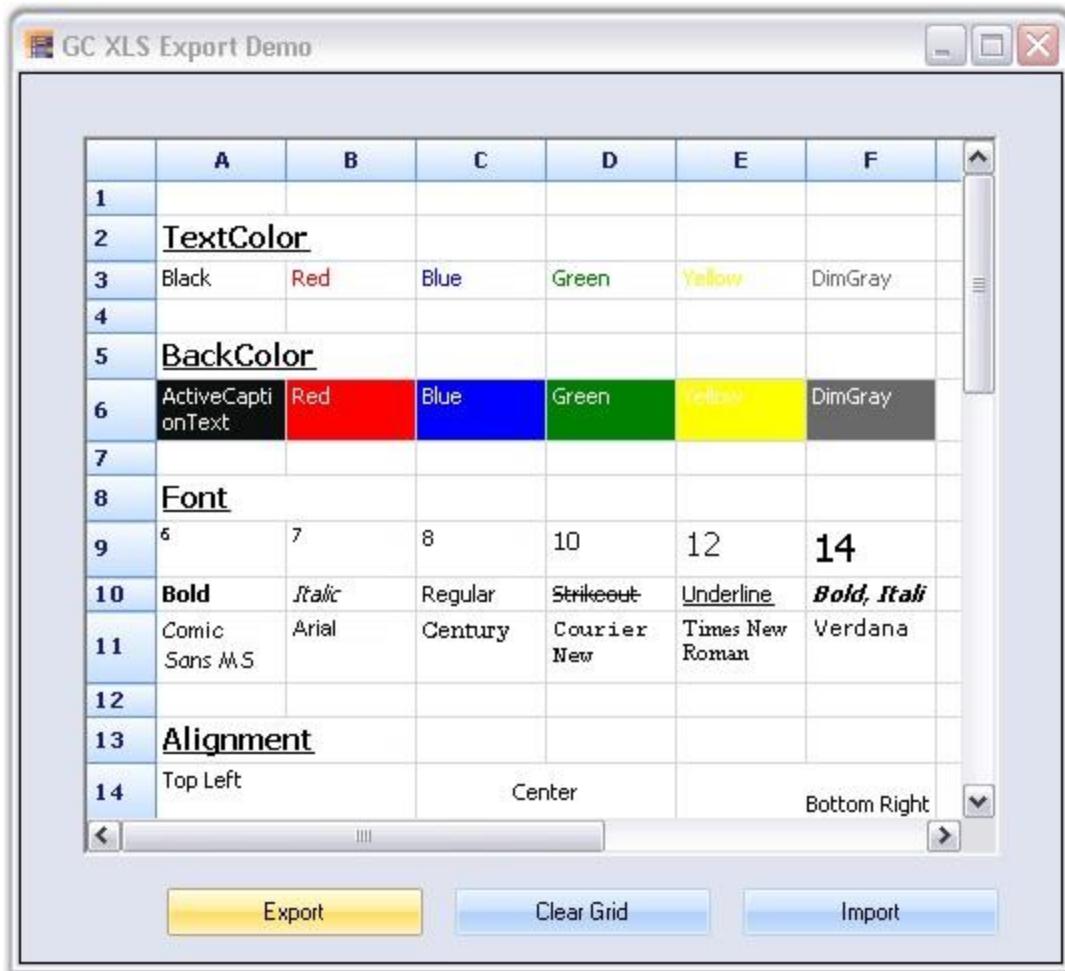


Figure 147: Grid to be Exported

Following code example illustrates how to convert the content in Grid Data Bound Grid control to an Excel spreadsheet.

### 1. Using C#

[C#]

```
Syncfusion.GridExcelConverter.GridExcelConverterControl gecc = new
Syncfusion.GridExcelConverter.GridExcelConverterControl();
gecc.GridToExcel(this.gridDataBoundGrid1.Model, @"C:\MyGC.xls");
```

### 2. Using VB.NET

[VB .NET]

```
Dim gecc As Syncfusion.GridExcelConverter.GridExcelConverterControl =  
New Syncfusion.GridExcelConverter.GridExcelConverterControl()  
gecc.GridToExcel(Me.gridDataBoundGrid1.Model, "C:\MyGC.xls")
```

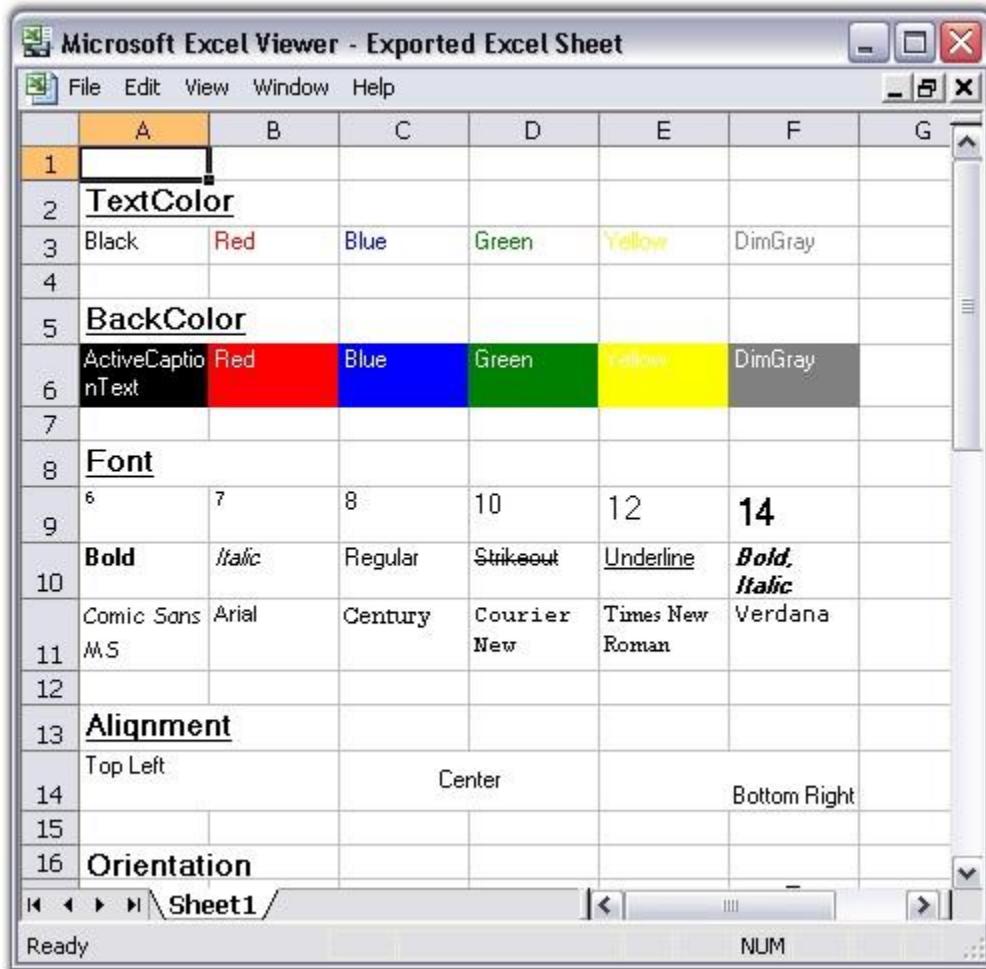


Figure 148: Exported Excel Sheet

A sample demonstrating this feature is available under the following sample installation path.

**<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Windows\Samples\2.0\Export\GC XLS Export Demo**

#### 4.1.4.10.3 PDF Export

Our Essential Grid control supports conversion of grid content to a PDF file. Data in Grid control can be converted to a PDF document for offline verification and/or computation. This can be achieved by making use of the **GridPDFConverter** class. The PDF libraries are used to support the conversion of grid content to a PDF page.

For making the control functional, the following dll files should be added along with the default dll files in the reference folder:

- Syncfusion.Pdf.Base
- Syncfusion.GridHelperClasses.Windows.

The **ExportToPdf** method is used to export the grid content to a PDF file. Following code example illustrates how to convert the content in grid to PDF.

#### 1. Using C#

**[C#]**

```
GridPDFConverter pdfConvertor = new GridPDFConverter();
pdfConvertor.ExportToPdf("Sample1.pdf", this.gridControl1);
```

#### 2. Using VB.NET

**[VB .NET]**

```
Dim pdfConvertor As GridPDFConverter = New GridPDFConverter()
pdfConvertor.ExportToPdf("Sample1.pdf", Me.gridControl1)
```

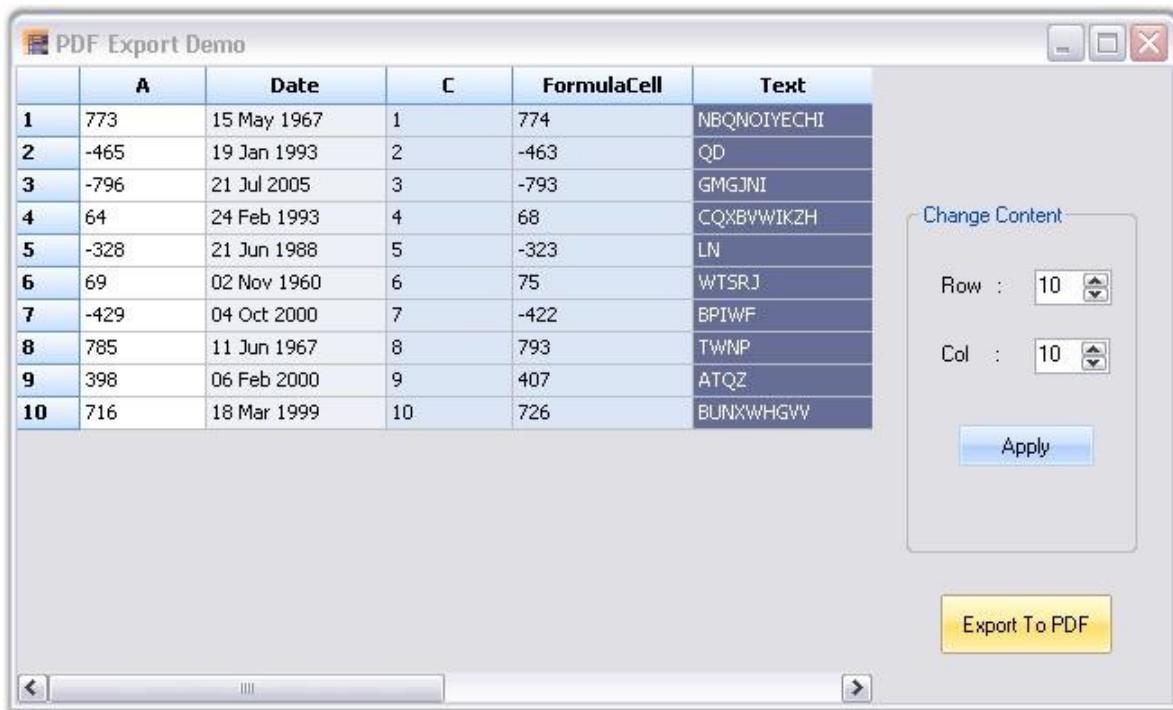


Figure 149: Grid to be Exported

A sample demonstrating this feature is available under the following sample installation path.

<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Windows\Samples\2.0\Export\PDF Converter Demo

#### Support to Access or Modify Document Attributes of Exported PDF

This feature allows you to access and modify PDF document attributes while exporting, or after exporting, a grid to PDF. When you want to check the page count of the exporting document, you can use this feature.

#### Events

Table 6: Export Event table

Event	Description	Arguments	Type	Reference links

Exporting	This will be triggered before exporting grid to PDF.	(object sender, EventArgs e)	event	N/A
Exported	This will be triggered after exporting grid to PDF.	(object sender, EventArgs e)	event	N/A

**Sample Link**

A demo of this feature is available in the following location:

**L..\\AppData\\Local\\Syncfusion\\EssentialStudio\\{Version}\\Windows\\Grid.Windows\\Samples\\2.0\\Export\\PDF Export Demo**

**Hooking the events in an application**

You can hook the events using the *ExportToPdf()* method of *PDFconverter*. The following code illustrates this:

[C#]

```
GridPDFConverter pdfConvertor = new GridPDFConverter();
pdfConvertor.Exporting += new
GridPDFConverter.PDFExportingEventHandler(pdfConvertor_Exporting);
pdfConvertor.Exported += new
GridPDFConverter.PDFExportedEventHandler(pdfConvertor_Exported);
```

[VB]

```
Dim pdfConvertor As GridPDFConverter = New GridPDFConverter()
AddHandler pdfConvertor.Exporting, AddressOf pdfConvertor_Exporting
AddHandler pdfConvertor.Exported, AddressOf pdfConvertor_Exported
```

#### 4.1.4.10.4 SOAP, Binary and XML Serialization

Essential Grid control has support for serialization and de-serialization of the grid's schema information.

#### 4.1.4.10.5 Export as Image Support

Essential Grid provides image export support for the Grid, GridDataBoundGrid, and GridGrouping controls in the Excel converter. With this support, users can enable or disable image export from a Grid control to Excel. This is a Boolean property and its default value is true. This property will affect the grid-to-Excel converter when the **ExportStyle** property is true.

#### Use Case Scenarios

This feature allows you to control the image export in the grid-to-Excel converter. This property will affect the grid when ExportStyle is true.

#### Properties

Property	Description	Data Type
ExportImage	Used to enable or disable the image export in Syncfusion Windows Forms grid.	Boolean

#### Syntax

##### [C#]

```
ExcelExport.ExportStyle = true;
// ExportImage works only when ExportStyle is true.
ExcelExport.ExportImage = false;
```

##### [VB .NET]

```
ExcelExport.ExportStyle = True
' ExportImage works only when ExportStyle is true.
ExcelExport.ExportImage = False
```



**Note:** *Serialization is the process of saving the state of an object as a stream of bytes. The reverse of this process is called deserialization.*

Grid control supports three different types of serialization techniques namely:

- **SOAP**-Helps convert the grid schema information to SOAP format.
  - **Binary**-Helps convert the grid schema information to binary format.
  - **XML**-Helps convert the grid schema information to XML format.
- **SOAP**
    - Following code example illustrates serialization of grid schema information by using SOAP technique.
      1. Using C#

[C#]

```
this.gridControl1.Model.SaveSoap(dlg.FileName);
```

2. Using VB.NET

[VB .NET]

```
Me.gridControl1.Model.SaveSoap(dlg.FileName)
```

- Following code example illustrates deserialization of grid schema information by using SOAP technique.

1. Using C#

[C#]

```
this.gridControl1.Model = GridModel.LoadSoap(dlg.FileName);  
this.gridControl1.Refresh();
```

2. Using VB.NET

[VB .NET]

```
Me.gridControl1.Model = GridModel.LoadSoap(dlg.FileName)  
Me.gridControl1.Refresh()
```

- **Binary**

- Following code example illustrates serialization of grid schema information by using Binary technique.

#### 1. Using C#

[C#]

```
this.gridControl1.Model.SaveBinary(dlg.FileName);
```

#### 2. Using VB.NET

[VB .NET]

```
Me.gridControl1.Model.SaveBinary(dlg.FileName);
```

- Following code example illustrates deserialization of grid schema information by using Binary technique.

#### 1. Using C#

[C#]

```
this.gridControl1.Model = GridModel.LoadBinary(dlg.FileName);
this.gridControl1.Refresh();
```

#### 2. Using VB.NET

[VB .NET]

```
Me.gridControl1.Model = GridModel.LoadBinary(dlg.FileName)
Me.gridControl1.Refresh()
```

### • XML

- Following code example illustrates serialization of grid schema information by using XML technique.

#### 1. Using C#

[C#]

```
Stream s = null;
s = File.Create(dlg.FileName);
XmlWriter xw = new XmlTextWriter(s, System.Text.Encoding.Default);
XmlSerializer xs = new
System.Xml.Serialization.XmlSerializer(this.gridControl1.Model.Data.GetType());
xs.Serialize(xw, this.gridControl1.Model.Data);
s.Close();
```

## 2. Using VB.NET

### [VB.NET]

```
Dim s As Stream = Nothing
s = File.Create(dlg.FileName)
Dim xw As XmlWriter = New XmlTextWriter(s,
System.Text.Encoding.Default)
Dim xs As XmlSerializer = New
System.Xml.Serialization.XmlSerializer(Me.gridControl1.Model.Data.GetType())
xs.Serialize(xw, Me.gridControl1.Model.Data)
s.Close()
```

- o Following code example illustrates deserialization of grid schema information by using XML technique.

## 1. Using C#

### [C#]

```
Stream s = null;
s = File.OpenRead(dlg.FileName);
XmlReader xw = new XmlTextReader(s);
XmlSerializer xs = new
System.Xml.Serialization.XmlSerializer(this.gridControl1.Model.Data.GetType());
s.Close();
this.gridControl1.Model.Data = (GridData)xs.Deserialize(xw);
this.gridControl1.Refresh();
```

## 2. Using VB.NET

[VB.NET]

```
Dim s As Stream = Nothing
s = File.OpenRead(dlg.FileName)
Dim xw As XmlReader = New XmlTextReader(s)
Dim xs As XmlSerializer = New
System.Xml.Serialization.XmlSerializer(Me.gridControl1.Model.Data.GetType())
s.Close()
Me.gridControl1.Model.Data = CType(xs.Deserialize(xw), GridData)
Me.gridControl1.Refresh()
```



*Figure 150: Serialization Illustrated*

A sample demonstrating this feature is available under the following sample installation path.

**<Install Location>\Syncfusion\EssentialStudio\Version  
Number\Windows\Grid.Windows\Samples\2.0\Serialization\Serialize Grid Control Demo**

#### 4.1.4.11 Virtual Grids

Essential Grid supports complete separation between the **datasource** and the grid. In a virtual grid, no cell data is stored in the **GridStyleInfo** objects or any other internal grid storage. All information is provided on demand through handled events. For example, whenever Essential Grid needs a row count for a grid, it fires a **QueryRowCount** event. In your handler, you must provide the row count from your datasource. Virtual grids can display large amounts of data extremely fast. There is no need to perform the time-consuming task of populating the grid.

To implement a Read-only virtual grid, you'll need to handle three events. To remove the Read-only limitation, you will have to handle a fourth event. In addition to these four events, there are other events that you may want to handle depending upon the behavior you are trying to implement. We will first discuss the required events and then discuss the optional events that you can handle to affect virtual grid behavior. You can also work through the virtual grid tutorial to see an implementation of a simple virtual grid.

The events are discussed under the below sections:

#### **4.1.4.11.1 Required Events**

These are the three events that you should handle in order to implement a virtual grid. They provide the basic information about the number of rows, columns and the values for your data.

##### *4.1.4.11.1.1 QueryRowCount Event*

This event is used to return the row count on demand. Here is a sample handler.

**[C#]**

```
private void GridQueryRowCount(object sender, GridRowColCountEventArgs e)
{
    // Determine number of rows.
    e.Count = this.numArrayRows;
    e.Handled = true;
}
```

**[VB .NET]**

```
Private Sub GridQueryRowCount(ByVal sender As Object, ByVal e As
GridRowColCountEventArgs)
```

```
' Determine number of rows.  
e.Count = Me.numArrayRows  
e.Handled = True  
End Sub
```

#### 4.1.4.11.1.2 QueryColCount Event

The **QueryColCount** event is used to return the column count on demand. Note that when you handle the event by assigning e.Count, you are setting the e.Handled property to true.

##### [C#]

```
private void GridQueryColCount(object sender, GridRowColEventArgs e)  
{  
    // Determine the number of columns.  
    e.Count = this.numArrayCols;  
    e.Handled = true;  
}
```

##### [VB .NET]

```
Private Sub GridQueryColCount(ByVal sender As Object, ByVal e As  
GridRowColEventArgs)  
  
    ' Determine the number of columns.  
    e.Count = Me.numArrayCols  
    e.Handled = True  
End Sub
```

#### 4.1.4.11.1.3 QueryCellInfo Event

**QueryCellInfo** is the workhorse event. It is used to provide the **GridStyleInfo** object for a given cell. In your handler for this event, you would normally set the **CellValue** for the GridStyleInfo object passed in with the event arguments. But, you can also set other members of this GridStyleInfo object. For example, you can set the BackColor to change the cell background. All of this is done on a demand basis. The **BackColor** value is not stored in any grid storage. There is another event, **PrepareViewStyleInfo** that you can handle to make a transient adjustment to a style just before it is displayed. This event is discussed in more detail later in this section.

The **GridQueryCellInfoEventArgs** members, e.ColumnIndex and e.RowIndex, specify the column and row of the requested style. The e.Style member holds the **GridStyleInfo** object whose value this event should set provided it is a cell that you want to populate. It is possible that e.ColumnIndex and / or e.RowIndex may have the value of -1. A -1 indicating that a **rowstyle** or **columnstyle** is being requested. So, e.ColumnIndex = -1 and e.RowIndex = 4 indicates that the rowstyle for row 4 is being requested (**GridControl.RowStyle[4]**). Similarly, a positive column value with the row value = -1 would be a request for that particular columnstyle. If both values are -1, then the **TableStyle** property is being requested.

One last comment before we look at the code. Header rows and columns in an Essential Grid are treated the same as other rows and columns with respect to **QueryCellInfo**. If you have a single header row, then anytime e.ColumnIndex is 0, a row header is being requested. Similarly, if you have a single column header row, e.RowIndex = 0 is a request for the column header.

#### [C#]

```
private void GridQueryCellInfo(object sender,
GridQueryCellInfoEventArgs e)
{
    if(e.ColumnIndex > 0 && e.RowIndex > 0)
    {
        // Using indexers, pass value to a cell from a given data
        source.
        e.Style.CellValue = this.intArray[e.RowIndex - 1, e.ColumnIndex -
1];
        e.Handled = true;
    }
}
```

#### [VB .NET]

```
Private Sub GridQueryCellInfo(ByVal sender As Object, ByVal e As
GridQueryCellInfoEventArgs)
    If ((e.ColumnIndex > 0) AndAlso (e.RowIndex > 0)) Then

        ' Using indexers, pass value to a cell from a given data source.
        e.Style.CellValue = Me.intArray(e.RowIndex - 1, e.ColumnIndex - 1)
        e.Handled = True
    End If
End Sub
```

#### 4.1.4.11.2 Optional Events

Optional events can be used to extend the functionality of the basic Read-only virtual grid that you get by handling these three required events. One event lets you save information back into your external **datasource** while other events let you dynamically specify row heights and column widths. You can also dynamically provide covered cell ranges.

##### 4.1.4.11.2.1 SaveCellInfo Event

This event is used to store data back into your data source when it has been changed by the user. Here is a sample handler.

###### [C#]

```
void GridSaveCellInfo(object sender, GridSaveCellInfoEventArgs e)
{
    if( e.ColIndex > 0 && e.RowIndex > 0)
    {
        // Store data back to the data source from the grid cell.
        this._extData[e.RowIndex - 1, e.ColIndex - 1] =
int.Parse(e.Style.CellValue.ToString());
        e.Handled = true;
    }
}
```

###### [VB .NET]

```
Private Sub GridSaveCellInfo(ByVal sender As Object, ByVal e As
GridSaveCellInfoEventArgs)
    If ((e.ColIndex > 0) AndAlso (e.RowIndex > 0)) Then

        ' Store data back to the data source from the grid cell.
        Me._extData((e.RowIndex - 1), (e.ColIndex - 1)) =
System.Int32.Parse(e.Style.CellValue.ToString())
        e.Handled = True
    End If
End Sub
```

##### 4.1.4.11.2.2 QueryRowHeight

This event is used to return row heights that are in demand.

**[C#]**

```
void GridQueryRowHeight(object sender, GridRowColSizeEventArgs e)
{
    if( e.Index % 2 == 0)
    {
        // Determine Row Height.
        e.Size = 20;
        e.Handled = true;
    }
}
```

**[VB.NET]**

```
Private Sub GridQueryRowHeight(ByVal sender As Object, ByVal e As
GridRowColSizeEventArgs)
    If ((e.Index Mod 2) = 0) Then

        ' Determine Row Height.
        e.Size = 20
        e.Handled = True
    End If
End Sub
```

#### 4.1.4.11.2.3 QueryColWidth

This event is used to return column widths that are in demand. Here is a sample handler.

**[C#]**

```
void GridQueryColWidth(object sender, GridRowColSizeEventArgs e)
{
    if( e.Index % 3 == 0)
    {
        // Assign Column Width.
        e.Size = 40;
        e.Handled = true;
    }
}
```

**[VB.NET]**

```
Private Sub GridQueryColWidth(ByVal sender As Object, ByVal e As GridRowColSizeEventArgs)
    If ((e.Index Mod 3) = 0) Then

        ' Assign Column Width.
        e.Size = 40
        e.Handled = True
    End If
End Sub
```

#### 4.1.4.11.2.4 QueryCoveredRange

This event is used to provide covered ranges on demand. If you have a pattern of cells covered, then you can use this event to provide the ranges.

**[C#]**

```
void GridQueryCoveredRange(object sender,
GridQueryCoveredRangeEventArgs e)
{
    // Cover odd rows, columns 1 through 3.
    if (e.RowIndex % 2 == 1 && e.ColIndex >= 1 && e.ColIndex <= 3)
    {
        e.Range = GridRangeInfo.Cells(e.RowIndex, 1, e.RowIndex, 3);
        e.Handled = true;
    }
    // Cover column 6 with odd-even row pairs.
    if (e.RowIndex > 0 && e.ColIndex == 6)
    {
        int row = (e.RowIndex-1)/2 * 2 + 1;
        int col = e.ColIndex;
        e.Range = GridRangeInfo.Cells(row, col, row+1, col);
        e.Handled = true;
    }
}
```

**[VB.NET]**

```
Private Sub GridQueryCoveredRange(ByVal sender As Object, ByVal e As GridQueryCoveredRangeEventArgs)
```

```

' Cover odd rows, columns 1 through 3.
If (((e.RowIndex Mod 2) = 1) AndAlso (e.ColumnIndex >= 1)) _
    AndAlso (e.ColumnIndex <= 3)) Then
    e.Range = GridRangeInfo.Cells(e.RowIndex, 1, e.RowIndex, 3)
    e.Handled = True
End If

' Cover column 6 with odd-even row pairs.
If ((e.RowIndex > 0) AndAlso (e.ColumnIndex = 6)) Then
    Dim row As Integer
    row = (((e.RowIndex - 1) / 2) * 2) + 1
    Dim col As Integer
    col = e.ColumnIndex
    e.Range = GridRangeInfo.Cells(row, col, (row + 1), col)
    e.Handled = True
End If
End Sub

```

#### 4.1.4.12 Pivot Grid

Essential Pivot Grid simulates the Pivot Table feature of MS Excel. The Pivot Grid pivots the data via drag-and-drop to organize the data in a cross-tabulated form. The major advantage of pivot grid is that you can extract any desired information within a limited span of time. Apart from being able to present the data in a proper manner, you can also summarize and group the data. Pivot Grid has its main application in the financial domain. It is used to organize and analyze the business data.

The Pivot Grid control is built on the foundation of the Grid control. It comprises of the following components.

- **Display Grid** - Displays the data extracted from the underlying database.
- **Pivot Table Field List** - Lists the available fields from the database. Provides a way to add / remove the fields from the grid.
- **Drag-Drop Panel** - Serves as a view state of the pivot grid, where you can rearrange the fields by performing a drag-and-drop operation between the row and column label areas.
- **Filter Area** - Lets you filter the results according to certain criteria in a desired manner.

Pivot Grid provides a UI that allows you to specify the rows and columns in the pivot table through drag-and-drop operations. The visual aspects of the control are saved in an Appearance object. The control supports Office 2003 and Office 2007 styles.

The calculations are done through the Grouping Engine, which is a part of Essential Grouping. The default calculation is *Summation*, but there exists an option to change the calculation type to *Average*, *Median*, *Percentiles*, *Variances*, *Standard Deviations*, and so on. You can also provide "custom" calculations through the grouping engine.

## Features

- Data-binding support
- Auto-calculation of Total Summary
- Filters
- Grouping support
- Customizable Appearance
- Support for XML and Binary Serialization

## APIs

Here is a brief explanation on some important methods implemented in the Pivot Grid.

- **CollapseAll()** - This will collapse all the expanded tables in the Pivot Grid.

```
[C#]
```

```
pivotGridControl1.CollapseAll();
```

- **ExpandAll()** - Expands all the collapsed nodes in the Pivot Grid.

```
[C#]
```

```
pivotGridControl1.ExpandAll();
```

- **InitSchema()** - A new Pivot schema will be created, and it will be associated with the Pivot Grid.

```
[C#]
```

```
pivotGridControl1.InitSchema();
```

- **ResetSchema()** - Resets the Pivot Grid control into an initial schema which will be empty.

```
[C#]
```

```
pivotGridControl1.ResetSchema();
```

- **SetAppearance()** - This method sets the appearance of the Pivot Grid.

[C#]

```
pivotGridControl1.SetAppearance(new  
PivotGridLibrary.PivotAppearance(pivotGridControl2));
```

Here is a brief explanation on some important properties implemented in the Pivot Grid.

- **AllString** - This will get the string values that appear in the dropdown filter, when all the filter values get selected.

[C#]

```
pivotGridControl1.AllString = "All";
```

- **AutoSizeColumns** - Sizes the column according to the calculated value of the display width.

[C#]

```
pivotGridControl1.AutoSizeColumns = true;
```

- **ColumnCount** - This specifies the number of columns in the main display grid.

[C#]

```
int i = pivotGridControl1.ColumnCount;
```

- **ColumnsCount** - This specifies the number of distinct fields in the pivot grid.

[C#]

```
int i = pivotGridControl1.ColumnsCount;
```

- **DataRowCount** - This specifies the number of rows in the underlying IList datasource.

[C#]

```
int i = pivotGridControl1.DataRowCount;
```

- **DefaultComputationName** - This specifies the name of the default calculation. The default value is *Sum*.

[C#]

```
pivotGridControl1.DefaultComputationName = "Sum";
```

- **DefaultDescriptionFormat** - This specifies the format of the calculated description. By default it will be, {0} of {1}, where {0} is the value of computation name and {1} is the value of the field name.

[C#]

```
pivotGridControl1.DefaultDescriptionFormat = "{0} of {1}";
```

- **FilterCount** - This specifies the number of distinct fields added to the filter.

[C#]

```
int i = pivotGridControl1.FilterCount;
```

- **FreezeHeaders** - This determines whether the row and column headers should be frozen.

[C#]

```
pivotGridControl1.FreezeHeaders = true;
```

- **GrandTotalString** - This provides the text for the summary cells of the Pivot Grid.

[C#]

```
pivotGridControl1.GrandTotalString = "Grand Total";
```

- **LeftPanelWidth** - This specifies the width of the left-most Panel.

[C#]

```
pivotGridControl1.LeftPanelWidth = 20;
```

- **LeftPanelHeight** - This specifies the height of the top-most Panel.

[C#]

```
pivotGridControl1.LeftPanelHeight = 20;
```

- **MainDisplayGrid** - This is a basic grid which stores the pivot results.

- **MultipleString** - This specifies the text that should appear when the Multiple Filter selected in the Filter Combo Box is changed. By default it will be set to **Multiple**.

[C#]

```
pivotGridControl1.MultipleString = "Multiple";
```

- **Print Option**

The print option is extended for the PivotGrid control to allow users to preview the contents before the contents are printed on paper.

This feature is used to print the PivotGrid control in landscape and portrait views. This feature has overridden the GridPrintDocumentAdv from Syncfusion.GridHelperClasses.Windows.

The pivot grid visual style color is automatically applied in the printed document based on the visual styles of the grid.

The print functionality can be invoked using the following code:

[C#]

```
private void button1_Click_1(object sender, EventArgs e)
{
    try
    {
        PivotGridPrintDocumentAdv pd = new
        PivotGridPrintDocumentAdv(this.pivotGridControl1);

        pd.DefaultPageSettings.Margins = new
        System.Drawing.Printing.Margins(25, 25, 25, 25);
        PrintPreviewDialog previewDialog = new PrintPreviewDialog();
        previewDialog.Document = pd;
        previewDialog.Show();
    }

    catch (Exception ex)
    {
        MessageBox.Show("Error while print preview" + ex.ToString());
    }
}
```

[VB]

```
Private Sub button1_Click_1(ByVal sender As Object, ByVal e As EventArgs)
    Try
        Dim pd As New PivotGridPrintDocumentAdv(Me.pivotGridControl1)
        pd.DefaultPageSettings.Margins = New
        System.Drawing.Printing.Margins(25, 25, 25, 25)
        Dim previewDialog As New PrintPreviewDialog()
        previewDialog.Document = pd
        previewDialog.Show()

        Catch ex As Exception
            MessageBox.Show("Error while print preview" & ex.ToString())
        End Try
    End Sub
```

The following screen shots illustrate the print feature of the PivotGrid control:

Headers and footers can also be added by using the **DrawGridPrintHeader** and **DrawGridPrintFooter** events. The following code illustrates how to add the header and footer.

**[C#]**

```
pd.DrawGridPrintHeader+=new
GridPrintDocumentAdv.DrawGridHeaderFooterEventHandler(pd_DrawGridPrintHeader);

pd.DrawGridPrintFooter+=new
GridPrintDocumentAdv.DrawGridHeaderFooterEventHandler(pd_DrawGridPrintFooter);
```

**[VB]**

```
AddHandler pd.DrawGridPrintHeader, AddressOf pd_DrawGridPrintHeader

AddHandler pd.DrawGridPrintFooter, AddressOf pd_DrawGridPrintFooter
```

The following image shows the printed output of the pivot grid:

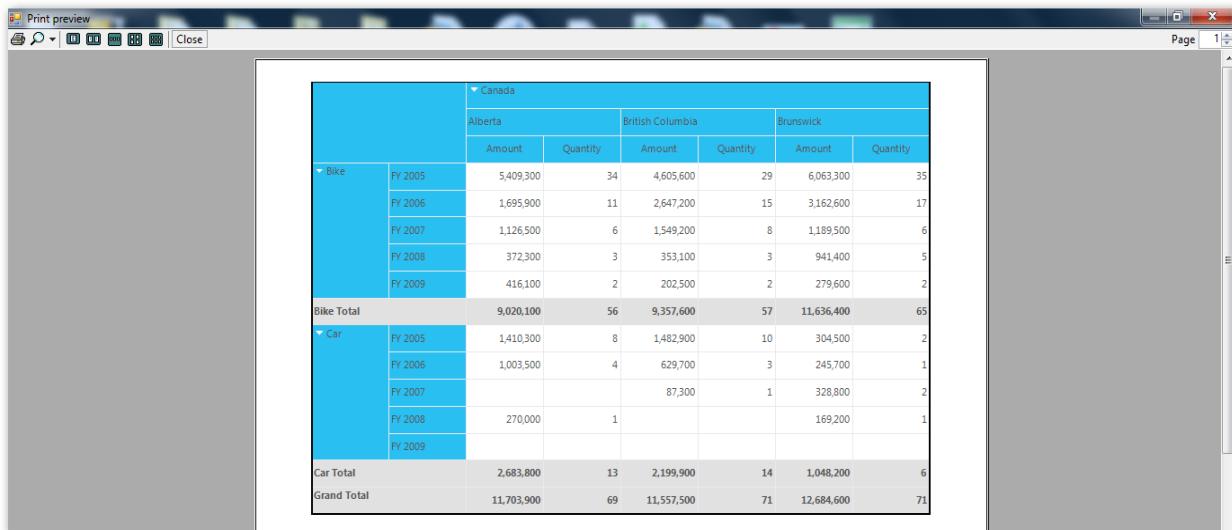


Figure 151: Pivot Grid in Print Preview

### Sample Link

A sample is available in the following location:

**<Installed Location>\Syncfusion\EssentialStudio\[Version Number]\Windows\PivotGrid.Windows\Samples\2.0\Print\Print Grid Demo**

#### 4.1.4.13 Appearance

This section comprises the following:

##### 4.1.4.13.1 GridFormatCellDialog

GridFormatCellDialog simulates the FormatCells dialog feature of MS Excel. It provides numerous formatting options such as Font, Alignment, Background and Number, which aid in formatting the grid cells dynamically. It is now available as an add-on feature for Essential Grid control.

The GridFormatCellDialog class accepts an instance of the Grid control to be formatted, and exposes the above mentioned formatting options to operate on the grid cells that are selected. Below image illustrates such a sample dialog.

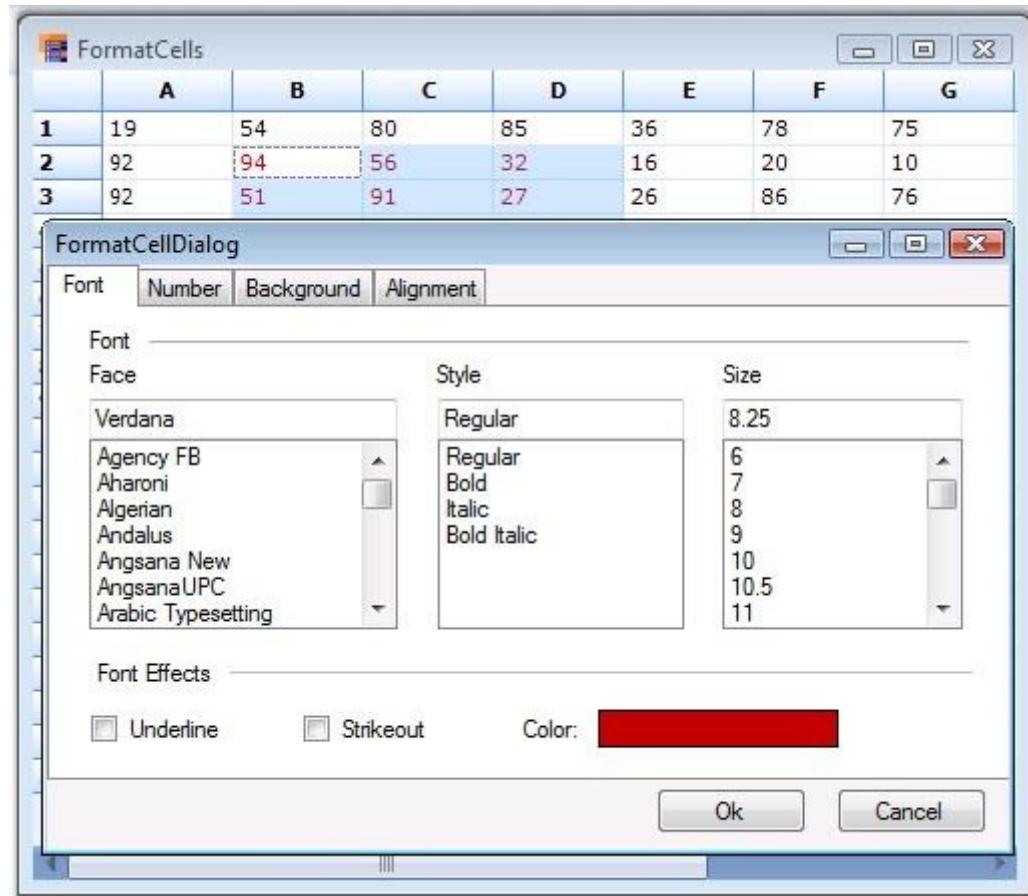


Figure 152: Format Cell Dialog Box

### Setting up GridFormatCellDialog

This GridFormatCellDialog can be enabled by instantiating the GridFormatCellDialog class, and invoking its **ShowDialog** method.



**Note:** You must select the cells to be formatted before activating this dialog.

#### [C#]

```
GridFormatCellDialog formatDialog = new
GridFormatCellDialog(this.gridControl1);
formatDialog.ShowDialog();
```

#### [VB .NET]

```
Dim formatDialog As GridFormatCellDialog = New
GridFormatCellDialog(Me.gridControl1)
```

```
formatDialog.ShowDialog()
```

## Formatting Options

### Font Tab

This provides options to set the font, font style, font size, font effects and font color for the desired grid cells.

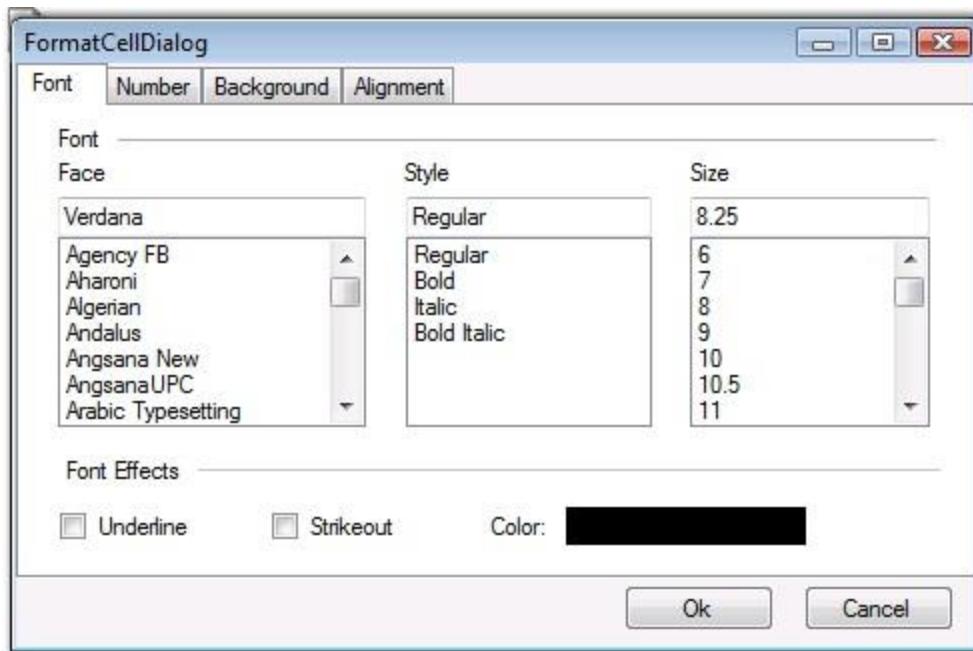


Figure 153: Font customization options in the Format Cell Dialog Box

### Number Tab

This allows you to specify a text format for the grid cells. The possible options are Number, Currency, Percentage, Date, Time, Scientific and Text.

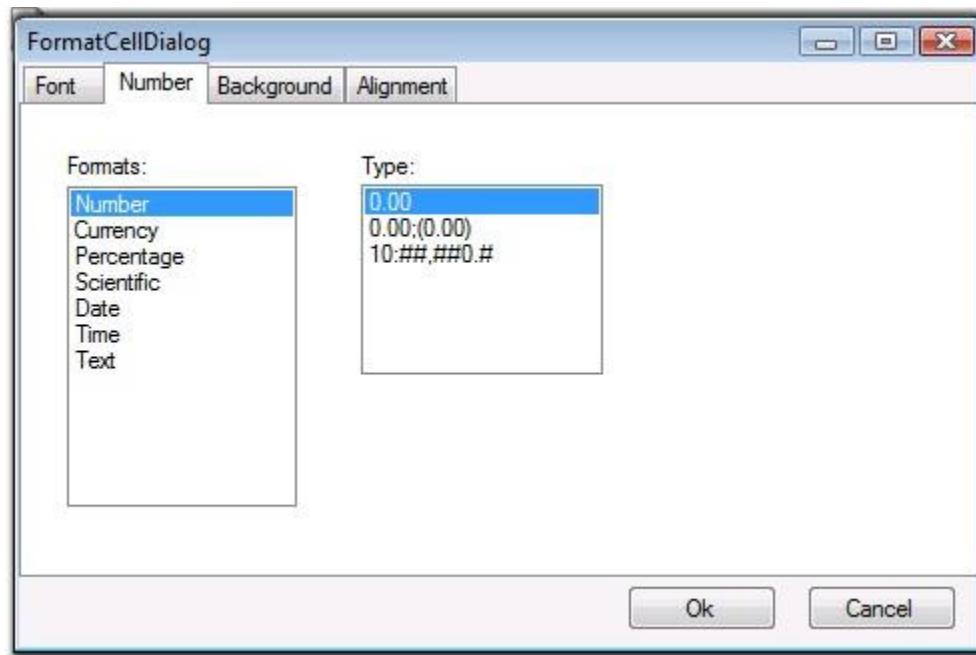


Figure 154: Text Format options in the Format Cell Dialog Box

### Background Tab

This allows you to set the background color for the grid cells. You can set gradient shades and pattern styles as well.

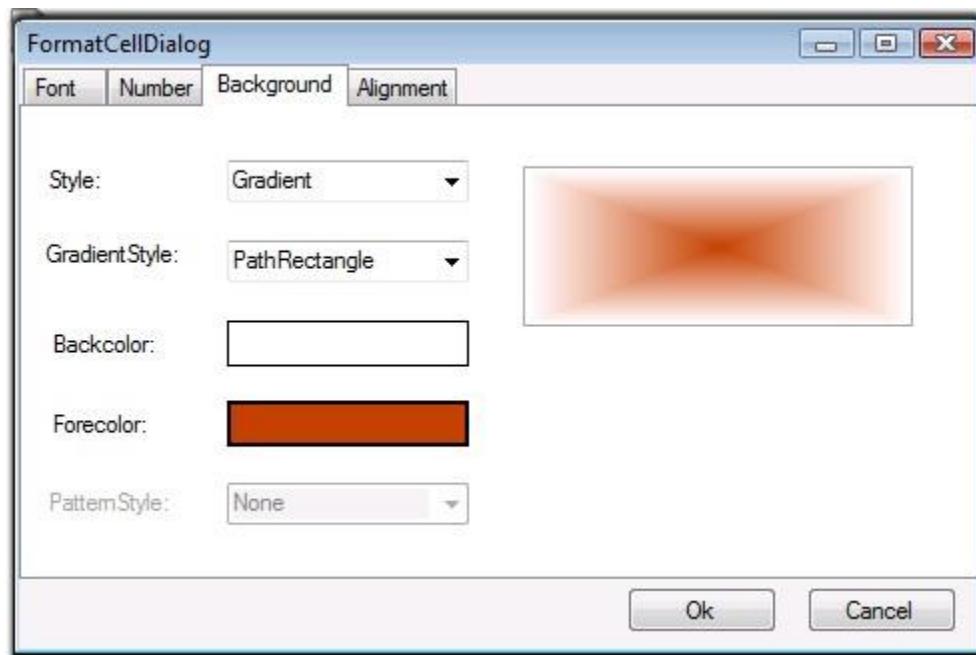


Figure 155: Background customization options in the Format Cell Dialog Box

### Alignment Tab

This provides various cell alignment options such as Horizontal Alignment, Vertical Alignment, Merge Cells, Wrap Text, and so on.

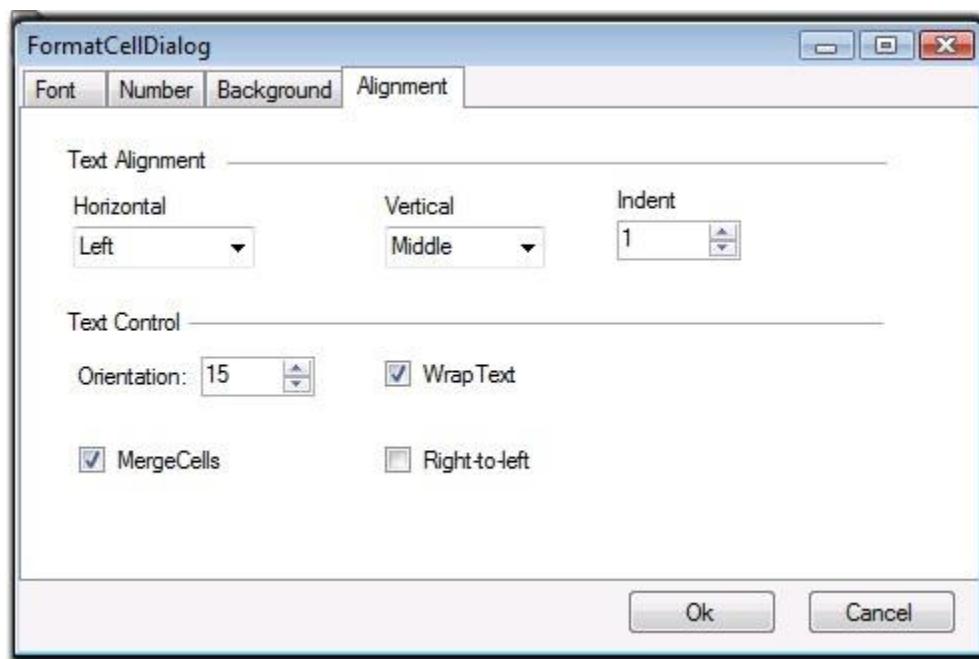


Figure 156: Alignment options in the Format Cell Dialog Box

#### 4.1.4.13.2 Grid Visual Styles

This section provides information on the **VisualStyles** and **ThemesEnabled** properties (XP themes) of the Essential Grid.

Essential Grid supports a range of appearances for grid cells. Styles can be set to the grid control by assigning **Syncfusion.Windows.Forms.GridVisualStyles** enumeration value to the **GridVisualStyles** property.

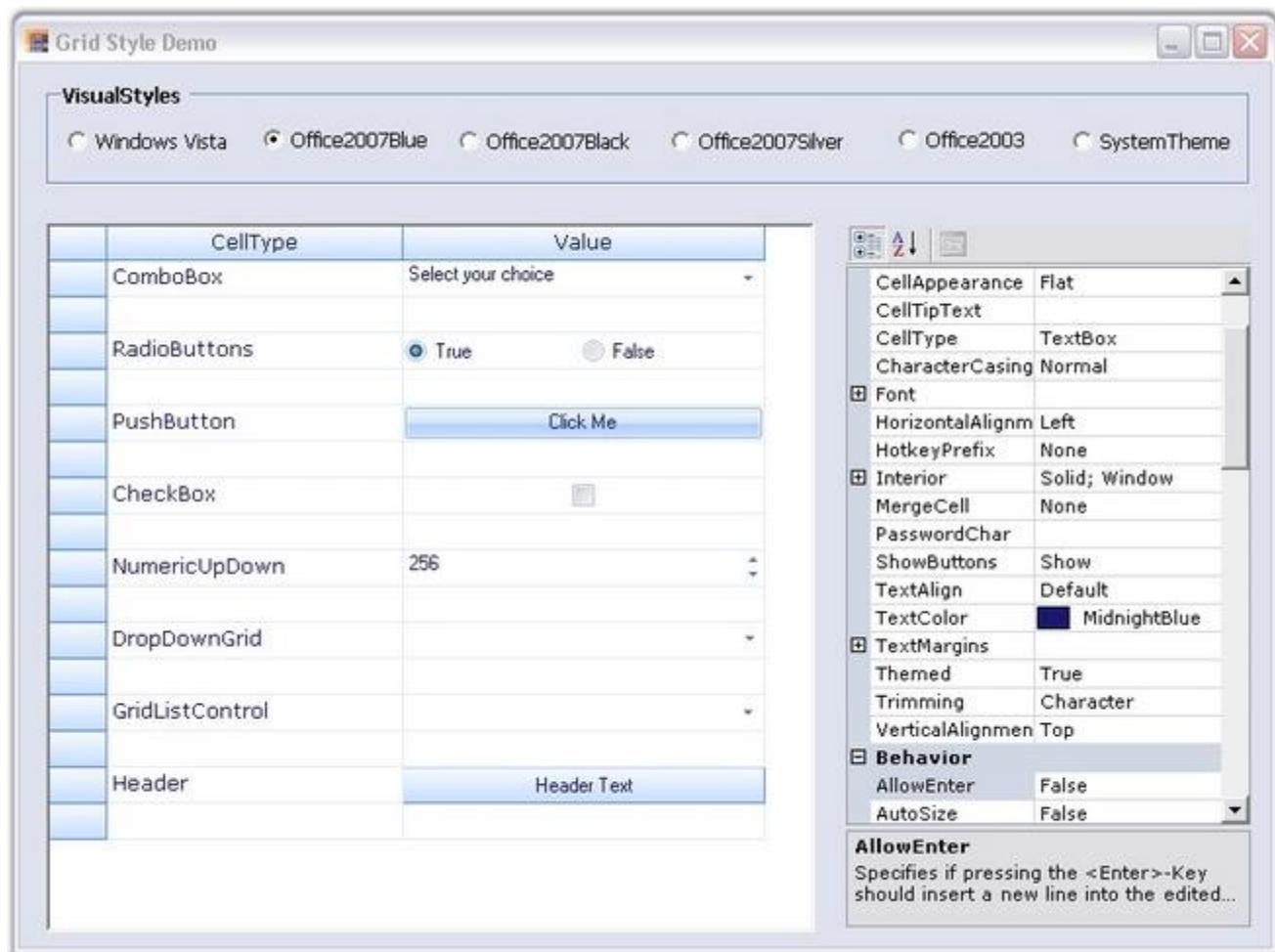


Figure 157: Grid Styles

The figure above displays various visual styles in the **VisualStyles** group box in the UI for the Essential Grid.

Following code example illustrates how to set the visual style for the Grid control.

**[C#]**

```
// Sets an Office 2007 Blue skin theme to the Essential Grid control.
gridControl1.GridVisualStyles =
Syncfusion.Windows.Forms.GridVisualStyles.Office2007Blue;
```

**[VB.NET]**

```
' Sets an Office 2007 Blue skin theme to the Essential Grid control.
```

```
gridControl1.GridVisualStyles =  
Syncfusion.Windows.Forms.GridVisualStyles.Office2007Blue
```

**ThemesEnabled** property determines whether XP Themes (visual styles) can be used for this control or not, when available.

Following code example illustrates how to set the theme for the Grid control.

**[C#]**

```
this.gridControl1.ThemesEnabled = true;
```

**[VB .NET]**

```
Me.gridControl1.ThemesEnabled = True
```

#### 4.1.4.13.3 Grid Control Designer

This section elaborates upon Grid control's edit designer. Grid control has an excellent user friendly design-time support. A Grid control's edit designer is added to the grid to ease the process of designing a Grid control on a cell level. Using the editor, the Grid can be modified, saved and loaded to XML formatted files or to SOAP formatted templates.

Following is the step-by-step procedure to edit Grid control's cell styles using GridControl Designer window:

1. Right click the Grid control. A context menu is displayed.
2. Select **Edit** from the context menu drop-down. The figure below illustrates this user-action:

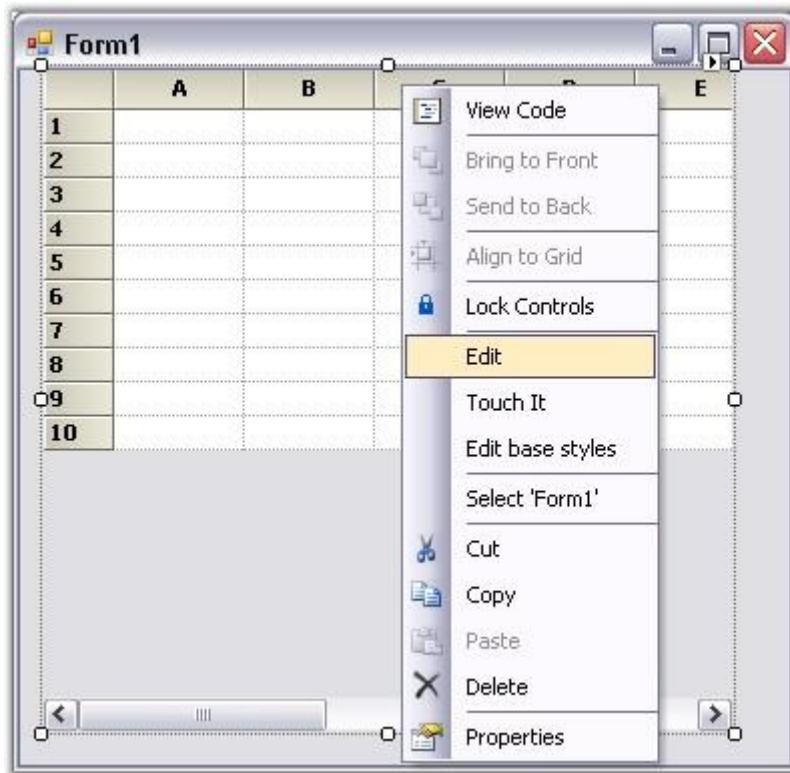


Figure 158: Grid control with the Context Menu



**Note:**

1. **The Editor opens up on the right hand side of the page and the Grid Properties tab is highlighted by default.**
2. **The cell content/styles and general grid properties can be modified under the Grid Properties tab.**

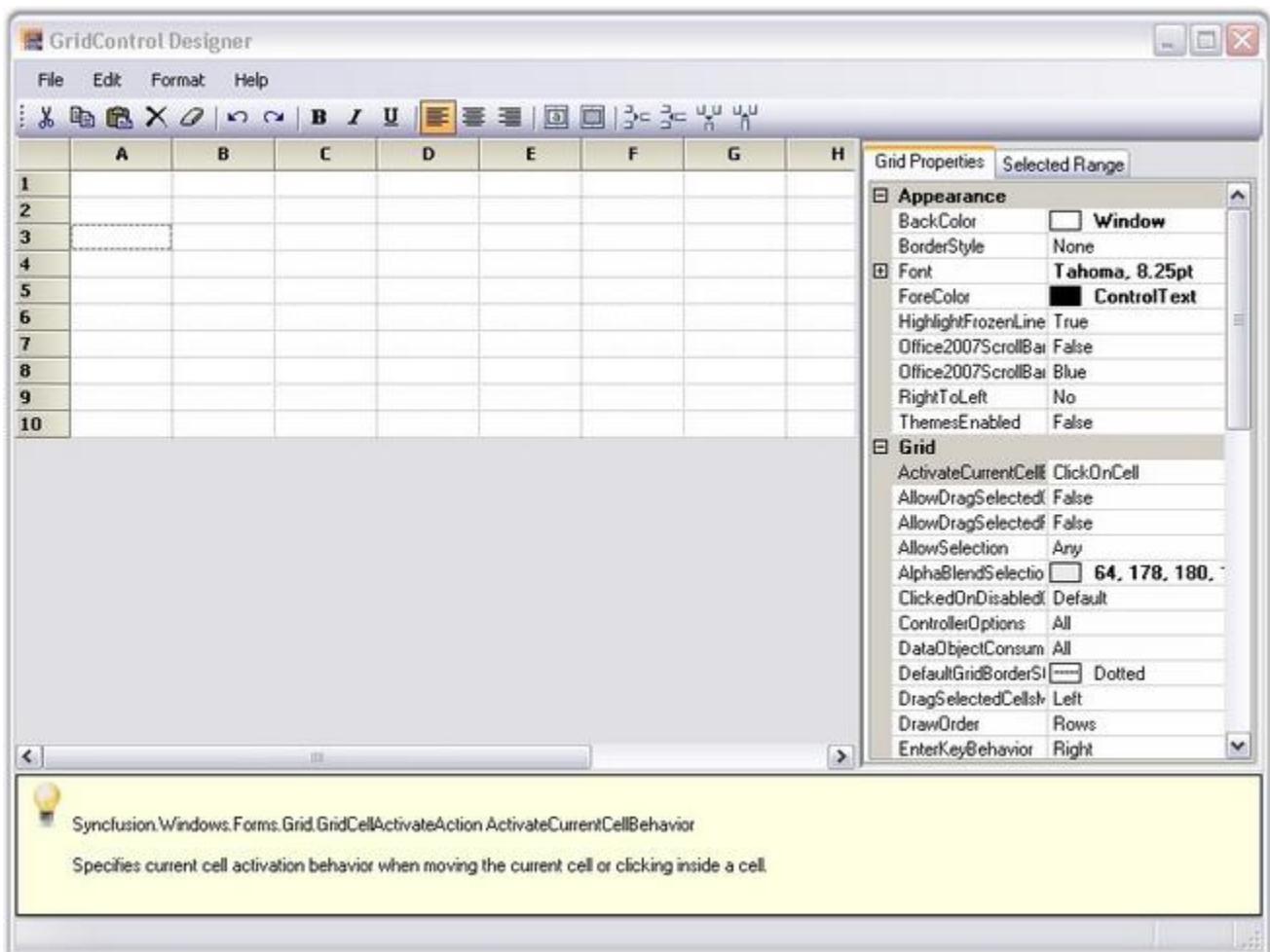


Figure 159: Grid Properties

The figure above shows the **GridControl Designer** window with **Grid Properties** tab.

### Modifying the Properties of a Selected Range

To modify the properties of a selected range, follow the steps listed below:

1. Select a range of cells.
2. Click **Selected Range** tab to view the **Property grid** for the selection.

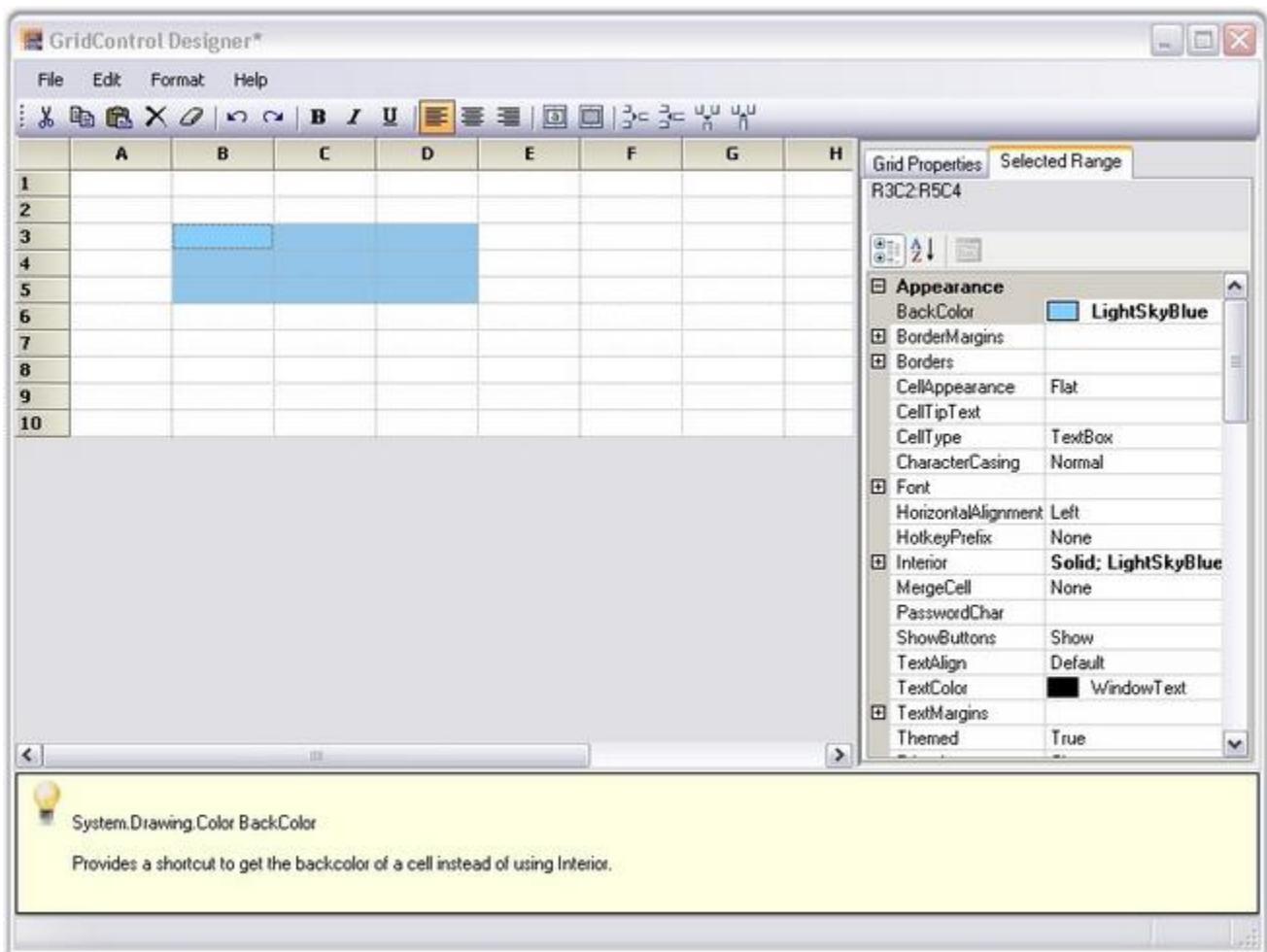


Figure 160: Grid control Designer

The figure above shows the **Property grid** under the **Selected Range** tab in the GridControl Designer window.

3. Make the required modifications in the property grid so that they are affected in the selected range of the main grid.
4. Exit editor after the modifications are done.



**Note:** *The system prompts you to save the changes to the Grid control in the designer, if exited without saving.*

#### 4.1.4.13.4 Grid Properties

Essential Grid provides support to customize the appearance and behavior of the grid cells. This section would provide you more insight on the properties affecting the Appearance, Print Styles, and Scroll Bar Settings available. It includes the following topics.

#### 4.1.4.13.4.1 Appearance Properties

The properties that majorly affect the appearance cells and data in cells of a grid can be named as Appearance properties.

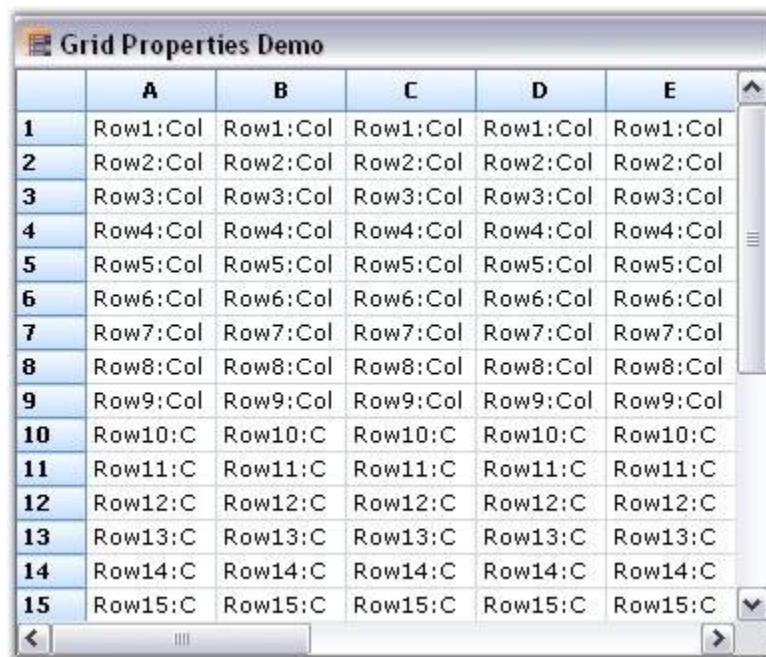


Figure 161: Grid Control

The following properties are used to customize the appearance of Grid.

- **TransparentBackground**-Specifies whether to display grid with background image. When this property is set to false, the background image will not be displayed even if it is set by using the BackgroundImage property.

The following code examples can be used to set this property:

##### 1. Using C#

```
[C#]
```

```
// Enable TransparentBackground property.  
this.gridControl1.TransparentBackground = true;
```

## 2. Using VB.NET

[VB .NET]

```
' Enable TransparentBackground property.  
Me.gridControl1.TransparentBackground = True
```

The following illustration shows how the Grid in "Figure 1" is transformed when the TransparentBackground property is set to true.

	A	B	C	D	E
1	Row1:Col	Row1:Col	Row1:Col	Row1:Col	Row1:Col
2	Row2:Col	Row2:Col	Row2:Col	Row2:Col	Row2:Col
3	Row3:Col	Row3:Col	Row3:Col	Row3:Col	Row3:Col
4	Row4:Col	Row4:Col	Row4:Col	Row4:Col	Row4:Col
5	Row5:Col	Row5:Col	Row5:Col	Row5:Col	Row5:Col
6	Row6:Col	Row6:Col	Row6:Col	Row6:Col	Row6:Col
7	Row7:Col	Row7:Col	Row7:Col	Row7:Col	Row7:Col
8	Row8:Col	Row8:Col	Row8:Col	Row8:Col	Row8:Col
9	Row9:Col	Row9:Col	Row9:Col	Row9:Col	Row9:Col
10	Row10:C	Row10:C	Row10:C	Row10:C	Row10:C

Figure 162: Grid with TransparentBackground property set to True

- **DisplayHorzLines**-Specifies whether horizontal grid lines marking the cells are to be displayed. Default value is set to *true*.

The following code examples can be used to set this property:

## 1. Using C#

[C#]

```
// Enable DisplayHorzLines property.  
this.gridControl1.Properties.DisplayHorzLines = false;
```

## 2. Using VB.NET

[VB .NET]

```
' Enable DisplayHorzLines property.
Me.gridControl1.Properties.DisplayHorzLines = False
```

The following illustration shows how the Grid in "Figure 1" is transformed when the Properties.DisplayHorzLines property is set to false.

	A	B	C	D	E
1	Row1:Col	Row1:Col	Row1:Col	Row1:Col	Row1:Col
2	Row2:Col	Row2:Col	Row2:Col	Row2:Col	Row2:Col
3	Row3:Col	Row3:Col	Row3:Col	Row3:Col	Row3:Col
4	Row4:Col	Row4:Col	Row4:Col	Row4:Col	Row4:Col
5	Row5:Col	Row5:Col	Row5:Col	Row5:Col	Row5:Col
6	Row6:Col	Row6:Col	Row6:Col	Row6:Col	Row6:Col
7	Row7:Col	Row7:Col	Row7:Col	Row7:Col	Row7:Col
8	Row8:Col	Row8:Col	Row8:Col	Row8:Col	Row8:Col
9	Row9:Col	Row9:Col	Row9:Col	Row9:Col	Row9:Col
10	Row10:C	Row10:C	Row10:C	Row10:C	Row10:C

Figure 163: Horizontal Lines hidden in Grid

- **DisplayVertLines**-Specifies whether vertical grid lines marking the cells are to be displayed. Default value is set to *true*.

The following code examples can be used to set this property:

#### 1. Using C#

```
[C#]

// Enable DisplayVertLines property.
this.gridControl1.Properties.DisplayVertLines = false;
```

#### 2. Using VB.NET

```
[VB .NET]

' Enable DisplayVertLines property.
Me.gridControl1.Properties.DisplayVertLines = False
```

The following illustration shows how the Grid in "Figure 1" is transformed when the Properties.DisplayVertLines property is set to false.

	A	B	C	D	E
1	Row1:Col	Row1:Col	Row1:Col	Row1:Col	Row1:Col
2	Row2:Col	Row2:Col	Row2:Col	Row2:Col	Row2:Col
3	Row3:Col	Row3:Col	Row3:Col	Row3:Col	Row3:Col
4	Row4:Col	Row4:Col	Row4:Col	Row4:Col	Row4:Col
5	Row5:Col	Row5:Col	Row5:Col	Row5:Col	Row5:Col
6	Row6:Col	Row6:Col	Row6:Col	Row6:Col	Row6:Col
7	Row7:Col	Row7:Col	Row7:Col	Row7:Col	Row7:Col
8	Row8:Col	Row8:Col	Row8:Col	Row8:Col	Row8:Col
9	Row9:Col	Row9:Col	Row9:Col	Row9:Col	Row9:Col
10	Row10:C	Row10:C	Row10:C	Row10:C	Row10:C

*Figure 164: Vertical Lines hidden in Grid*

- **ColHeaders**-Specifies whether column headers are to be displayed. Default value is set to *true*.

The following code examples can be used to set this property:

1. Using C#

```
[C#]  
  
// Hiding the column headers.  
this.gridControl1.Properties.ColHeaders = false;
```

2. Using VB.NET

```
[VB.NET]  
  
' Hiding the column headers.  
Me.gridControl1.Properties.ColHeaders = False
```

The following illustration shows how the Grid in "Figure 1" is transformed when the Properties.ColHeaders property is set to false.

<b>1</b>	Row1:Col	Row1:Col	Row1:Col	Row1:Col	Row1:Col
<b>2</b>	Row2:Col	Row2:Col	Row2:Col	Row2:Col	Row2:Col
<b>3</b>	Row3:Col	Row3:Col	Row3:Col	Row3:Col	Row3:Col
<b>4</b>	Row4:Col	Row4:Col	Row4:Col	Row4:Col	Row4:Col
<b>5</b>	Row5:Col	Row5:Col	Row5:Col	Row5:Col	Row5:Col
<b>6</b>	Row6:Col	Row6:Col	Row6:Col	Row6:Col	Row6:Col
<b>7</b>	Row7:Col	Row7:Col	Row7:Col	Row7:Col	Row7:Col
<b>8</b>	Row8:Col	Row8:Col	Row8:Col	Row8:Col	Row8:Col
<b>9</b>	Row9:Col	Row9:Col	Row9:Col	Row9:Col	Row9:Col
<b>10</b>	Row10:C	Row10:C	Row10:C	Row10:C	Row10:C

*Figure 165: Column Headers hidden in Grid*

- **RowHeaders**-Specifies whether row headers are to be displayed. Default value is set to *true*.

The following code examples can be used to set this property:

1. Using C#

```
[C#]

// Hiding the row headers.
this.gridControl1.Properties.RowHeaders = false;
```

2. Using VB.NET

```
[VB .NET]

' Hiding the row headers.
Me.gridControl1.Properties.RowHeaders = False
```

The following illustration shows how the Grid in "Figure 1" is transformed when the Properties.RowHeaders property is set to false.

A	B	C	D	E
Row1:Col	Row1:Col	Row1:Col	Row1:Col	Row1:Col
Row2:Col	Row2:Col	Row2:Col	Row2:Col	Row2:Col
Row3:Col	Row3:Col	Row3:Col	Row3:Col	Row3:Col
Row4:Col	Row4:Col	Row4:Col	Row4:Col	Row4:Col
Row5:Col	Row5:Col	Row5:Col	Row5:Col	Row5:Col
Row6:Col	Row6:Col	Row6:Col	Row6:Col	Row6:Col
Row7:Col	Row7:Col	Row7:Col	Row7:Col	Row7:Col
Row8:Col	Row8:Col	Row8:Col	Row8:Col	Row8:Col
Row9:Col	Row9:Col	Row9:Col	Row9:Col	Row9:Col
Row10:C	Row10:C	Row10:C	Row10:C	Row10:C

Figure 166: Row Headers hidden in Grid

- **Buttons3D**-Specifies if the row and column headers should have a three dimensional look which in turn makes headers visually appealing. If this property is set to false, the row and column headers will appear flat. Default value is set to *true*.

The following code examples can be used to set this property:

#### 1. Using C#

[C#]

```
// Enable Buttons3D property.
this.gridControl1.Properties.Buttons3D = false;
```

#### 2. Using VB.NET

[VB .NET]

```
' Enable Buttons3D property.
Me.gridControl1.Properties.Buttons3D = False
```

The following illustration shows how the Grid in "Figure 1" is transformed when the Properties.Buttons3D property is set to false.

	<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>	<b>E</b>
<b>1</b>	Row1:Col	Row1:Col	Row1:Col	Row1:Col	Row1:Col
<b>2</b>	Row2:Col	Row2:Col	Row2:Col	Row2:Col	Row2:Col
<b>3</b>	Row3:Col	Row3:Col	Row3:Col	Row3:Col	Row3:Col
<b>4</b>	Row4:Col	Row4:Col	Row4:Col	Row4:Col	Row4:Col
<b>5</b>	Row5:Col	Row5:Col	Row5:Col	Row5:Col	Row5:Col
<b>6</b>	Row6:Col	Row6:Col	Row6:Col	Row6:Col	Row6:Col
<b>7</b>	Row7:Col	Row7:Col	Row7:Col	Row7:Col	Row7:Col
<b>8</b>	Row8:Col	Row8:Col	Row8:Col	Row8:Col	Row8:Col
<b>9</b>	Row9:Col	Row9:Col	Row9:Col	Row9:Col	Row9:Col
<b>10</b>	Row10:C	Row10:C	Row10:C	Row10:C	Row10:C

Figure 167: Buttons3D effect disabled for Rows and Columns Headers in Grid

- **GridLineColor**-Specifies the color for the grid lines (for example, active border). Default value is set to *GrayText*.

The following code examples can be used to set this property:

#### 1. Using C#

```
[C#]
// Specify the color for the grid lines.
this.gridControl1.Properties.GridLineColor =
System.Drawing.Color.IndianRed;
```

#### 2. Using VB.NET

```
[VB .NET]
' Specify the color for the grid lines.
Me.gridControl1.Properties.GridLineColor =
System.Drawing.Color.IndianRed
```

The following illustration shows how the Grid in "Figure 1" is transformed when the `Properties.GridLineColor` property is set to IndianRed.

	A	B	C	D	E
1	Row1:Col	Row1:Col	Row1:Col	Row1:Col	Row1:Col
2	Row2:Col	Row2:Col	Row2:Col	Row2:Col	Row2:Col
3	Row3:Col	Row3:Col	Row3:Col	Row3:Col	Row3:Col
4	Row4:Col	Row4:Col	Row4:Col	Row4:Col	Row4:Col
5	Row5:Col	Row5:Col	Row5:Col	Row5:Col	Row5:Col
6	Row6:Col	Row6:Col	Row6:Col	Row6:Col	Row6:Col
7	Row7:Col	Row7:Col	Row7:Col	Row7:Col	Row7:Col
8	Row8:Col	Row8:Col	Row8:Col	Row8:Col	Row8:Col
9	Row9:Col	Row9:Col	Row9:Col	Row9:Col	Row9:Col
10	Row10:C	Row10:C	Row10:C	Row10:C	Row10:C

Figure 168: Grid with Line Color set to Red

- **BackgroundImage**-Enables to insert a background image for the grid.

The following code examples can be used to set this property:

#### 1. Using C#

[C#]

```
// Specify the background image.
this.gridControl1.BackgroundImage =
Image.FromFile(FindImageFile(@"..\..\..\..\\pic.jpg"));
```

#### 2. Using VB.NET

[VB.NET]

```
' Specify the background image.
Me.gridControl1.BackgroundImage =
Image.FromFile(FindImageFile("../..\..\..\\pic.jpg"))
```

The following illustration shows how the Grid in "Figure 1" is transformed when the BackgroundImage property is set.

	A	B	C	D	E
1	Row1:Col	Row1:Col	Row1:Col	Row1:Col	Row1:Col
2	Row2:Col	Row2:Col	Row2:Col	Row2:Col	Row2:Col
3	Row3:Col	Row3:Col	Row3:Col	Row3:Col	Row3:Col
4	Row4:Col	Row4:Col	Row4:Col	Row4:Col	Row4:Col
5	Row5:Col	Row5:Col	Row5:Col	Row5:Col	Row5:Col
6	Row6:Col	Row6:Col	Row6:Col	Row6:Col	Row6:Col
7	Row7:Col	Row7:Col	Row7:Col	Row7:Col	Row7:Col
8	Row8:Col	Row8:Col	Row8:Col	Row8:Col	Row8:Col
9	Row9:Col	Row9:Col	Row9:Col	Row9:Col	Row9:Col
10	Row10:C	Row10:C	Row10:C	Row10:C	Row10:C

Figure 169: Background Image set for Grid

- **TextColor**-Specifies the color of the text in the grid.

The following code examples can be used to set this property:

#### 1. Using C#

[C#]

```
// Set grid text color.
this.gridControl1.TableStyle.TextColor = Color.MidnightBlue;
```

#### 2. Using VB.NET

[VB.NET]

```
' Set grid text color.
Me.gridControl1.TableStyle.TextColor = Color.MidnightBlue
```

The following illustration shows how the Grid in "Figure 1" is transformed when the TableStyle.TextColor property is set to MidnightBlue.

	A	B	C	D	E
<b>1</b>	Row1:Col	Row1:Col	Row1:Col	Row1:Col	Row1:Col
<b>2</b>	Row2:Col	Row2:Col	Row2:Col	Row2:Col	Row2:Col
<b>3</b>	Row3:Col	Row3:Col	Row3:Col	Row3:Col	Row3:Col
<b>4</b>	Row4:Col	Row4:Col	Row4:Col	Row4:Col	Row4:Col
<b>5</b>	Row5:Col	Row5:Col	Row5:Col	Row5:Col	Row5:Col
<b>6</b>	Row6:Col	Row6:Col	Row6:Col	Row6:Col	Row6:Col
<b>7</b>	Row7:Col	Row7:Col	Row7:Col	Row7:Col	Row7:Col
<b>8</b>	Row8:Col	Row8:Col	Row8:Col	Row8:Col	Row8:Col
<b>9</b>	Row9:Col	Row9:Col	Row9:Col	Row9:Col	Row9:Col
<b>10</b>	Row10:C	Row10:C	Row10:C	Row10:C	Row10:C

*Figure 170: Grid with Text Color set to MidnightBlue*

- **BackColor**-Specifies the color of the grid line marker when the user resizes rows or columns by dragging the row or column headers. Default value is set to *Red*.

The following code examples can be used to set this property:

1. Using C#

```
[C#]  
  
// Set the grid background color.  
this.gridControl1.BackColor = Color.Beige;
```

2. Using VB.NET

```
[VB .NET]  
  
' Set the grid background color.  
Me.gridControl1.BackColor = Color.Beige
```

The following illustration shows how the Grid in "Figure 1" is transformed when the BackColor property is set to Beige.

	A	B	C	D	E
1	Row1:Col	Row1:Col	Row1:Col	Row1:Col	Row1:Col
2	Row2:Col	Row2:Col	Row2:Col	Row2:Col	Row2:Col
3	Row3:Col	Row3:Col	Row3:Col	Row3:Col	Row3:Col
4	Row4:Col	Row4:Col	Row4:Col	Row4:Col	Row4:Col
5	Row5:Col	Row5:Col	Row5:Col	Row5:Col	Row5:Col
6	Row6:Col	Row6:Col	Row6:Col	Row6:Col	Row6:Col
7	Row7:Col	Row7:Col	Row7:Col	Row7:Col	Row7:Col
8	Row8:Col	Row8:Col	Row8:Col	Row8:Col	Row8:Col
9	Row9:Col	Row9:Col	Row9:Col	Row9:Col	Row9:Col
10	Row10:C	Row10:C	Row10:C	Row10:C	Row10:C

Figure 171: Grid with Background Color set to Beige

- **ResizingCellsLinesColor**-Specifies the color for the grid lines (for example, active border). Default value is set to *GrayText*.

The following code examples can be used to set this property:

#### 1. Using C#

[C#]

```
// Specify the color for the grid line marker while resizing rows and
// columns.
this.gridControl1.Properties.ResizingCellsLinesColor =
Color.PaleVioletRed;
```

#### 2. Using VB.NET

[VB .NET]

```
' Specify the color for the grid line marker while resizing rows and
// columns.
Me.gridControl1.Properties.ResizingCellsLinesColor =
Color.PaleVioletRed
```

- **Borders**-Specifies settings for Top, Left, Bottom and Right borders.

The following code examples can be used to set this property:

#### 1. Using C#

[C#]

```
// Set border settings for the grid.  
this.gridControl1.TableStyle.Borders.All = new  
GridBorder(GridBorderStyle.Solid, Color.SteelBlue);
```

## 2. Using VB.NET

**[VB .NET]**

```
' Set border settings for the grid.  
Me.gridControl1.TableStyle.Borders.All = New  
GridBorder(GridBorderStyle.Solid, Color.SteelBlue)
```

- **FixedLinesColor**-Specifies the color of frozen grid lines (for example, row or column headers). Default value is set to *ActiveCaption*.

The following code examples can be used to set this property:

### 1. Using C#

**[C#]**

```
// Set the color of frozen grid lines.  
this.gridControl1.Properties.FixedLinesColor = Color.YellowGreen;
```

### 2. Using VB.NET

**[VB .NET]**

```
' Set the color of frozen grid lines.  
Me.gridControl1.Properties.FixedLinesColor = Color.YellowGreen
```

A sample demonstrating these properties is available under the following sample installation path.

**<Install Location>\Syncfusion\EssentialStudio\Version  
Number\Windows\Grid.Windows\Samples\2.0\Appearance\Grid Properties Demo**

#### *4.1.4.13.4.2 Print Properties*

The following properties are associated with printing in Grid. They are generally referred to as Print Styles.

- **BlackWhite**-Specifies if the grid should be printed only in black and white.

The following code examples illustrate how to set this property:

1. Using C#

[C#]

```
// Specify if the grid should print only in black and white.  
this.gridControl1.Properties.BlackWhite = true;
```

2. Using VB.NET

[VB.NET]

```
' Specify if the grid should print only in black and white.  
Me.gridControl1.Properties.BlackWhite = True
```

- **Printing**-Prints the grid.

The following code examples illustrate how to set this property:

1. Using C#

[C#]

```
// Prints the grid.  
this.gridControl1.Properties.Printing = true;
```

2. Using VB.NET

[VB.NET]

```
' Prints the grid.  
Me.gridControl1.Properties.Printing = True
```

- **PrintFrame**-Specifies the appearance of a frame around the grid while printing.

The following code examples illustrate how to set this property:

1. Using C#

[C#]

```
// Specify if a frame should be drawn around the grid while printing.  
this.gridControl1.Properties.PrintFrame = true;
```

## 2. Using VB.NET

**[VB .NET]**

```
' Specify if a frame should be drawn around the grid while printing.  
Me.gridControl1.Properties.PrintFrame = True
```

- **PrintColHeader**-Specifies if column headers should be printed.

The following code examples illustrate how to set this property:

## 1. Using C#

**[C#]**

```
// Specify if column headers should be printed.  
this.gridControl1.Properties.PrintColHeader = true;
```

## 2. Using VB.NET

**[VB .NET]**

```
' Specify if column headers should be printed.  
Me.gridControl1.Properties.PrintColHeader = True
```

- **PrintRowHeader**-Specifies if row headers should be printed.

The following code examples illustrate how to set this property:

## 1. Using C#

**[C#]**

```
// Specify if row headers should be printed.  
this.gridControl1.Properties.PrintRowHeader = true;
```

## 2. Using VB.NET

**[VB .NET]**

```
' Specify if row headers should be printed.  
Me.gridControl1.Properties.PrintRowHeader = True
```

- **CenterVertical**-Specifies if the grid should be centered vertically on printing.

The following code examples illustrate how to set this property:

1. Using C#

[C#]

```
// Specify if the grid should be centered vertically on printing.  
this.gridControl1.Properties.CenterVertical = true;
```

2. Using VB.NET

[VB .NET]

```
' Specify if the grid should be centered vertically on printing.  
Me.gridControl1.Properties.CenterVertical = True
```

- **PrintHorzLines**-Specifies if horizontal lines of the grid should be printed.

The following code examples illustrate how to set this property:

1. Using C#

[C#]

```
// Specify if horizontal lines of the grid should be printed.  
this.gridControl1.Properties.PrintHorzLines = true;
```

2. Using VB.NET

[VB .NET]

```
' Specify if horizontal lines of the grid should be printed.  
Me.gridControl1.Properties.PrintHorzLines = True
```

- **PrintVertLines**-Specifies if vertical lines of the grid should be printed.

The following code examples illustrate how to set this property:

1. Using C#

[C#]

```
// Specify if vertical lines of the grid should be printed.  
this.gridControl1.Properties.PrintVertLines = true;
```

## 2. Using VB.NET

[VB.NET]

```
' Specify if vertical lines of the grid should be printed.  
Me.gridControl1.Properties.PrintVertLines = True
```

- **CenterHorizontal**—specifies if a grid should be centered horizontally while printing.

The following code examples illustrate how to set this property in the Grid control.

[C#]

```
// Specify if the grid should be centered horizontally while printing.  
this.gridControl1.Properties.CenterHorizontal = true;
```

[VB.NET]

```
' Specify if the grid should be centered vertically while printing.  
Me.gridControl1.Model.Properties.CenterHorizontal = False
```

A sample demonstrating these properties is available under the following sample installation path.

**<Install Location>\Syncfusion\EssentialStudio\Version  
Number\Windows\Grid.Windows\Samples\2.0\Print**

### 4.1.4.13.4.3 Scroll Bar Properties

Essential Grid provides support to control the functionalities and appearance of the grid scroll bars.

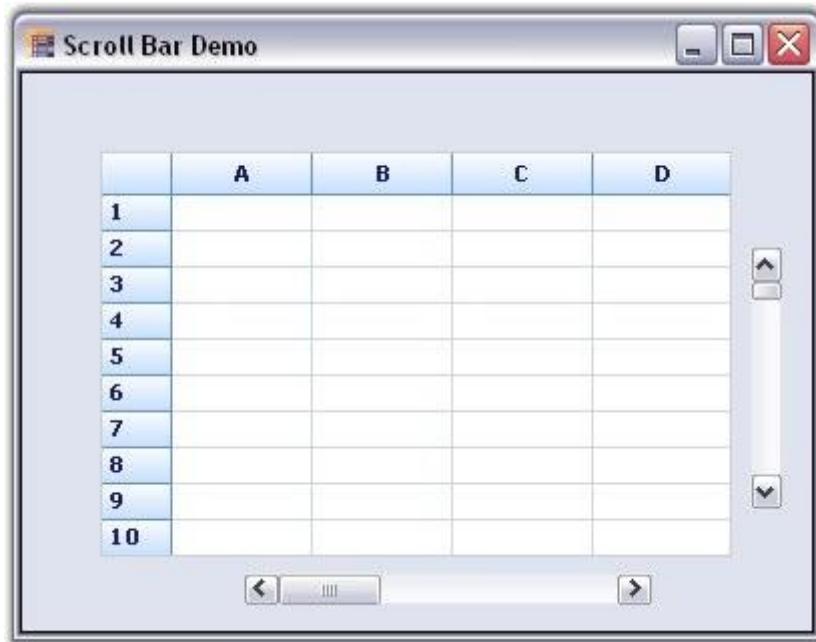


Figure 172: Grid with Horizontal and Vertical Scroll Bars

The following properties are associated with scrolling in grid.

- **HscrollPixel**-Specifies whether to enable/disable horizontal pixel scrolling for the grid. Default value is set to *false*.  
The following code examples can be used to set this property:

1. Using C#

[C#]

```
// Enable horizontal pixel scrolling for the grid.  
this.gridControl1.HScrollPixel = true;
```

2. Using VB.NET

[VB .NET]

```
' Enable horizontal pixel scrolling for the grid.  
Me.gridControl1.HScrollPixel = True
```

- **VscrollPixel**-Specifies whether to enable/disable vertical pixel scrolling for the grid. Default value is set to *false*.

The following code examples can be used to set this property:

1. Using C#

[C#]

```
// Enable vertical pixel scrolling for the grid.  
this.gridControl1.VScrollPixel = true;
```

2. Using VB.NET

[VB.NET]

```
' Enable vertical pixel scrolling for the grid.  
Me.gridControl1.VScrollPixel = True
```

- **HorizontalScrollTips**-Specifies if the control should display scroll tips while the user is dragging a horizontal scroll bar thumb. Default value is set to *false*.

The following code examples can be used to set this property:

1. Using C#

[C#]

```
// Specify whether scroll tips should be displayed while dragging the  
horizontal scroll bar thumb.  
this.gridControl1.HorizontalScrollTips = true;
```

2. Using VB.NET

[VB.NET]

```
' Specify whether scroll tips should be displayed while dragging the  
horizontal scroll bar thumb.  
Me.gridControl1.HorizontalScrollTips = True
```

3. The following illustration shows how the Grid in "Figure 1" is transformed when the **HorizontalScrollTips** property is set to true.



Figure 173: `HorizontalScrollTips = "True"`

- **VerticalScrollTips**-Specifies if the control should display scroll tips while the user is dragging a vertical scroll bar thumb. Default value is set to *false*.

The following code examples can be used to set this property:

1. Using C#

[C#]

```
// Specify whether scroll tips should be displayed while dragging the
vertical scroll bar thumb.
this.gridControl1.VerticalScrollTips = true;
```

2. Using VB.NET

[VB .NET]

```
' Specify whether scroll tips should be displayed while dragging the
vertical scroll bar thumb.
Me.gridControl1.VerticalScrollTips = True
```

The following illustration shows how the Grid in "Figure 1" is transformed when the `VerticalScrollTips` property is set to true.



Figure 174: VerticalScrollTips = "True"

- **HscrollBehavior**-Specifies the behavior of the horizontal scroll bar. GridScrollbarMode enumeration provides the following options to control the scroll bar behavior: Automatic, AutoScroll, DetectIfShared, DisableAutoScroll, Disabled, Enabled and Shared.  
The following code examples can be used to set this property:

1. Using C#

[C#]

```
// Set the behavior of the horizontal scroll bar.  
this.gridControl1.HscrollBehavior = GridScrollbarMode.Shared;
```

2. Using VB.NET

[VB .NET]

```
' Set the behavior of the horizontal scroll bar.  
Me.gridControl1.HscrollBehavior = GridScrollbarMode.Shared
```

- **VscrollBehavior**-Specifies the behavior of the vertical scroll bar. GridScrollbarMode enumeration provides the following options to control the scroll bar behavior: Automatic, AutoScroll, DetectIfShared, DisableAutoScroll, Disabled, Enabled and Shared.  
The following code examples can be used to set this property:

1. Using C#

[C#]

```
// Set the behavior of the vertical scroll bar.  
this.gridControl1.VScrollBehavior = GridScrollbarMode.Shared;
```

2. Using VB.NET

[VB .NET]

```
' Set the behavior of the vertical scroll bar.  
Me.gridControl1.VScrollBehavior = GridScrollbarMode.Shared
```

- **HorizontalThumbTrack**-Specifies whether the control should scroll while the user is dragging the horizontal scroll bar thumb. Default value is set to *false*.  
The following code examples can be used to set this property:

1. Using C#

[C#]

```
// Specify whether the control should scroll while dragging the  
horizontal scroll bar thumb.  
this.gridControl1.HorizontalThumbTrack = true;
```

2. Using VB.NET

[VB .NET]

```
' Specify whether the control should scroll while dragging the  
horizontal scroll bar thumb.  
Me.gridControl1.HorizontalThumbTrack = True
```

The following illustration shows how the Grid in "Figure 1" is transformed when the HorizontalThumbTrack property is set to true.

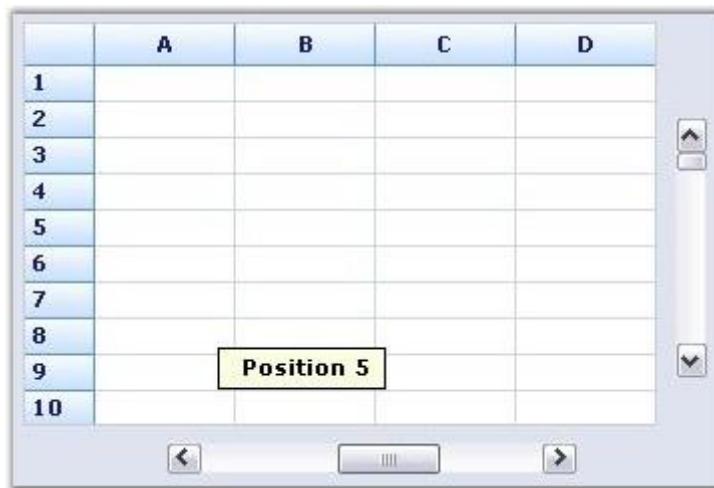


Figure 175: `HorizontalThumbTrack = "True"`

- **VerticalThumbTrack**-Specifies whether the control should scroll while the user is dragging vertical scroll bar thumb. Default value is set to *false*.  
The following code examples can be used to set this property:

1. Using C#

[C#]

```
// Specify whether the control should scroll while dragging the
vertical scroll bar thumb.
this.gridControl1.VerticalThumbTrack = true;
```

2. Using VB.NET

[VB .NET]

```
' Specify whether the control should scroll while dragging the vertical
scroll bar thumb.
Me.gridControl1.VerticalThumbTrack = True
```

The following illustration shows how the Grid in "Figure 1" is transformed when the `VerticalThumbTrack` property is set to true.

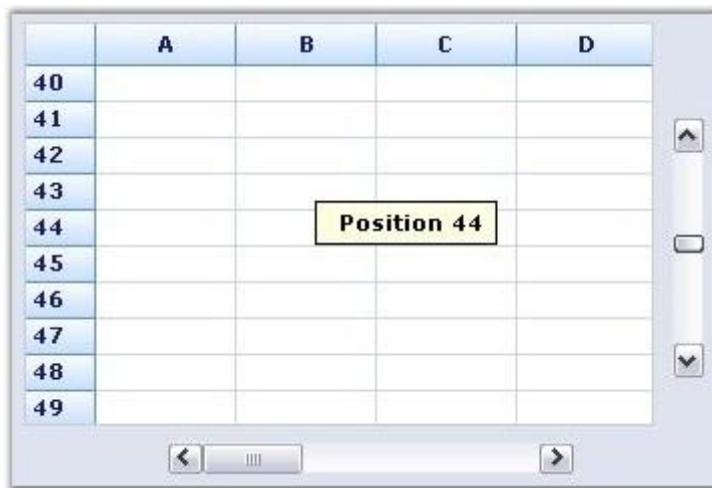


Figure 176: *VerticalThumbTrack = "True"*

- **Office2007ScrollBars**-Toggles between standard and Office 2007 scroll bars. Default value is set to *false*.

The following code examples can be used to set this property:

1. Using C#

[C#]

```
// Toggle to Office 2007 scroll bar.  
this.gridControl1.Office2007ScrollBars = true;
```

2. Using VB.NET

[VB .NET]

```
' Toggle to Office 2007 scroll bar.  
Me.gridControl1.Office2007ScrollBars = True
```

- **Office2007ScrollBarsColorScheme**-Specifies the style for Office 2007 scroll bars. Default value is set to *Blue*.

The following code examples can be used to set this property:

1. Using C#

[C#]

```
// Set the style for Office 2007 scroll bar.  
this.gridControl1.Office2007ScrollBarsColorScheme =  
Office2007ColorScheme.Blue;
```

## 2. Using VB.NET

[VB.NET]

```
' Set the style for Office 2007 scroll bar.  
Me.gridControl1.Office2007ScrollBarsColorScheme =  
Office2007ColorScheme.Blue
```

- **ScrollFrozen**-Defines scroll behavior when user moves current cell with arrow keys into frozen cells area. Default value is set to *true*.

The following code examples can be used to set this property:

### 1. Using C#

[C#]

```
// Define scroll behavior in frozen cells.  
this.gridControl1.ScrollFrozen = true;
```

### 2. Using VB.NET

[VB.NET]

```
' Define scroll behavior in frozen cells.  
Me.gridControl1.ScrollFrozen = True
```

- **ScrollTipFormat**-Specifies the text to be displayed in the ScrollTip window with a place holder for scroll position.

The following code examples can be used to set this property:

### 1. Using C#

[C#]

```
// Set the text to be displayed in the ScrollTip window with a place  
holder for scroll position.
```

```
this.gridControl1.ScrollTipFormat = "Position {0}";
```

## 2. Using VB.NET

[VB.NET]

```
' Set the text to be displayed in the ScrollTip window with a place
holder for scroll position.
Me.gridControl1.ScrollTipFormat = "Position {0}"
```

- **AutoScrolling**-Specifies whether to enable/disable automatic scrolling.  
The following code examples can be used to set this property:

### 1. Using C#

[C#]

```
// Enable AutoScrolling.
this.gridControl1.AutoScrolling = true;
```

### 2. Using VB.NET

[VB.NET]

```
' Enable AutoScrolling.
Me.gridControl1.AutoScrolling = True
```

- **Hscroll**-Specifies whether to enable/disable horizontal scroll bar.  
The following code examples can be used to set this property:

### 1. Using C#

[C#]

```
// Enable horizontal scroll bar.
this.gridControl1.HScroll = true;
```

### 2. Using VB.NET

[VB.NET]

```
' Enable horizontal scroll bar.  
Me.gridControl1.HScroll = True
```

- **VScroll**-Specifies whether to enable/disable vertical scroll bar.  
The following code examples can be used to set this property:

1. Using C#

```
[C#]
```

```
// Enable vertical scroll bar.  
this.gridControl1.VScroll = true;
```

2. Using VB.NET

```
[VB .NET]
```

```
' Enable vertical scroll bar.  
Me.gridControl1.VScroll = True
```

A sample demonstrating these properties is available under the following sample installation path.

*<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Windows\Samples\2.0\Zoom and Scrolling\Scroll Bar Demo*

#### 4.1.4.13.5 Custom Drawing

Essential Grid enables custom drawing on its cells. Custom Drawing denotes adding text and drawings such as lines, polygons, etc., to the cell. It has custom draw events such as **CellDrawn** event and **DrawCell** event. Using these events, you can provide your applications the exact appearance that you desire.

- **DrawCell**-This event is handled for every cell before the grid draws a specified cell. This event is generally used to add custom drawing to a cell. It can also be used to draw shapes like lines, polygons, etc.,
- **CellDrawn event**-This event is handled for every cell when the grid has drawn the specified cell. You can handle the CellDrawn event and use its Graphics argument to do custom drawing after the grid has finished drawing the cell.

Following code example illustrates handling the Draw Cell event.

### 1. Using C#

[C#]

```
// DrawCell event is used to apply styles to the grid.
private void gridControl1_DrawCell(object sender, GridDrawCellEventArgs e)
{
    if (e.RowIndex == 0)
    {
        e.Style.Interior = new BrushInfo(GradientStyle.Vertical,
Color.FromArgb(255, 229, 201), Color.FromArgb(255, 153, 52));
    }
    else if (e.ColIndex == 0)
    {
        e.Style.Interior = new BrushInfo(GradientStyle.Horizontal,
Color.White, Color.FromArgb(102, 110, 152));
    }
    else if (e.RowIndex % 2 == 0)
    {
        e.Style.Interior = new
BrushInfo(GradientStyle.BackwardDiagonal, Color.FromArgb(51, 51, 101),
Color.White);
    }
}
```

### 2. Using VB.NET

[VB .NET]

```
' DrawCell event is used to apply styles to the grid.
Private Sub gridControl1_DrawCell(ByVal sender As Object, ByVal e As
GridDrawCellEventArgs)
    If e.RowIndex = 0 Then
        e.Style.Interior = New BrushInfo(GradientStyle.Vertical,
Color.FromArgb(255, 229, 201), Color.FromArgb(255, 153, 52))
    ElseIf e.ColIndex = 0 Then
        e.Style.Interior = New BrushInfo(GradientStyle.Horizontal,
Color.White, Color.FromArgb(102, 110, 152))
    ElseIf e.RowIndex Mod 2 = 0 Then
        e.Style.Interior = New BrushInfo(GradientStyle.BackwardDiagonal,
Color.FromArgb(51, 51, 101), Color.White)
    End If
```

```
End Sub
```

The above code identifies the 1st row, 1st column and the even rows using their index values, and paints them with unique interior styles.

Following code example illustrates handling the CellDrawn Event.

### 1. Using C#

```
[C#]
```

```
// Handling CellDrawn Event to customize the appearance of grid cells.
private void gridControl1_CellDrawn(object sender,
GridDrawCellEventArgs e)
{
    if(e.ColumnIndex==6 &&e.RowIndex>0)
    {
        Rectangle rec = e.Bounds, rect=e.Bounds;
        rec.X=(e.Bounds.Left+e.Bounds.Right)/2;
        if(e.Style.CellValue.ToString() == "1")
        {
            e.Graphics.FillEllipse(Brushes.Gray,rect);

GridImageCellRenderer.DrawImage(e.Graphics,this.imageList1,1,rec,false);
;
        }
        else
        {
            e.Graphics.FillEllipse(Brushes.LightGray,rect);

GridImageCellRenderer.DrawImage(e.Graphics,this.imageList1,0,rec,false)
;
        }
    }
}
```

### 2. Using VB.NET

```
[VB .NET]
```

```
' Handling CellDrawn Event to customize the appearance of grid cells.
Private Sub gridControl1_CellDrawn(ByVal sender As Object, ByVal e As
GridDrawCellEventArgs)
    If e.ColumnIndex = 6 AndAlso e.RowIndex > 0 Then
        Dim rec As Rectangle = e.Bounds, rect As Rectangle = e.Bounds
```

```

rec.X = (e.Bounds.Left + e.Bounds.Right) / 2
If e.Style.CellValue.ToString() = "1" Then
    e.Graphics.FillEllipse(Brushes.Gray, rect)
    GridImageCellRenderer.DrawImage(e.Graphics, Me.imageList1, 1,
    rec, False)
Else
    e.Graphics.FillEllipse(Brushes.LightGray, rect)
    GridImageCellRenderer.DrawImage(e.Graphics, Me.imageList1, 0,
    rec, False)
End If
End If
End Sub

```

The above code identifies the cells of the 6th column except the cell corresponding to the column header, using their index values, and customizes their appearance.

#### 4.1.4.13.6 Formatting Drop-down List

Essential Grid has built-in support for displaying a Grid List control as a drop-down inside a grid cell. We can embed grid list controls into the grid cells and customize them.

The following screen shot shows a grid cell with Grid List control as its drop-down.

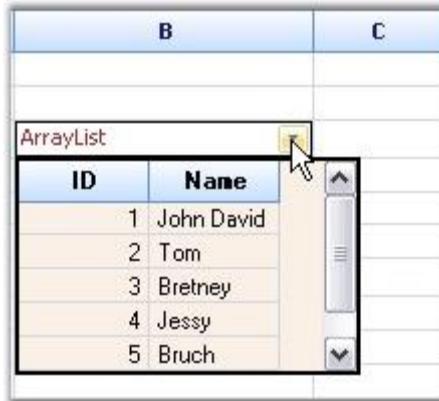


Figure 177: Grid List control embedded in a Grid Cell

Let us see how to add a Grid List control to a grid cell and bind some data.

To have a Grid List control in a Grid cell, set its **CellType** property as *GridListControl*. An array is used as data source in the following example. You can set its **DataSource** and **DisplayMember** properties as follows:

**[C#]**

```
ArrayList array = new ArrayList();
array.Add(new MyClass(001, "John David"));
array.Add(new MyClass(002, "Tom"));
array.Add(new MyClass(003, "Bretney"));
array.Add(new MyClass(004, "Jessy"));
array.Add(new MyClass(005, "Bruch"));
array.Add(new MyClass(006, "Johny"));

// Set up a Grid List control cell.
this.gridControl1[rowIndex, 2].CellType = "GridListControl";

// Specify the data source and display member for the Grid List
control.
this.gridControl1[rowIndex, 2].DataSource = array;
this.gridControl1[rowIndex, 2].DisplayMember = "Name";
```

**[VB.NET]**

```
Dim array As ArrayList = New ArrayList()
array.Add(New [MyClass](1, "John David"))
array.Add(New [MyClass](2, "Tom"))
array.Add(New [MyClass](3, "Bretney"))
array.Add(New [MyClass](4, "Jessy"))
array.Add(New [MyClass](5, "Bruch"))
array.Add(New [MyClass](6, "Johny"))

' Set up a Grid List control cell.
Me.gridControl1(rowIndex, 2).CellType = "GridListControl"

' Specify the data source and display member for the Grid List control.
Me.gridControl1(rowIndex, 2).DataSource = array
Me.gridControl1(rowIndex, 2).DisplayMember = "Name"
```

We have now added a Grid List control in a Grid cell and bound the data to it.

This Grid List control can be customized by accessing the *GridDropDownGridListControlCellRenderer* class, inside the *CurrentCellShowedDropDown* event handler.

 **Note:** CurrentCellShowedDropDown event is handled once the drop-down of the specified grid cell is made visible by clicking the downward arrow at the end of the cell.

The following code illustrates this event:

**[C#]**

```
private void gridControl1_CurrentCellShowedDropDown(object sender,
EventArgs e)
{
    // Retrieve the DropDownList Cell Renderer.
    GridDropDownGridListControlCellRenderer listRenderer =
(GridDropDownGridListControlCellRenderer)this.gridControl1.CellRenderers[
"GridListControl"];

    // Apply styles to Grid List control in the drop-down.
    listRenderer.ListControlPart.Grid.TableStyle.Font.Size = 17.8f;
    listRenderer.ListControlPart.BorderStyle = BorderStyle.FixedSingle;
    listRenderer.ListControlPart.Grid.BackColor = Color.FromArgb(250,
240, 230);
    listRenderer.ListControlPart.Grid.DefaultGridBorderStyle =
GridBorderStyle.Solid;
    listRenderer.ListControlPart.Grid.TableStyle.TextColor =
Color.MidnightBlue;
    listRenderer.ListControlPart.Grid.Properties.GridLineColor =
Color.FromArgb(208, 215, 229);
    listRenderer.ListControlPart.FillLastColumn = false;
}
```

**[VB.NET]**

```
Private Sub gridControl1_CurrentCellShowedDropDown(ByVal sender As
Object, ByVal e As EventArgs)

    ' Retrieve the DropDownList Cell Renderer.
    Dim listRenderer As GridDropDownGridListControlCellRenderer =
 CType(Me.gridControl1.CellRenderers("GridListControl"),
GridDropDownGridListControlCellRenderer)

    ' Apply styles to Grid List control in the drop-down.
    listRenderer.ListControlPart.Grid.TableStyle.Font.Size = 17.8F
    listRenderer.ListControlPart.BorderStyle = BorderStyle.FixedSingle
    listRenderer.ListControlPart.Grid.BackColor = Color.FromArgb(250,
240, 230)
    listRenderer.ListControlPart.Grid.DefaultGridBorderStyle =
GridBorderStyle.Solid
```

```
listRenderer.ListControlPart.Grid.TableStyle.TextColor =  
Color.MidnightBlue  
listRenderer.ListControlPart.Grid.Properties.GridLineColor =  
Color.FromArgb(208, 215, 229)  
listRenderer.ListControlPart.FillLastColumn = False  
End Sub
```

You can apply various styles such as table style, border style, text color, grid line color etc, to the Grid List control inside a grid cell as in Figure 1.

#### 4.1.4.14 Working with Rows and Columns

Grid control has properties that allow users to manipulate rows and columns programmatically. The following properties will be discussed in this section.

- **GridControl.Cols, GridControl.Rows** - Allows you to hide rows and columns, to freeze them to prevent scrolling and to control the number of headers.
- **GridControl.ColWidths, GridControl.RowHeights** - Allows you to set the row heights and column widths programmatically.
- **GridListControl.ColumnStyle, GridListControl.RowStyle** - Allows you to set the row or column styles.

For a Grid Data Bound Grid, you can access the first two items through the **GridDataBoundGrid.Model** property. The Grid Data Bound Grid does not use **RowStyles** or **ColStyles**. It uses the **GridBoundColumn.StyleInfo** object to set column styles with row styles not being directly supported. See the section on Grid Data Bound Grid for more information on how to set its styles.

This section comprises the following topics:

##### 4.1.4.14.1 Hiding Rows and Columns

The **GridControl.Cols.Hidden** collection will allow you to specify whether a column is hidden or not. You can index these properties directly as shown in the code below or you can use the **Hidden.SetRange** method to provide settings for a range of rows or columns.

[C#]

```
// Hide column 2.
```

```
this.gridControl1.Cols.Hidden[2] = true;  
  
// Hide row 3.  
this.gridControl1.Rows.Hidden[3] = true;
```

**[VB.NET]**

```
' Hide column 2.  
Me.GridControl1.Cols.Hidden(2) = True  
  
' Hide row 3.  
Me.GridControl1.Rows.Hidden(3) = True
```

You can also use this property to hide the default row headers and column headers. These headers are just column zero and row zero respectively. To hide them you can use code like the one given below.

**[C#]**

```
// Hide default row headers.  
this.gridControl1.Cols.Hidden[0] = true;  
  
// Hide default column headers.  
this.gridControl1.Rows.Hidden[0] = true;
```

**[VB.NET]**

```
' Hide default row headers.  
Me.GridControl1.Cols.Hidden(0) = True  
  
' Hide default column headers.  
Me.GridControl1.Rows.Hidden(0) = True
```

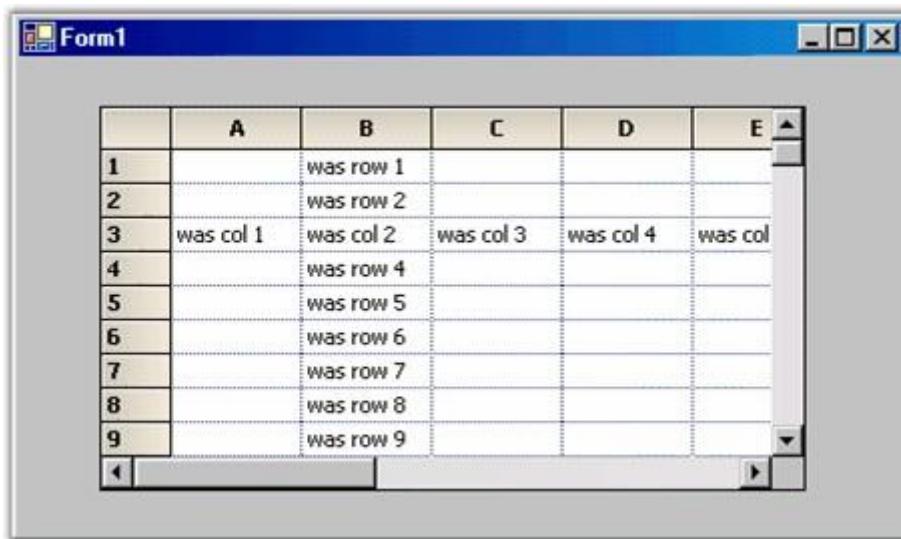


Figure 178: Default Grid With No Hidden Rows or Columns

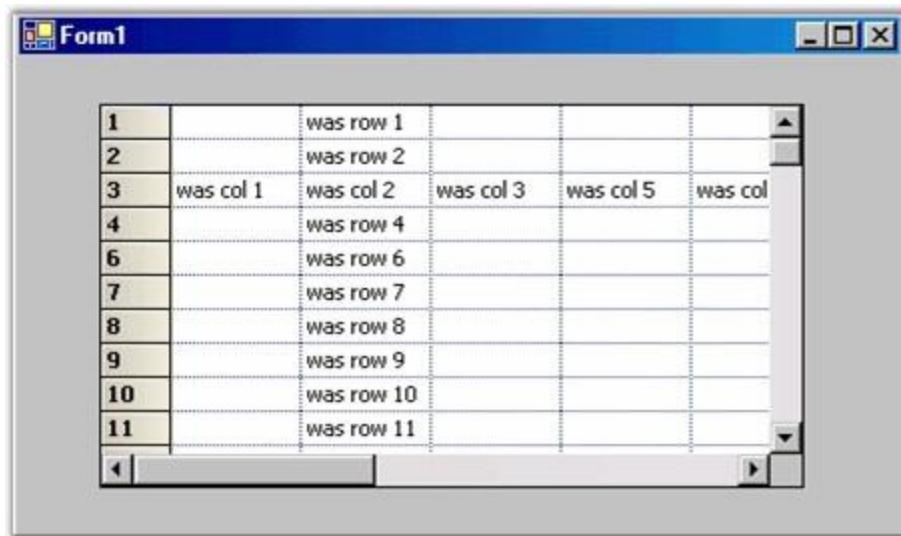


Figure 179: Grid with the Column Headers, Row 5 and Column 4 Hidden

#### 4.1.4.14.2 Header Rows and Columns

As we have seen in the previous section, it is possible to hide both the row and column headers. We can also have more than one header row and / or more than one header column. The properties that control the number of header rows and columns is **GridControl.Rows.HeaderCount** and **GridControl.Cols.HeaderCount**. This **HeaderCount** property is the index of the last header row or column. So, to have a total of three column header rows, set **Rows.HeaderCount** to two.

**[C#]**

```
// Total of three column header rows.
this.gridControl1.Rows.HeaderCount = 2;
```

**[VB .NET]**

```
' Total of three column header rows.
Me.GridControl1.Cols.Rows.HeaderCount = 2
```

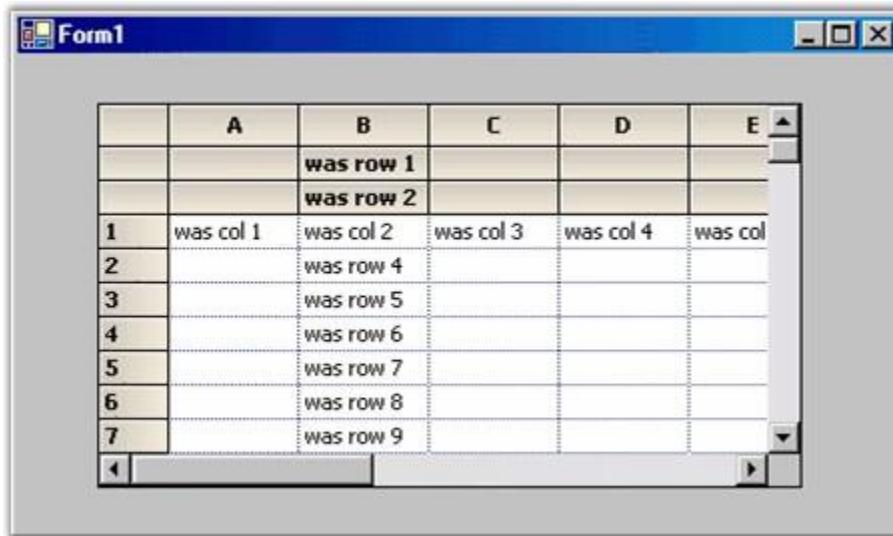


Figure 180: Grid With Three Column Header Rows

#### 4.1.4.14.3 Frozen Rows and Columns

A frozen row is one that cannot be scrolled. For example, the default column header (row 0) is a frozen row. Frozen rows will always be displayed at the top of the grid. You can set the number of frozen rows using the **GridControl.Rows.FrozenCount** property. In our previous code sample, we used the **Rows.HeaderCount** property to set up two additional column header rows. To cause the new headers to be fixed and not to scroll, you need to set the **Rows.FrozenCount** to two. Note that you can freeze non-header type rows as well but, in the following code samples, we are freezing headers only.

**[C#]**

```
// Have 3 non-scrollable rows at the top.
this.gridControl1.Rows.FrozenCount = 2;
```

```
// Total of three column header rows.  
this.gridControl1.Rows.HeaderCount = 2;
```

**[VB.NET]**

```
' Have 3 non-scrollable rows at the top.  
Me.GridControl1.Rows.FrozenCount = 2  
  
' Total of three column header rows.  
Me.GridControl1.Rows.HeaderCount = 2
```

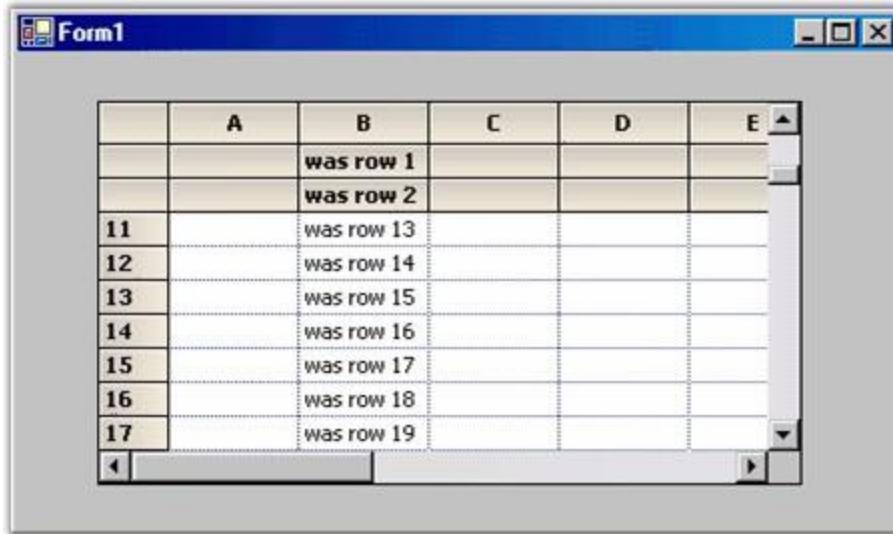


Figure 181: Grid with Three Frozen Column Header Rows

As we have said, frozen rows will always appear at the top of the grid and frozen columns will always appear to the left of the grid. It is possible to freeze an interior range of row or columns, using the **GridControl.Rows.FreezeRange** or **GridControl.Cols.FreezeRange** method. But, the **FreezeRange** method will move the requested rows / columns to the top or left and then it will set the **FrozenCount** to actually freeze the rows or columns.

**[C#]**

```
// Moves rows 3 and 4 to the top of the grid and freezes them.  
this.gridControl1.Rows.FreezeRange(3, 4);
```

**[VB.NET]**

```
' Moves rows 3 and 4 to the top of the grid and freezes them.  
Me.GridControl1.Rows.FreezeRange(3, 4)
```

#### 4.1.4.14.4 Moving Rows and Columns

The methods **GridControl.Rows.MoveRange** and **GridControl.Cols.MoveRange** are used to move rows and columns in a grid. The **MoveRange** method takes three parameters that are used to determine the start position, number of items to move and the target position.

[C#]

```
// Starting at row 7, move 2 rows to row 4.  
this.gridControl1.Rows.MoveRange(7, 2, 4);
```

[VB .NET]

```
' Starting at row 7, move 2 rows to row 4.  
Me.GridControl1.Rows.MoveRange(7, 2, 4)
```

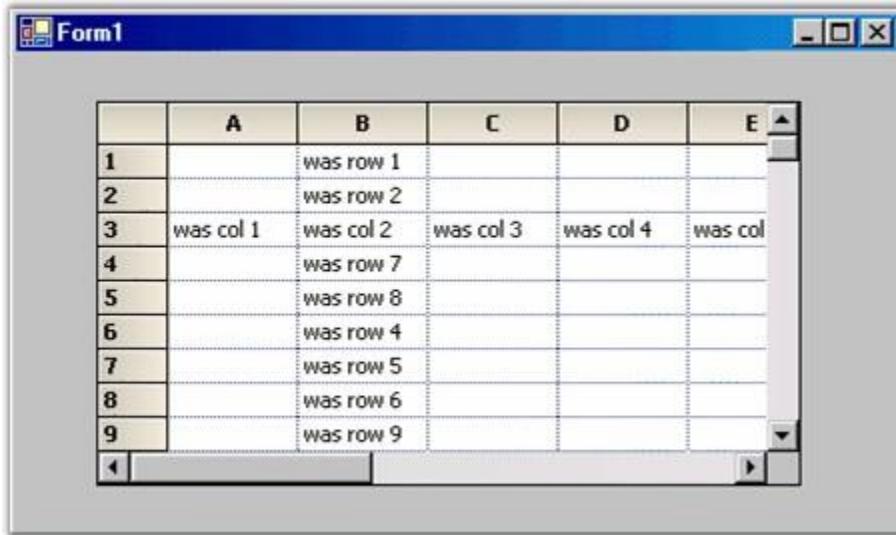


Figure 182: Grid After Moving Rows 7 and 8 to Row 4

#### 4.1.4.14.5 Setting Column Widths and Row Heights

The **GridControl.ColWidths** and **GridControl.RowHeights** collections will allow you to programmatically set the width of a column and / or the height of a row.



**Note:** Before you can use `GridDataBoundGrid.Model.ColWidths` to explicitly set column widths in a Grid Data Bound Grid, you must first set `GridDataBoundGrid.AllowResizeToFit` to false. Otherwise, the grid will try to size columns based on the width of the header text.

[C#]

```
// Set the width of column 3.  
this.gridControl1.ColWidths[3] = 40;  
  
// Set the height of row 4.  
this.gridControl1.RowHeights[4] = 40;
```

[VB .NET]

```
' Set the width of column 3.  
Me.GridControl1.ColWidths(3) = 40  
  
' Set the height of row 4.  
Me.GridControl1.RowHeights(4) = 40
```

	A	B	C	D	E
1		was row 1			
2		was row 2			
3	was col 1	was col 2	was col 3	was col 4	was col 5
4		was row 4			
5		was row 5			
6		was row 6			
7		was row 7			
8		was row 8			
▼					

Figure 183: Grid After Sizing Column 3 and Row 4

#### 4.1.4.14.6 Setting Column Styles and Row Styles

The **GridControl.ColStyles** and **GridControl.RowStyle** collections will allow you to programmatically set the default row or column style. This code will set the **backcolor** and the text color as well as set the font to bold for column two and row three.



**Note:** RowStyles and ColStyles are not supported in a Grid Data Bound Grid. For that grid, you will need to use the GridBoundColumn.StyleInfo property to set column styles and you will need to use the grid.Model.QueryCellInfo event to set row styles.

**[C#]**

```
// Set Back Color, Text Color and Font Style of Column 2.  
this.gridControl1.ColStyles[2].BackColor = Color.Red;  
this.gridControl1.ColStyles[2].TextColor = Color.White;  
this.gridControl1.ColStyles[2].Font.Bold = true;  
  
// Set Back Color, Text Color and Font Style of Row 3.  
this.gridControl1.RowStyle[3].BackColor = Color.Red;  
this.gridControl1.RowStyle[3].TextColor = Color.White;  
this.gridControl1.RowStyle[3].Font.Bold = true;
```

**[VB .NET]**

```
' Set Back Color, Text Color and Font Style of Column 2.  
Me.GridControl1.ColStyles(2).BackColor = Color.Red  
Me.GridControl1.ColStyles(2).TextColor = Color.White  
Me.GridControl1.ColStyles(2).Font.Bold = True  
  
' Set Back Color, Text Color and Font Style of Row 3.  
Me.GridControl1.RowStyle(3).BackColor = Color.Red  
Me.GridControl1.RowStyle(3).TextColor = Color.White  
Me.GridControl1.RowStyle(3).Font.Bold = True
```

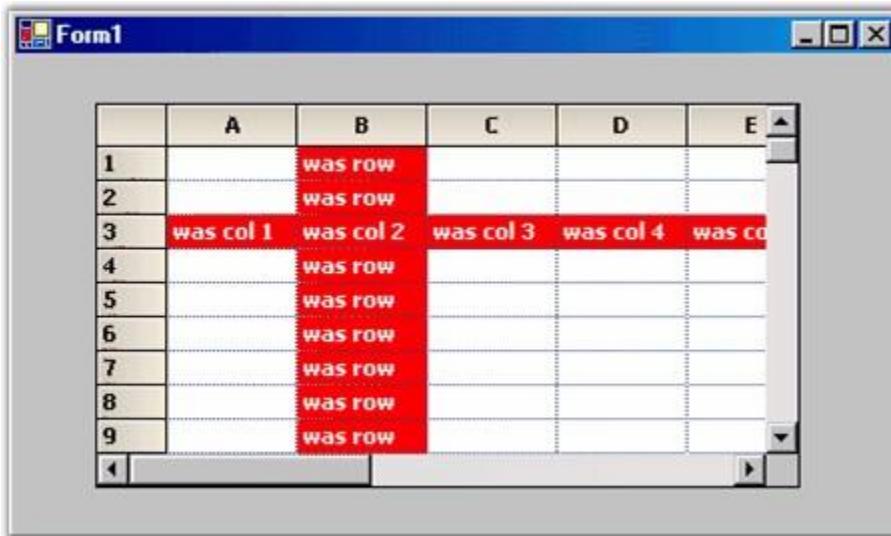


Figure 184: Grid After Setting the Styles For Column 2 and Row 3

#### 4.1.4.14.7 Controlling the Resize Behavior

Essential Grid supports the resizing behavior of columns and rows in the Grid control. This is achieved by using the **ResizeColsBehavior** and **ResizeRowsBehavior** properties.

The **GridResizeCellsBehavior** enumeration provides the following options to control the resizing behavior.

- **AllowDragOutside**-Allows the user to drag the cell boundary outside the grid client area and resize the specific row or column.



**Note:** *Grid client area is the area where the cells along with row and column headers are visible to the client. Dragging outside the client area means dragging beyond the boundary of the grid.*

- **InsideGrid**-Allows the user to resize rows or columns from anywhere inside the grid by dragging the divider between any two row or column headers.
- **None**-Turns off the mouse control over the resizing rows and columns.
- **OutlineBounds**-Highlights the original cell boundaries of resizing row or column.
- **OutlineHeaders**-Highlights the header boundaries when the user resizes the associated row or column.
- **ResizeAll**-Resizes all rows or columns automatically when the user resizes one row or column with the mouse. All rows and columns are resized to the same size as the current row/column being resized.
- **ResizeSingle**-Resizes the row or column being resized by the user using the mouse.



**Note:** Also you can control the mouse controller's behavior at run time while the user is performing the action by subscribing to the **ResizingColumns** and **ResizingRows** events.

The following code illustrates how to use this method in Grid control:

1. Using C#

[C#]

```
this.gridControl1.ResizeColsBehavior =  
    GridResizeCellsBehavior.InsideGrid;  
this.gridControl1.ResizeRowsBehavior =  
    GridResizeCellsBehavior.InsideGrid;
```

2. Using VB.NET

[VB .NET]

```
Me.gridControl1.ResizeColsBehavior = GridResizeCellsBehavior.InsideGrid  
Me.gridControl1.ResizeRowsBehavior = GridResizeCellsBehavior.InsideGrid
```

#### 4.1.4.14.8 Resize To Fit

Essential Grid supports this feature to enable resizing of columns and rows based on the content of cells. The **ResizeToFit** method is used for this purpose.

The following code illustrates how to use this method in Grid control:

1. Using C#

[C#]

```
// Resize the column widths.  
this.gridControl1.ColWidths.ResizeToFit(GridRangeInfo.Cols(1, 5));  
  
// Resize the row heights.  
this.gridControl1.RowHeights.ResizeToFit(GridRangeInfo.Rows(1, 5));
```

2. Using VB.NET

**[VB.NET]**

```
' Resize the column widths.
Me.gridControl1.ColWidths.ResizeToFit(GridRangeInfo.Cols(1, 5))

' Resize the row heights.
Me.gridControl1.RowHeights.ResizeToFit(GridRangeInfo.Rows(1, 5))
```



**Note:** The parameter passed to the `ResizeToFit` method is either `GridRangeInfo.Cols` or `GridInfo.Rows` method, which in turn has two parameters:

1. The first parameter corresponds to the starting row/column that is to be resized to fit.
2. The second parameter corresponds to the ending row/column upto which the resize has to be done.

The following image shows the application of resize to fit operation to the first five rows of the grid.

	A	B	C	D	E	F	G	H	I	J	K	
1	row0 col0	row0 col1	row0 col2	row0 col3	row0 col4	row0 col5	row0 col6	row0 col7	row0 col8	row0 col9	row0 col10	row0 col11
2	row1 col0	row1 col1	row1 col2	row1 col3	row1 col4	row1 col5	row1 col6	row1 col7	row1 col8	row1 col9	row1 col10	row1 col11
3	row2 col0	row2 col1	row2 col2	row2 col3	row2 col4	row2 col5	row2 col6	row2 col7	row2 col8	row2 col9	row2 col10	row2 col11
4	row3 col0	row3 col1	row3 col2	row3 col3	row3 col4	row3 col5	row3 col6	row3 col7	row3 col8	row3 col9	row3 col10	row3 col11
5	row4 col0	row4 col1	row4 col2	row4 col3	row4 col4	row4 col5	row4 col6	row4 col7	row4 col8	row4 col9	row4 col10	row4 col11
6	row5 col0	row5 col1	row5 col2	row5 col3	row5 col4	row5 col5	row5 col6	row5 col7	row5 col8	row5 col9	row5 col10	row5 col11
7	row6 col0	row6 col1	row6 col2	row6 col3	row6 col4	row6 col5	row6 col6	row6 col7	row6 col8	row6 col9	row6 col10	row6 col11
8	row7 col0	row7 col1	row7 col2	row7 col3	row7 col4	row7 col5	row7 col6	row7 col7	row7 col8	row7 col9	row7 col10	row7 col11
9	row8 col0	row8 col1	row8 col2	row8 col3	row8 col4	row8 col5	row8 col6	row8 col7	row8 col8	row8 col9	row8 col10	row8 col11
10	row9 col0	row9 col1	row9 col2	row9 col3	row9 col4	row9 col5	row9 col6	row9 col7	row9 col8	row9 col9	row9 col10	row9 col11
11	row10 col0	row10 col1	row10 col2	row10 col3	row10 col4	row10 col5	row10 col6	row10 col7	row10 col8	row10 col9	row10 col10	row10 col11
12	row11 col0	row11 col1	row11 col2	row11 col3	row11 col4	row11 col5	row11 col6	row11 col7	row11 col8	row11 col9	row11 col10	row11 col11
13	row12 col0	row12 col1	row12 col2	row12 col3	row12 col4	row12 col5	row12 col6	row12 col7	row12 col8	row12 col9	row12 col10	row12 col11
14	row13 col0	row13 col1	row13 col2	row13 col3	row13 col4	row13 col5	row13 col6	row13 col7	row13 col8	row13 col9	row13 col10	row13 col11
15	row14 col0	row14 col1	row14 col2	row14 col3	row14 col4	row14 col5	row14 col6	row14 col7	row14 col8	row14 col9	row14 col10	row14 col11
16	row15 col0	row15 col1	row15 col2	row15 col3	row15 col4	row15 col5	row15 col6	row15 col7	row15 col8	row15 col9	row15 col10	row15 col11
17	row16 col0	row16 col1	row16 col2	row16 col3	row16 col4	row16 col5	row16 col6	row16 col7	row16 col8	row16 col9	row16 col10	row16 col11
18	row17 col0	row17 col1	row17 col2	row17 col3	row17 col4	row17 col5	row17 col6	row17 col7	row17 col8	row17 col9	row17 col10	row17 col11
19	row18 col0	row18 col1	row18 col2	row18 col3	row18 col4	row18 col5	row18 col6	row18 col7	row18 col8	row18 col9	row18 col10	row18 col11
20	row19 col0	row19 col1	row19 col2	row19 col3	row19 col4	row19 col5	row19 col6	row19 col7	row19 col8	row19 col9	row19 col10	row19 col11
21	row20 col0	row20 col1	row20 col2	row20 col3	row20 col4	row20 col5	row20 col6	row20 col7	row20 col8	row20 col9	row20 col10	row20 col11
22	row21 col0	row21 col1	row21 col2	row21 col3	row21 col4	row21 col5	row21 col6	row21 col7	row21 col8	row21 col9	row21 col10	row21 col11
23	row22 col0	row22 col1	row22 col2	row22 col3	row22 col4	row22 col5	row22 col6	row22 col7	row22 col8	row22 col9	row22 col10	row22 col11
24	row23 col0	row23 col1	row23 col2	row23 col3	row23 col4	row23 col5	row23 col6	row23 col7	row23 col8	row23 col9	row23 col10	row23 col11
25	row24 col0	row24 col1	row24 col2	row24 col3	row24 col4	row24 col5	row24 col6	row24 col7	row24 col8	row24 col9	row24 col10	row24 col11

**Set RowHeight    Set Column Width    ColWidths - ResizeToFit    RowHeights - ResizeToFit**

Figure 185: Resize to Fit



**Note:** The preceding image is the output of a demo that is available in the samples in the following installed location.

<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Windows\Samples\2.0\Grid Layout\Resize To Fit Demo

The two buttons **Set RowHeight** and **Set Column Width** seen in the image above are used to set irregular height and width to the specified rows and columns of the grid respectively. The **ColWidths – Resize To Fit** and **RowHeights – Resize To Fit** are enabled only when the rows or columns are set to irregular height and width by using the Set RowHeight and Set Column Width buttons respectively.

#### 4.1.4.14.9 ResizeToFitOptimized

Essential Grid supports a ResizeToFitOptimized feature to enable resizing of columns and rows based on the contents of grid cells. The existing **ResizeToFit** method doesn't resize the columns or rows to make the entire cell value visible in the control. The **ResizeToFitOptimized** method is used for this purpose.

#### Use Case Scenarios

This feature enables you to display the entire cell with resized columns and rows even if the grid cells has special characters such as tab, newline, etc.

#### Methods Table

Table 7: Method Table

Method	Description	Parameters	Return Type
ResizeToFitOptimized	Resizes a range of rows or columns to optimally fit contents of the specified range of cells.	Overloads: <ul style="list-style-type: none"> <li>1) ResizeToFitOptimized(GridRangeInfo range)</li> <li>2) ResizeToFitOptimized(GridRangeInfo range, GridResizeToFitOptions option)</li> <li>3) ResizeToFitOptimized(GridRangeInfo range, GridTextOptions textOption)</li> </ul>	Void

### Applying **ResizeToFitOptimized** to an Application

The following code example illustrates how to use this **ResizeToFitOptimized** method in Grid control.

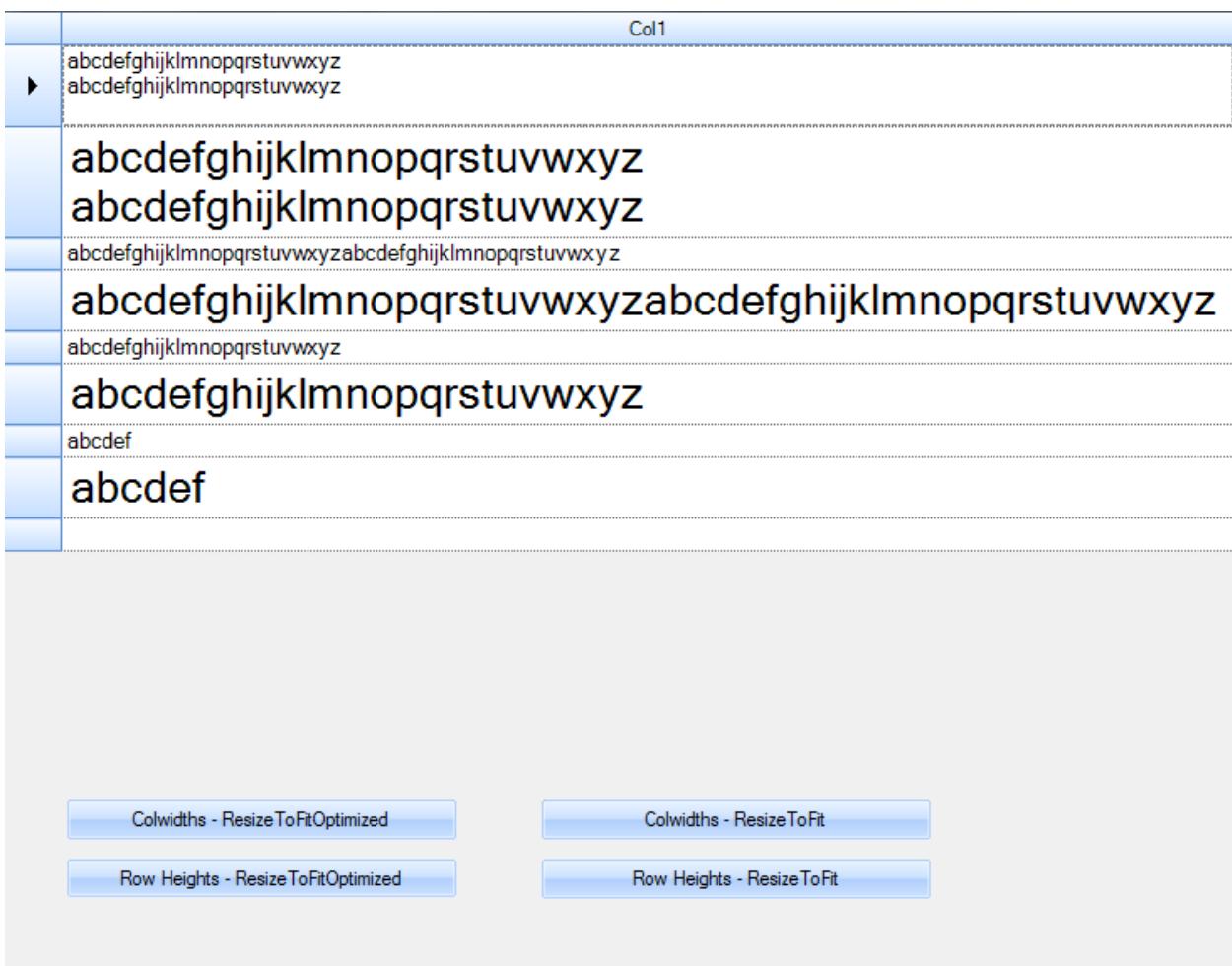
[C#]

```
// Resize the column width  
this.gridControl1.ColWidths.ResizeToFitOptimized(GridRangeInfo.Col(1));  
  
// Resize the row height  
this.gridControl1.RowHeights.ResizeToFitOptimized(GridRangeInfo.Rows(1, 8));
```

[VB .NET]

```
' Resize the column width  
Me.gridControl1.ColWidths.ResizeToFitOptimized(GridRangeInfo.Col(1));  
  
' Resize the row height  
Me.gridControl1.RowHeights.ResizeToFitOptimized(GridRangeInfo.Rows(1, 8))
```

The following image shows the application of **ResizeToFitOptimized** to the first column of the grid.

Figure 186: Applying `ResizeToFitOptimized` to first column

#### 4.1.4.14.10 ResizeToFit Behavior in AutoSize

Essential Grid supports resizing columns and rows based on the content of cells. This is achieved by using the `ResizeToFit()` method.

**AutoSize** enables the cell height to be automatically increased when the edited text does not fit into the cell and when **WrapText** is set to true. If **WrapText** is set to false, **AutoSize** will affect the column width, but it does not have the functionality of resizing the rows or columns after text has been entered as the `ResizeToFit()` method does.

AutoSize also supports resizing rows and columns based on their cell content during the binding of the data source to the grid.

Content can be entered in the grid and then the AutoSize property can be applied. This will resize the altered rows and columns.

```
[C#]
```

```
this.gridGroupingControl1.Appearance.AnyCell.AutoSize = true;
```

**[VB]**

```
me.gridGroupingControll.Appearance.AnyCell.AutoSize = true;
```

AutoSize can also be applied to different cell types to resize the rows and columns to fit the contents of the cell, and the row height and column width will be resized when editing the cell contents.

#### 4.1.4.14.11 Autosizing Custom Cell

Essential Grid supports automatic resizing of cells in the Grid control when custom controls are placed inside the cells.

The Grid lets you add custom controls to cells by creating a **CellModel** class and a **CellRenderer** class. These custom controls can have different sizes. When these controls are placed in the Grid, the corresponding cell is automatically resized to fit the controls. This is achieved by overriding the **OnQueryPreferredClientSize** method in the model class. The proper size of the control can be returned by using this method. The **ResizeToFit** method will then resize the cell to the size returned by the OnQueryPreferredClientSize method.

The following code examples illustrates how to implement this feature in the Grid control:

##### 1. Using C#

**[C#]**

```
// Override this method to calculate proper control size and return the
// same.
protected override Size OnQueryPreferredClientSize(Graphics g, int
rowIndex, int colIndex, GridStyleInfo style, GridQueryBounds
queryBounds)
{
    if(Grid[rowIndex,colIndex].Tag == null)
        throw new Exception("No User Control is tagged");
    else
    {
        // Get the type of the control from Style.Tag.
        Control userControl = Grid[rowIndex,colIndex].Tag as Control;

        // Calculate the size of the control.
        Size size = userControl.Size;
        size.Height += 2;

        // Return the size.
    }
}
```

```
        return size;
    }
}
```

## 2. Using VB.NET

### [VB .NET]

```
' Override this method to calculate proper control size and return the
same.
Protected Overrides Function OnQueryPreferredClientSize(ByVal g As
Graphics, ByVal rowIndex As Integer, ByVal colIndex As Integer, ByVal
style As GridStyleInfo, ByVal queryBounds As GridQueryBounds) As Size
    If Grid(rowIndex, colIndex).Tag Is Nothing Then
        Throw New Exception("No User Control is tagged")
    Else

        ' Get the type of the control from Style.Tag.
        Dim userControl As Control = TryCast(Grid(rowIndex,
colIndex).Tag, Control)

        ' Calculate the size of the control.
        Dim size As Size = userControl.Size
        size.Height += 2

        ' Return the size.
        Return size
    End If
End Function
```

The following image shows how the cell resizes itself automatically to the size of the control, when a custom control is added to it.

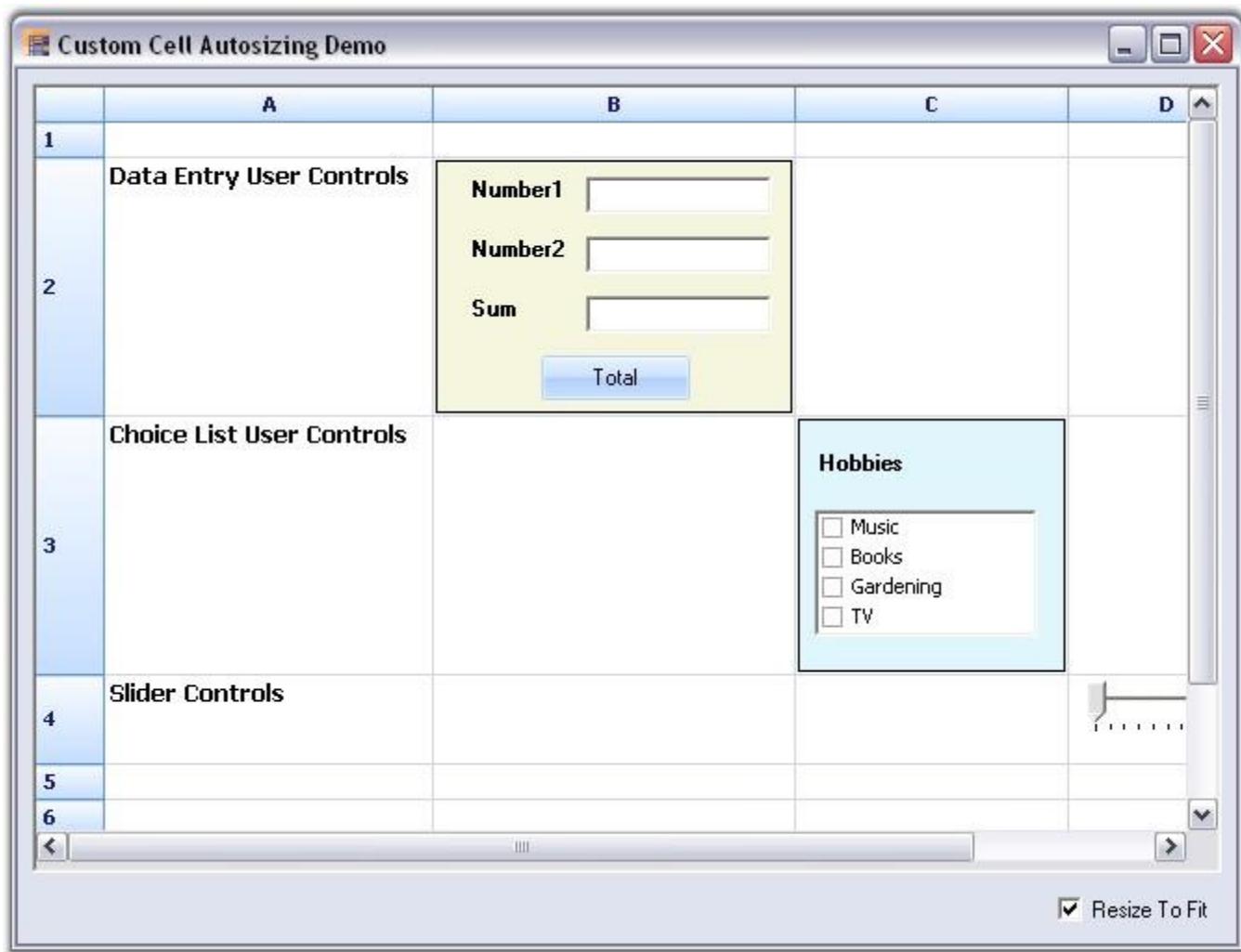


Figure 187: Autosizing While Adding Custom Controls

#### 4.1.4.14.12 Enter Key Behavior

This feature enables you to apply Enter key behavior for the following Windows Forms Grid controls: Grid, GridGrouping, and GridDataBoundGrid. By default, the Enter key behavior is set to move to the cell to the right. Using the **EnterKeyBehavior** property, you can set how the cell selection navigates when Enter is pressed.

The **EnterKeyBehavior** property provides support for the following navigation options to move the cell selection when Enter is pressed.

- Bottom
- BottomRight
- Down
- Left
- MostLeft
- MostRight

- None
- PageDown
- PageUp
- Right
- Top
- TopLeft
- Up

### Use Case Scenarios

You can use this property to change the navigation direction of Enter key behavior in the grid. The **EnterKeyBehavior** property works based on **WrapCellBehavior**. Enter key behavior navigates to the first column in the next row when at the end of a row and moving to the right.

### Properties

Property	Description	Type	Data Type
EnterKeyBehavior	Navigate to other cells when Enter is pressed.	GridDirectionType	Enum
WrapCellBehavior	Go to first column in next row or last column in previous row when at end or beginning of a row and moving based on Enter key behavior.	GridWrapCellBehavior	Enum

### Example

The following code illustrates how to set the **EnterKeyBehavior** property for Syncfusion Windows Forms Grid controls.

#### For the Grid Control

##### [C#]

```
this.gridControl1.EnterKeyBehavior = GridDirectionType.Top;
this.gridControl1.Model.Options.WrapCellBehavior =
GridWrapCellBehavior.WrapGrid;
```

##### [VB .NET]

```
Me.gridControl1.EnterKeyBehavior = GridDirectionType.Top
Me.gridControl1.Model.Options.WrapCellBehavior=GridWrapCellBehavior.WrapGrid;
```

### For GridGroupingControl

[C#]

```
this.gridGroupingControl1.TableModel.Options.EnterKeyBehavior =  
GridDirectionType.Top;  
  
this.gridGroupingControl1.TableModel.Options.WrapCellBehavior =  
GridWrapCellBehavior.WrapGrid;
```

[VB.NET]

```
Me.gridGroupingControl1.TableModel.Options.EnterKeyBehavior =  
GridDirectionType.Top  
  
Me.gridGroupingControl1.TableModel.Options.WrapCellBehavior =  
GridWrapCellBehavior.WrapGrid
```

### For GridDataBoundGrid

[C#]

```
this.gridDataBoundGrid1.Model.Options.EnterKeyBehavior =  
GridDirectionType.Top;  
  
this.gridDataBoundGrid1.Model.Options.WrapCellBehavior =  
GridWrapCellBehavior.WrapGrid;
```

[VB.NET]

```
Me.gridDataBoundGrid1.Model.Options.EnterKeyBehavior =  
GridDirectionType.Top  
  
Me.gridDataBoundGrid1.Model.Options.WrapCellBehavior =  
GridWrapCellBehavior.WrapGrid
```

### Sample Link

Samples for this feature are available in the following location:

**..\\..\AppData\Local\Syncfusion\EssentialStudio\10.4.0.53\Windows\Grid.Grouping.Windows  
\Samples\2.0\Styling and Formatting\Cell Formatting Demo**

#### 4.1.4.15 Covered Cells

Covered cells will allow you to group a range of cells together so that the group appears to be a single cell. For example, you might want a header to span two columns; this can be done using covered cells. Covered cells also support a large button which, occupies several cells.

To specify a covered range, you will have to use the **GridControl.CoveredRange** collection. You must add a **GridRangeInfo** object to this collection to specify the range of cells that you want covered. You can then use the top-left corner to control what is being displayed in this large cell.

##### [C#]

```
// Cover (2,2) through (3,4) so it is treated as a single cell.
this.gridControl1.Model.CoveredRanges.Add(GridRangeInfo.Cells(2, 2, 3,
4));
this.gridControl1[2, 2].CellType = "PushButton";
this.gridControl1[2, 2].CellAppearance = GridCellAppearance.Raised;
this.gridControl1[2, 2].Description = "Big Button";
this.gridControl1[2, 2].Font.Size = 12;
this.gridControl1[2, 2].Font.Bold = true;

// Cover (6,2) through (7,4) so it is treated as a large fancy label.
this.gridControl1.Model.CoveredRanges.Add(GridRangeInfo.Cells(6, 2, 7,
4));
this.gridControl1[6, 2].CellType = "Static";
this.gridControl1[6, 2].Text = "Big Label";
this.gridControl1[6, 2].HorizontalAlignment =
GridHorizontalAlignment.Center;
this.gridControl1[6, 2].VerticalAlignment =
GridVerticalAlignment.Middle;
this.gridControl1[6, 2].Font.Size = 12;
this.gridControl1[6, 2].Font.Bold = true;
this.gridControl1[6, 2].Interior = new
BrushInfo(GradientStyle.PathRectangle, Color.FromArgb(0xED, 0xF0,
0xF6), Color.FromArgb(0x2A, 0x43, 0x7E));
this.gridControl1[6, 2].TextColor = Color.FromArgb(0x66, 0x6E, 0x98);
```

##### [VB.NET]

```
' Cover (2,2) through (3,4) so it is treated as a single cell.
Me.gridControl1.Model.CoveredRanges.Add(GridRangeInfo.Cells(2, 2, 3,
4))
Me.gridControl1(2, 2).CellType = "PushButton"
Me.gridControl1(2, 2).CellAppearance = GridCellAppearance.Raised
Me.gridControl1(2, 2).Description = "Big Button"
Me.gridControl1(2, 2).Font.Size = 12
```

```
Me.gridControl1(2, 2).Font.Bold = True

' Cover (6,2) through (7,4) so it is treated as a large fancy label.
Me.gridControl1.Model.CoveredRanges.Add(GridRangeInfo.Cells(6, 2, 7,
4))
Me.gridControl1(6, 2).CellType = "Static"
Me.gridControl1(6, 2).Text = "Big Label"
Me.gridControl1(6, 2).HorizontalAlignment =
GridHorizontalAlignment.Center
Me.gridControl1(6, 2).VerticalAlignment = GridVerticalAlignment.Middle
Me.gridControl1(6, 2).Font.Size = 12
Me.gridControl1(6, 2).Font.Bold = True
Me.GridControl1(6, 2).Interior = New
BrushInfo(GradientStyle.PathRectangle, _Color.FromArgb(0xED, 0xF0,
0xF6),
_Color.FromArgb(0x2A, 0x43, 0x7E ))
Me.GridControl1(6, 2).TextColor = Color.FromArgb(0x66, 0x6E, 0x98)
```

#### 4.1.4.16 Deriving GridPrintDocument

The **GridPrintDocument** has the events **BeginPrint**, **PrintPage** and **EndPrint** which, are inherited from **PrintDocument** which, allows you access to the printing flow at certain points. To gain more control, you can derive the **GridPrintDocument**, and override members like **OnPrint**. This will allow you to access grid members like **ViewLayout** and **TopRowIndex** to obtain information about the page that is being printed. The following code will show you how to print the top and bottom row of the page.

[C#]

```
public class MyPrintDocument : GridPrintDocument
{
    GridControlBase _grid;

    public MyPrintDocument(GridControlBase grid, bool
printPreview):base(grid, printPreview)
    {
        _grid = grid;
    }

    protected override void
OnPrintPage(System.Drawing.Printing.PrintPageEventArgs ev)
    {
        base.OnPrintPage(ev);
```

```
_grid.PrintingMode = true;

// Get Top Row Index.
int topRow = _grid.TopRowIndex;
_grid.ViewLayout.Reset();

// Get Bottom Row Index.
int botRow = this._grid.ViewLayout.LastVisibleRow
            - (this._grid.ViewLayout.HasPartialVisibleRows
? 1 : 0);

_grid.PrintingMode = false;

// Print
Console.WriteLine("OnPrintPage " + topRow.ToString() + "    " +
botRow.ToString());
}
```

**[VB.NET]**

```
Public Class MyPrintDocument
Inherits GridPrintDocument
    Private _grid As GridControlBase
    Public Sub New(grid As GridControlBase, printPreview As Boolean)
        MyBase.New(grid, printPreview)
        _grid = grid
    End Sub

    Protected Overrides Sub OnPrintPage(ByVal ev As
System.Drawing.Printing.PrintPageEventArgs)
        MyBase.OnPrintPage(ev)
        _grid.PrintingMode = True

        ' Get Top Row Index.
        Dim topRow As Integer = _grid.TopRowIndex
        _grid.ViewLayout.Reset()

        ' Get Bottom Row Index.
        Dim botRow As Integer = Me._grid.ViewLayout.LastVisibleRow
        If Me._grid.ViewLayout.HasPartialVisibleRows Then
            botRow = botRow - 1
        End If
        _grid.PrintingMode = False

        // Print
    End Sub
End Class
```

```

Console.WriteLine(("OnPrintPage " + topRow.ToString() + "    " +
botRow.ToString()))

' OnPrintPage
End Sub

' MyPrintDocument
End Class

```

#### 4.1.4.17 Floating Cells

Floating cells are those cells whose content floats over empty, adjacent cells. You can enable floating cells at the grid level by setting the **GridControl.FloatCellsMode**.

Setting this property to the **GridFloatCellsMode.BeforeDisplayCalculation** will force the floating cells to always be calculated just prior to being displayed. Setting the property to the **GridFloatCellsMode.OnDemandCalculation** will calculate the floating cells only if the cell contents or size changes. This latter option is more efficient.

You can control a cell whether or not it floats over adjacent cells through the **FloatCell** property in the cell's **GridStyleInfo** object. You can also prevent a cell from being flooded by using its **GridStyleInfo.FloodCell** property. In the code given below, all three lines (1, 3, 5) hold the same text in column one. But, the floating cells in lines three and five are stopped short; line three by an occupied cell and line five by a **FloodCell** false settings.

[C#]

```

// Enable Float Cells.
this.gridControl1.FloatCellsMode =
GridFloatCellsMode.OnDemandCalculation;

// Specify Cell Text.
this.gridControl1[1, 1].Text = "This is a text that floats over several
cells.";
this.gridControl1[3, 1].Text = "This is a text that floats over several
cells.";
this.gridControl1[5, 1].Text = "This is a text that floats over several
cells.";
this.gridControl1[3, 3].Text = "3.14159";

// Code to prevent cell(5,2) from being flooded.
this.gridControl1[5, 2].FloodCell = false;

```

[VB.NET]

```
' Enable Float Cells.  
Me.gridControl1.FloatCellsMode = GridFloatCellsMode.OnDemandCalculation  
  
' Specify Cell Text.  
Me.gridControl1(1, 1).Text = "This is a text that floats over several  
cells."  
Me.gridControl1(3, 1).Text = "This is a text that floats over several  
cells."  
Me.gridControl1(5, 1).Text = "This is a text that floats over several  
cells."  
Me.gridControl1(3, 3).Text = "3.14159"  
  
' Code to prevent cell(5,2) from being flooded.  
Me.gridControl1(5, 2).FloodCell = False  
Me.gridControl1(2, 2).Font.Bold = True
```

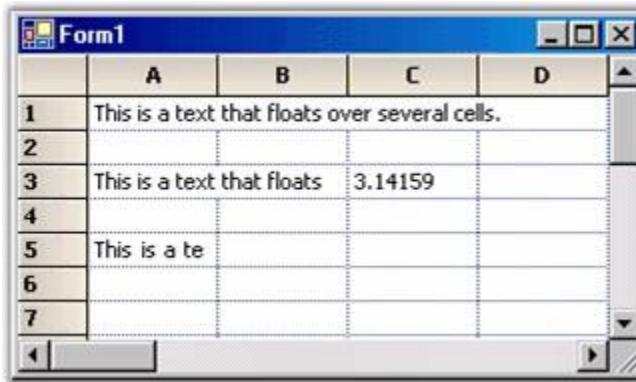


Figure 188: Floating Cells

#### 4.1.4.18 TabBarSplitterControl

TabBarSplitterControl enables users to create Tab Pages with dynamic splitters; when used with a grid control, it gives a workbook like appearance. It comes with Office 2007 Style, by default, and supports all the three color schemes.

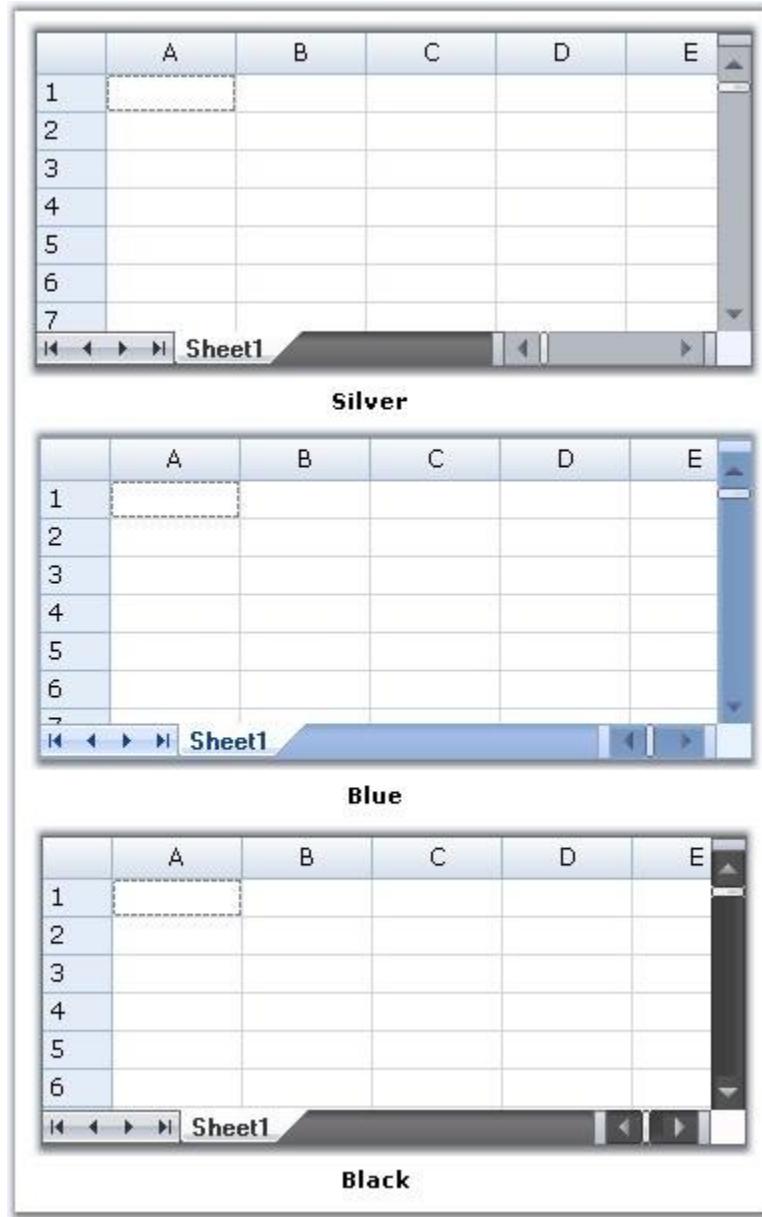
[C#]

```
this.tabBarSplitterControl.Style =  
Syncfusion.Windows.Forms.TabBarSplitterStyle.Office2007;
```

```
this.tabBarSplitterControl.Office2007ColorScheme =  
Syncfusion.Windows.Forms.Office2007Theme.Silver;
```

**[VB.NET]**

```
Me.tabBarSplitterControl.Style =  
Syncfusion.Windows.Forms.TabBarSplitterStyle.Office2007  
Me.tabBarSplitterControl.Office2007ColorScheme =  
Syncfusion.Windows.Forms.Office2007Theme.Silver
```



*Figure 189: Office 2007 Color Schemes*

## Custom Colors

We can apply custom colors to the TabBarSplitterControl by setting Office2007ColorScheme property to "Managed" and by giving the color through ApplyManagedColor method as follows.

### [C#]

```
this.tabBarSplitterControl.Office2007ColorScheme =  
Syncfusion.Windows.Forms.Office2007Theme.Managed;  
Office2007Colors.ApplyManagedColors(this, Color.PowderBlue);
```

### [VB .NET]

```
Me.tabBarSplitterControl.Office2007ColorScheme =  
Syncfusion.Windows.Forms.Office2007Theme.Managed  
Office2007Colors.ApplyManagedColors(Me, Color.PowderBlue)
```

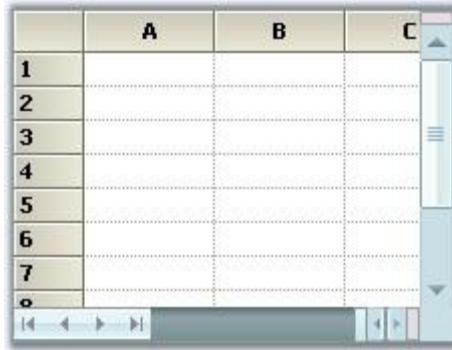


Figure 190: Custom Color of Tab Bar Splitter Control set to "PowderBlue"

### 4.1.4.19 PrepareViewStyleInfo Event

The **PrepareViewStyleInfo** event is raised to allow custom formatting of a cell by changing its style object just before it is drawn. This allows formatting based on the current view state, e.g. current cell context, focused control, and so on.

For example, if you want draw the current row with a bold text, you can use the **PrepareViewStyleInfo** to accomplish this task. The idea is to change the style to bold font for any cell in the current one. Given below are the steps that you can follow in order to implement this functionality.

- Add a handler for **CurrentCellDeactivated** to refresh the old row to remove its bold font.
- Add a handler for **CurrentCellActivated** to refresh the new current row to add its bold font.
- Add a handler for **PrepareViewStyleInfo** to conditionally change the bold style of the font if the requested cell is on the current row.

To see a full working sample, check the **HighlightCurrent** sample that ships with Essential Grid. Notice that the work is done just by making the grid refresh (redraw) a row. During this refresh, the **PrepareViewStyleInfo** is selected and the style is modified to be bold if the row is current. This means that no bold style information is saved anywhere. The **GridStyleInfo** object is just temporarily modified immediately before it is used in the drawing.

#### 4.1.4.20 Print Preview and Printing

Essential Grid directly supports printing and print previews through the .NET Framework classes **Systems.Windows.Forms.PrintPreviewDialog** and **Systems.Windows.Forms.PrintDialog**. A derived **PrintDocument**, **GridPrintDocument** is passed to these classes. This **GridPrintDocument** implements the printing logic that is needed to print multi-page grids.

Following code example illustrates how to enable print previewing.

##### 1. Using C#

[C#]

```
// PrintPreview button handler.  
private void PrintPreview_Click(object sender, System.EventArgs e)  
{  
    if (this.gridControl1 != null)  
    {  
        try  
        {  
            // Uses the default printer.  
            GridPrintDocument pd = new  
GridPrintDocument(this.gridControl1, true);  
            PrintPreviewDialog dlg = new PrintPreviewDialog() ;  
            dlg.Document = pd;  
            dlg.ShowDialog();  
        }  
        catch(Exception ex)  
        {  
            MessageBox.Show("An error occurred attempting to preview  
the grid - " + ex.Message);  
        }  
    }  
  
    private void Print_Click(object sender, System.EventArgs e)  
{  
        if (this.gridControl1 != null)  
        {  
            try  
            {  
                GridPrintDocument pd = new  
GridPrintDocument(this.gridControl1, true);  
                PrintDialog dlg = new PrintDialog() ;  
                dlg.Document = pd;  
                if( dlg.ShowDialog() == DialogResult.OK)  
                    pd.Print();  
            }  
            catch(Exception ex)  
            {  
                MessageBox.Show("An error occurred attempting to print the  
grid - " + ex.Message);  
            }  
        }  
    }  
}
```

## 2. Using VB.NET

**[VB.NET]**

```
' PrintPreview button handler.  
Private Sub PrintPreview_Click(ByVal sender As Object, ByVal e As EventArgs)  
    If (Not (gridControl1) Is Nothing) Then  
        Try  
            Dim pd As GridPrintDocument  
            pd = New GridPrintDocument(gridControl1, True)  
  
            'Uses the default printer.  
            Dim dlg As PrintPreviewDialog  
            dlg = New PrintPreviewDialog()  
            dlg.Document = pd  
            dlg.ShowDialog()  
  
            Catch ex As Exception  
                MessageBox.Show(("An error occurred attempting to preview  
the grid - " + ex.Message))  
            End Try  
        End If  
    End Sub  
  
' Print button handler.  
Private Sub Print_Click(ByVal sender As Object, ByVal e As EventArgs)  
    If (Not (gridControl1) Is Nothing) Then  
        Try  
            Dim pd As GridPrintDocument  
            pd = New GridPrintDocument(gridControl1, True)  
  
            ' Uses the default printer.  
            Dim dlg As PrintDialog  
            dlg = New PrintDialog()  
            dlg.Document = pd  
            If dlg.ShowDialog() = DialogResult.OK Then  
                pd.Print()  
            End If  
  
            Catch ex As Exception  
                MessageBox.Show(("An error occurred attempting to print  
the grid - " + ex.Message))  
            End Try  
        End If  
    End Sub
```

## Grid Helper Features

The following are the features of the **Grid Helper** that supports print preview and printing:

- Advanced printing
- Page layout
- Print Column To Fit

## Advanced Printing

Multiple grids can be printed across various pages using the helper class **GridPrintDocumentAdv**. This is achieved by drawing the full size grid to a large bitmap and then scaling this bitmap to fit the output page.

- The Print Preview can be enabled by using **GridPrintDocumentAdv** class or by clicking the Print Preview button under the **Grid Printing Options** in the UI.
- Columns can be specified to fit in a single page using the **ScaleColumnsToFitPage** property or selecting the **Scale Columns To Fit** check box on under the **Grid Printing Options** in the UI.
- Headers and footers can be added by using the **DrawGridPrintHeader** and **DrawGridPrintFooter** events or by selecting the **Show Header and Footer** check box under the **Grid Printing Options** in the UI.

Following code example illustrates Advanced Printing in Grid.

### 1. Using C#

[C#]

```
Syncfusion.GridHelperClasses.GridPrintDocumentAdv pd = new  
Syncfusion.GridHelperClasses.GridPrintDocumentAdv(this.gridControl1);  
pd.DefaultPageSettings.Margins = new  
System.Drawing.Printing.Margins(25, 25, 25, 25);  
pd.HeaderHeight = 70;  
pd.FooterHeight = 50;  
  
pd.ScaleColumnsToFitPage = true;  
  
PrintPreviewDialog previewDialog = new PrintPreviewDialog();  
previewDialog.Document = pd;  
previewDialog.Show();
```

### 2. Using VB.NET

[VB .NET]

```

Dim pd As Syncfusion.GridHelperClasses.GridPrintDocumentAdv = New
Syncfusion.GridHelperClasses.GridPrintDocumentAdv(Me.gridControl1)
pd.DefaultPageSettings.Margins = New
System.Drawing.Printing.Margins(25, 25, 25, 25)
pd.HeaderHeight = 70
pd.FooterHeight = 50

pd.ScaleColumnsToFitPage = True

Dim previewDialog As PrintPreviewDialog = New PrintPreviewDialog()
previewDialog.Document = pd
previewDialog.Show()

```

Following screen shot illustrates Advanced Printing functionality provided by the **GridPrintDocumentAdv** class.

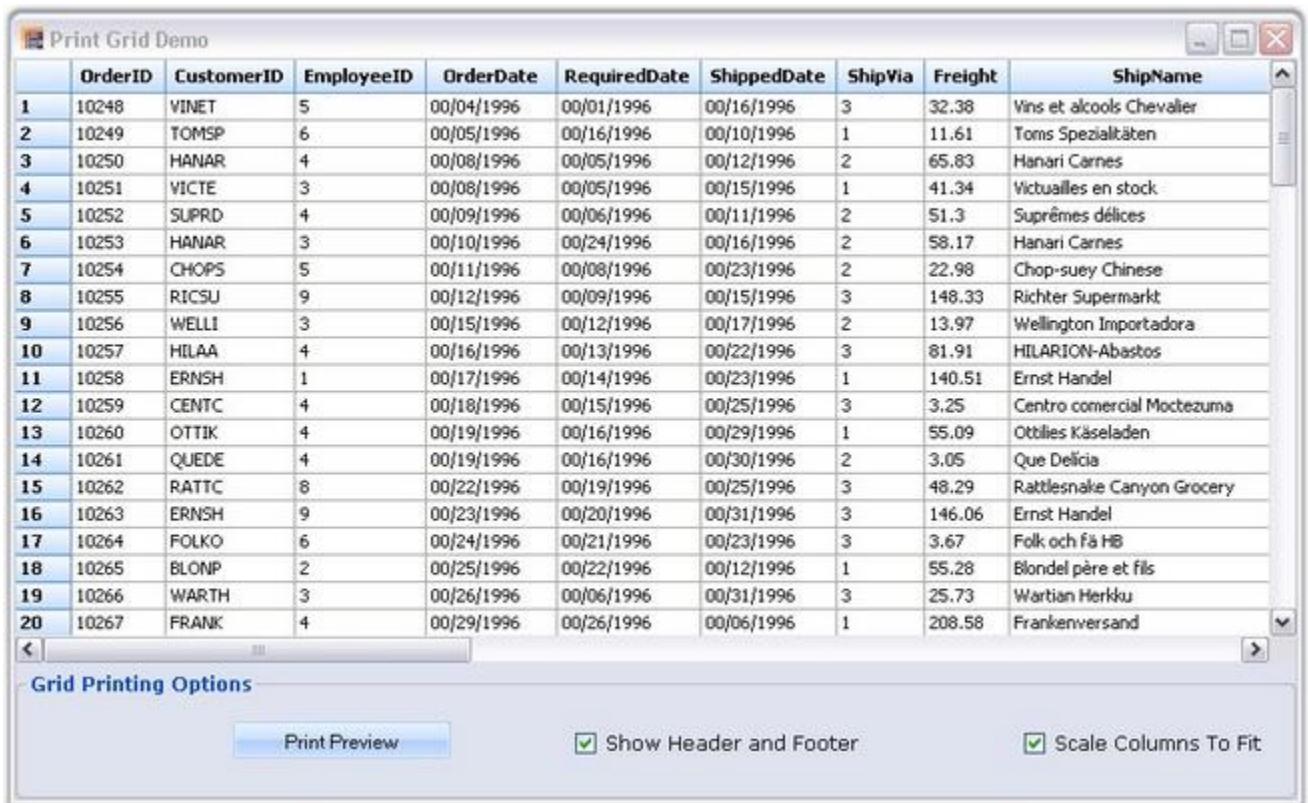


Figure 191: Print Grid

## Page Layout

The print Page Layout feature helps to view the printing layout for the grid by displaying a segment line and a page number with each segment. This helps users to analyze page breaks within the grid and manage them accordingly. Colors for the line and text of the page layout can be defined with the properties available. Following code example illustrates this.

1. Using C#

[C#]

```
LayoutSupportHelper layoutHelper;  
layoutHelper = new LayoutSupportHelper(gridControl1);  
layoutHelper.LineColor = Color.Blue;  
layoutHelper.TextColor = Color.Green;
```

2. Using VB.NET

[VB .NET]

```
Dim layoutHelper As LayoutSupportHelper  
layoutHelper = New LayoutSupportHelper(gridControl1)  
layoutHelper.LineColor = Color.Blue  
layoutHelper.TextColor = Color.Green
```

Following screen shot shows the page layout of the grid, with the segment line and page number.

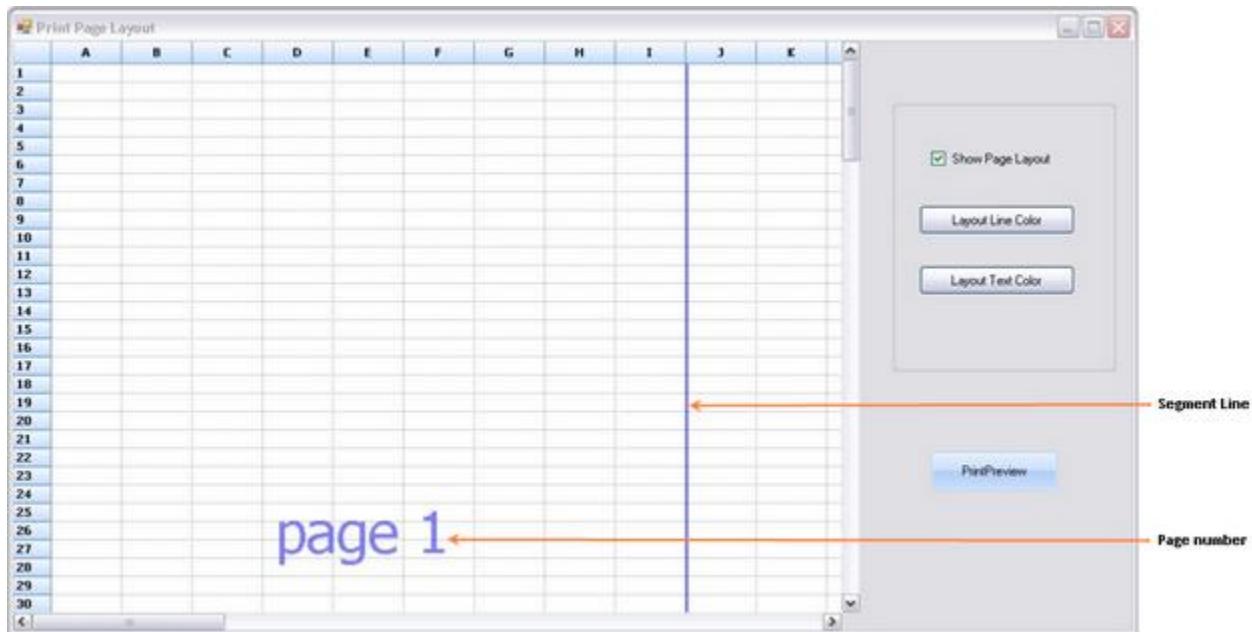


Figure 192: Print Page

 **Note:** The functionality mentioned above can also be achieved on UI by selecting Show Page Layout check box on the UI, which allows the user to view the page layout.

### Print To Fit

An entire grid can be printed on a single page by deriving **GridPrintDocument** class to handle the printing of entire grid on a single page. The class achieves this by drawing the full-size grid to a large bitmap and then scaling the same to fit the output page. Following code example illustrates this.

#### [C#]

```
GridPrintToFitDocument pd = new  
GridPrintToFitDocument(this.gridControl1, true);  
PrintDialog dlg = new PrintDialog();  
dlg.Document = pd;  
if (dlg.ShowDialog() == DialogResult.OK)  
pd.Print();
```

#### [VB .NET]

```
Dim dlg As PrintDialog = New PrintDialog()  
dlg.Document = pd  
If dlg.ShowDialog() = DialogResult.OK Then  
pd.Print()  
End If
```

Following screen shot illustrates the Grid's **Print To Fit** feature.

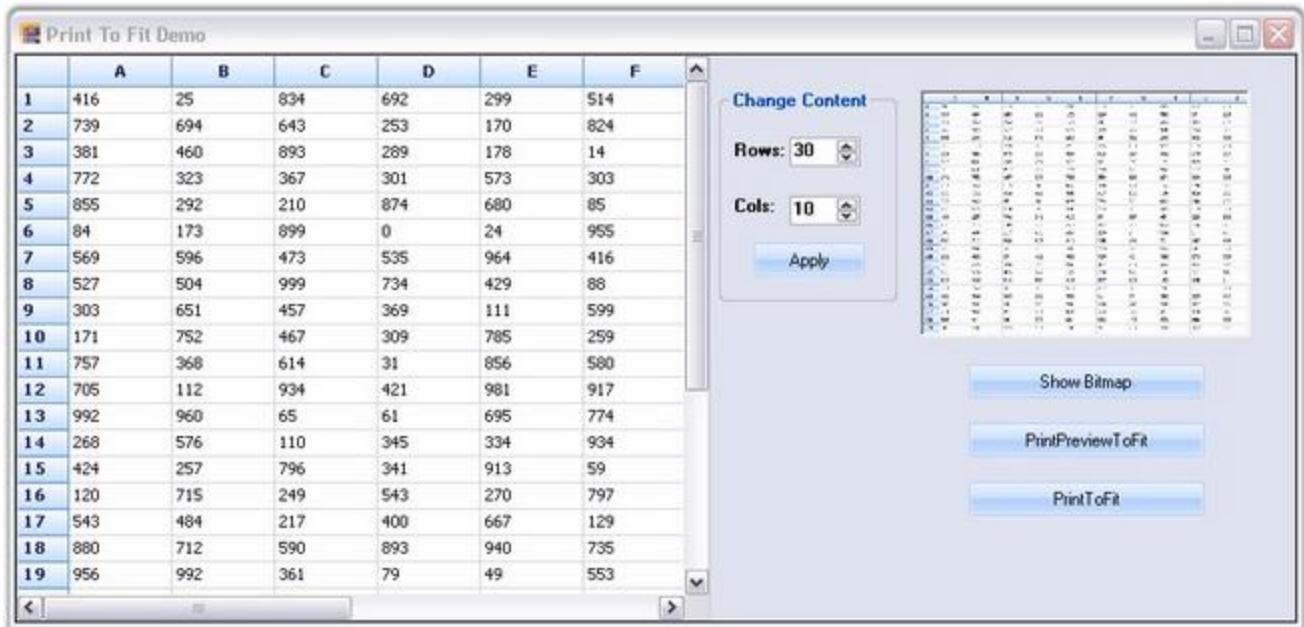


Figure 193: Print To Fit

This functionality can also be achieved by clicking the **PrintToFit** button on the UI. Refer Figure 3 on this page.

#### 4.1.4.21 Multiple Grid Printing

Multiple grids can be printed across various pages using the helper class **MultipleGridPrintDocument**. This is achieved by drawing the full-size grid to a large bitmap and then scaling this bitmap to fit the output page.

- **MultiGridPrinting** - Customizes the way printing support is provided for grids. It enables multiple grids to be printed in a single print.
- **PrintGridInNewPage** - Multiple grids can be printed continuously. However, the consecutive grid's starting page will begin on a new page.
- **DefaultGridPrint** - Multiple grids will be printed without considering the column breaks.
- **ScaleColumnsToFit** - Multiple grid columns will be scaled to fit the printed page.

#### Properties

Table 8: Properties Table

Property	Description	Type	Data Type	Reference links
----------	-------------	------	-----------	-----------------

GridPrintOptions	Used to customize the printing of a grid.	enum	enum	
------------------	---	------	------	--

GridPrintOptions Name	Description
MultiGridPrinting	Multiple grids are printed in a single print continuously one after another.
PrintGridInNewPage	Prints multiple grids in a new page.
DefaultGridPrint	Default grid printing without column breaks for each new page.
ScaleColumnsToFit	Multiple grid columns are scaled to fit the printed page.

### Sample Link

<Installed  
Location>\Syncfusion\EssentialStudio\<InstalledVersion>\Windows\Grid.Windows\Samples\2.0\Printing\Multi - GridPrinting

#### 4.1.4.21.1 Adding Multi-Grid Printing to an Application

- The Print Preview can be enabled by using the **MultipleGridPrintDocument** class or by clicking the Print Preview button under the Grid Printing Options in the UI.
- Headers and footers can be added by using the **DrawGridPrintHeader** and **DrawGridPrintFooter** events or by selecting the Show Header and Footer check box under the Grid Printing Options in the UI.

[C#]

```
List<Control> gridsToPrint = new List<Control>();
foreach (Control cd in this.Controls)
{
    if (cd is Control)
    {
        gridsToPrint.Add((Control)cd);
    }
}
MultiGridPrintDocument pd = new MultiGridPrintDocument(gridsToPrint);
pd.GridPrintOption =
MultiGridPrintDocument.GridPrintOptions.MultipleGridPrint;
pd.ShowHeaderFooterOnAllPages = true;
PrintPreviewDialog printDialog = new PrintPreviewDialog();
printDialog.Document = pd;
```

```
printDialog.ShowDialog();
```

**[VB]**

```
Dim ctrls As New List(Of Control)()
Dim gridsToPrint As New List(Of Control)()
For Each cd As Control In Me.Controls
    If TypeOf cd Is Control Then
        gridsToPrint.Add(CType(cd, Control))
    End If
Next cd
Dim pd As New MultiGridPrintDocument(gridsToPrint)
pd.GridPrintOption =
MultiGridPrintDocument.GridPrintOptions.MultipleGridPrint
pd.ShowHeaderFooterOnAllPages = True
Dim printDialog As New PrintPreviewDialog()
printDialog.Document = pd
printDialog.ShowDialog()
```

The following screen shot illustrates the advanced printing functionality provided by the **MultipleGridPrintDocument** class:

The screenshot shows a Windows application titled "Multi-Grid Printing". It displays two data grids side-by-side. The left grid is a "Orders" table with columns: OrderID, CustomerID, EmployeeID, OrderDate, RequiredDate, and ShippedDate. The right grid is a "Customers" table with columns: Contact Name and CompanyName. Both grids have scroll bars. Below the grids are "Grid Printing Options" and "TypeOfPrinting" settings.

	OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	ShippedDate
1	10248	VINET	5	07/04/1996	08/01/1996	07/16/1996
2	10249	TOMSP	6	07/05/1996	08/16/1996	07/10/1996
3	10250	HANAR	4	07/08/1996	08/05/1996	07/12/1996
4	10251	VICTE	3	07/08/1996	08/05/1996	07/15/1996
5	10252	SUPRD	4	07/09/1996	08/06/1996	07/11/1996
6	10253	HANAR	3	07/10/1996	07/24/1996	07/16/1996
7	10254	CHOPS	5	07/11/1996	08/08/1996	07/23/1996
8	10255	RICSU	9	07/12/1996	08/09/1996	07/15/1996
9	10256	WELLI	3	07/15/1996	08/12/1996	07/17/1996
10	10257	HILAA	4	07/16/1996	08/13/1996	07/22/1996
11	10258	ERNSH	1	07/17/1996	08/14/1996	07/23/1996
12	10259	CENTC	4	07/18/1996	08/15/1996	07/25/1996
13	10260	OTTIK	4	07/19/1996	08/16/1996	07/29/1996
14	10261	QUED	4	07/19/1996	08/16/1996	07/30/1996
15	10262	RATTC	8	07/22/1996	08/19/1996	07/25/1996

Contact Name	CompanyName
Maria Anders	Alfreds Futterkiste
Ana Trujillo	Ana Trujillo Emparedados y helados
Antonio Moreno	Antonio Moreno Taquería
Thomas Hardy	Around the Horn
Christina Berglund	Berglunds snabbköp
Hanna Moos	Blauer See Delikatessen
Frédérique Citeaux	Blondel père et fils
Martín Sommer	Bólido Comidas preparadas
Laurence Lebihan	Bon app'
Elizabeth Lincoln	Bottom-Dollar Markets
Victoria Ashworth	B's Beverages
Patricia Simpson	Cactus Comidas para llevar
Francisco Chang	Centro comercial Moctezuma
Yang Wang	Chop-suey Chinese
Pedro Afonso	Comércio Mineiro

**Grid Printing Options**

- MultiGridPrint
- PrintGridInNewPage
- DefaultGridPrint
- ScaleColumnToFit

**TypeOfPrinting**

- MultiGridPrint
- CustomizePrintPage

**Show Header and Footer**

**Print Preview**    **Print**

#### 4.1.4.22 Drag Column Header

In Grid control, a column header can be dragged to a new position by clicking on it, similar to how the fields in Microsoft Outlook are dragged without selecting the columns. This feature can be enabled in Grid control by adding the **DragColumnHeader** option under the **ControllerOptions** property. The event **QueryAllowDragColumnHeader** can be handled, while performing the drag operation.

The following code examples illustrate this feature.

##### 1. Using C#

```
[C#]

this.gridControl1.ControllerOptions |=
GridControllerOptions.DragColumnHeader;
void gridControl1_QueryAllowDragColumnHeader(object sender,
GridQueryDragColumnHeaderEventArgs e)
{
    if (e.Reason != GridQueryDragColumnHeaderReason.HitTest)
        System.Diagnostics.Debug.WriteLine("gridControl1_QueryAllowDragColumnHeader: " + e.ToString());
}
```

##### 2. Using VB.NET

```
[VB .NET]

Me.gridControl1.ControllerOptions = Me.gridControl1.ControllerOptions Or GridControllerOptions.DragColumnHeader
Private Sub gridControl1_QueryAllowDragColumnHeader(ByVal sender As Object, ByVal e As GridQueryDragColumnHeaderEventArgs)
    If e.Reason <> GridQueryDragColumnHeaderReason.HitTest Then
        System.Diagnostics.Debug.WriteLine("gridControl1_QueryAllowDragColumnHeader: " & e.ToString())
    End If
End Sub
```

The following screen shot illustrates how to drag the column header.

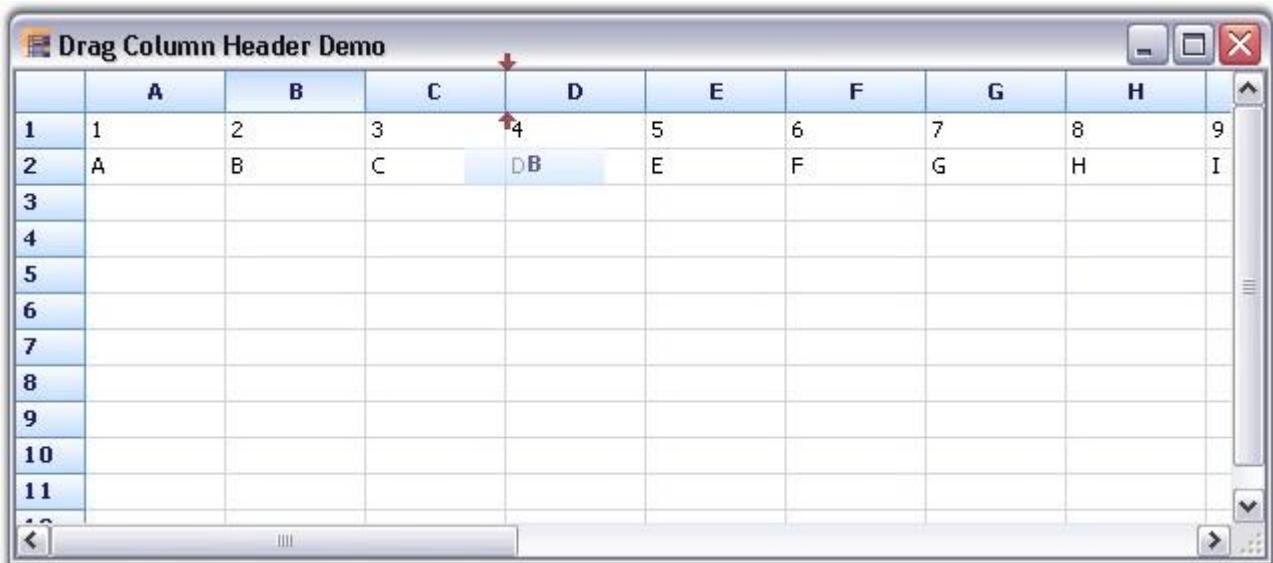


Figure 194: Drag Column Header

#### 4.1.4.23 OLE Drag-and-Drop

Essential Grid offers support functionality like Object Linking and Embedding (OLE) Drag-and-Drop. A range in one grid can be selected and dragged to another grid or into a **Rich Edit Box**. The following screen shot shows a selected region of grid, that has been dragged and dropped into another grid:

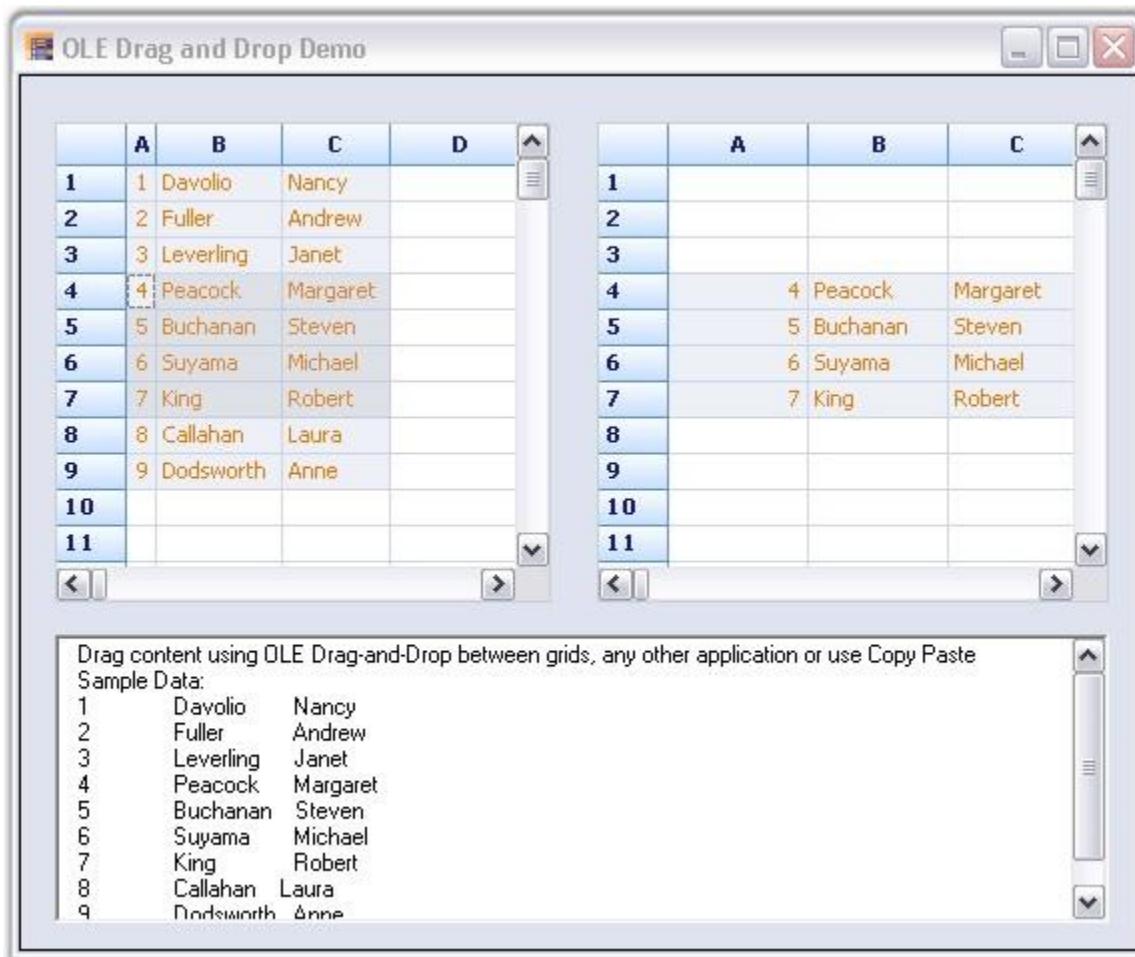


Figure 1: OLE Drag-and-Drop

The following code example illustrates this feature.

### 1. Using C#

```

[C#]

gridControl1.AllowDrop = true;
gridControl2.AllowDrop = true;

private void gridControl1_DragOver(object sender, DragEventArgs e)
{
    e.Effect = DragDropEffects.Copy;
}

private void gridControl2_DragOver(object sender, DragEventArgs e)
{
}

```

```
e.Effect = DragDropEffects.Copy;  
}
```

## 2. Using VB.NET

### [VB .NET]

```
gridControl1.AllowDrop = True  
gridControl2.AllowDrop = True  
  
private void gridControl1_DragOver(Object sender, DragEventArgs e)  
e.Effect = DragDropEffects.Copy  
  
private void gridControl2_DragOver(Object sender, DragEventArgs e)  
e.Effect = DragDropEffects.Copy
```

### 4.1.4.24 Selection Modes

Essential Grid supports different selection modes for grid cells. A specific selection behavior can be set through the **GridControl.AllowSelection** property.

The following screen shot shows a window with a list of selection modes.

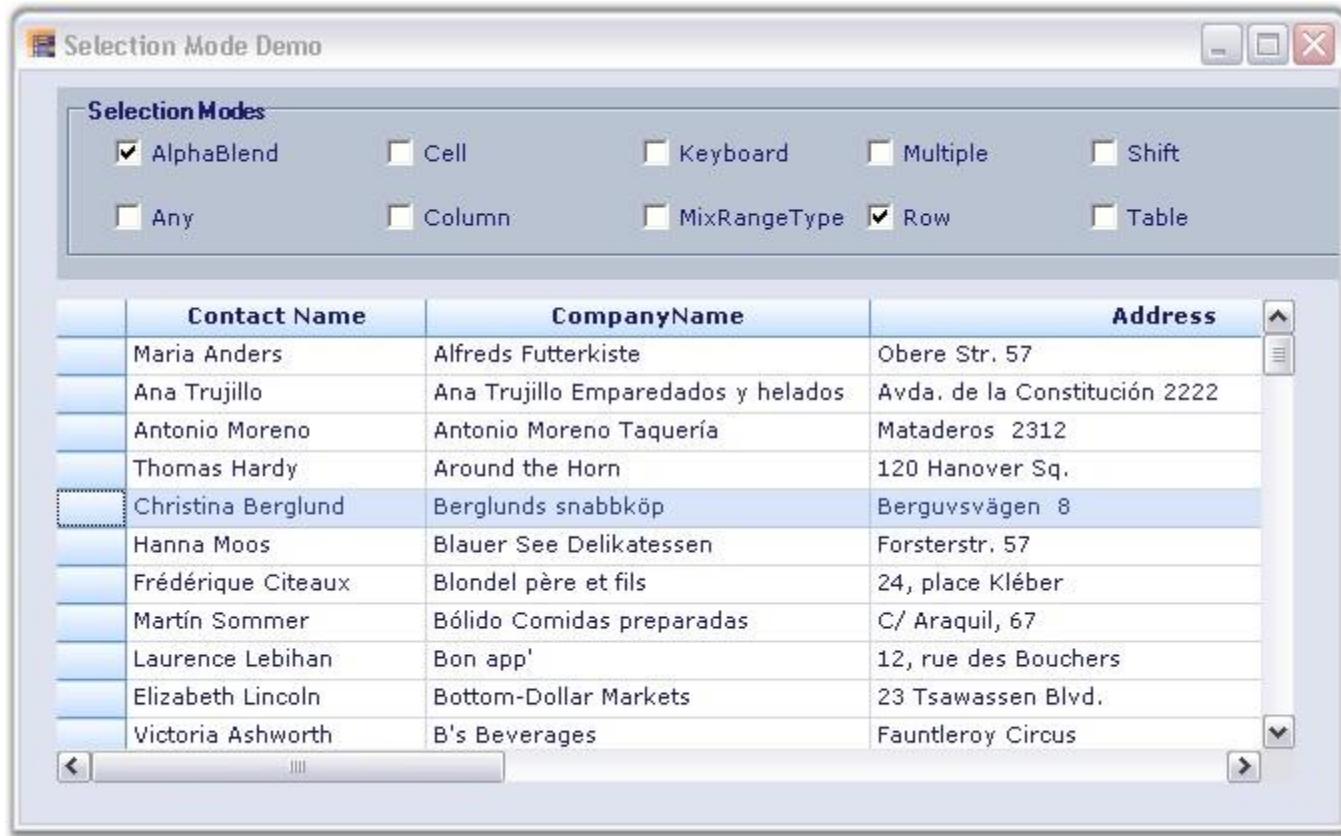


Figure 195: Selection Mode

Different types of selection modes are listed with their corresponding descriptions:

- The alpha blending to highlight selected cells can be achieved by using **GridSelectionFlags.AlphaBlend** option or selecting the **AlphaBlend** check box under **Selection Modes** group box in the UI.
- The default behavior for selecting cells, rows, columns, tables, multiple extending **SHIFT** key support, and alpha blending can be achieved by using **GridSelectionFlags.Any** option or selecting the **Any** check box under **Selection Modes** group box in the UI.
- Column selection can be achieved by using **GridSelectionFlags.Column** option or selecting the **Column** check box under **Selection Modes** group box in the UI.
- Row selection can be achieved by using **GridSelectionFlags.Row** option or selecting the **Row** check box under **Selection Modes** group box in the UI.
- An existing selection can be extended when a user holds the **SHIFT** key and uses the arrow keys by using **GridSelectionFlags.Keyboard** option or selecting the **Keyboard** check box under **Selection Modes** group box in the UI.
- Selection of both rows and columns simultaneously when multiple selection is enabled can be achieved by using **GridSelectionFlags.MixRangeType** option or selecting the **MixRangeType** check box under **Selection Modes** group box in the UI.
- Selection of entire table can be achieved by using **GridSelectionFlags.Table** option or selecting the **Table** check box under **Selection Modes** group box in the UI.

- Selection of multiple ranges of cells using the **CTRL** key can be achieved by using **GridSelectionFlags.Multiple** option or selecting the **Multiple** check box under **Selection Modes** group box in the UI.
- An existing selection using the **SHIFT** key can be extended by using **GridSelectionFlags.Shift** option or selecting the **Shift** check box under **Selection Modes** group box in the UI.
- Selection of cells using the **CTRL** key can be disabled by using **GridSelectionFlags.None** option or selecting the **Cell** check box under **Selection Modes** group box in the UI.

### **Setting Specific Selection Mode**

Specific selection modes can be set by using the following code examples:

#### 1. Using C#

**[C#]**

```
this.gridControl1.AllowSelection = GridSelectionFlags.Row;
```

#### 2. Using VB.NET

**[VB .NET]**

```
Me.gridControl1.AllowSelection = GridSelectionFlags.Row
```

### **4.1.4.25 Banner Cells**

Banner cells are multiple cells spanning a single background image. An image to be displayed in the cell can be loaded on disk, by changing the **BackgroundImage** property for a cell in the **Property Grid** and applying a Banner for the cell area, displaying the image. For a cell background color, **Gradient** style can be set. Custom cell backgrounds can be drawn by handling the **DrawCellBackground** event. The Banner cells can also be defined through a recurring pattern, by handling **QueryBanneredRange** event.

The following screen shot shows an example of how multiple cells span a single background image to form banner cells.

	A	B	C	D	E	F	G	H	I	J	K
1											
2											
3											
4											
5											
6											
7											
8											
9											
10											
11											
12											
13											
14											
15											
16											
17											
18											
19											
20											
21											
22											
23											
24											
25											
26											
27											
28											
29											

*Figure 196: Banner Cells*

### Displaying Image using Banner Cells

The following code examples illustrate how to display images by using banner cells:

#### 1. Using C#

[C#]

```
GridStyleInfo style;
style = grid[4, 3];
grid.BanneredRanges.Add(GridRangeInfo.FromTlhw(4, 3, 8, 3));
style.BackgroundImage =
GetImage(@"common\Images\Grid\BannerCells\back1.jpg");
style.Text = "back1.jpg";
```

```
style.TextColor = Color.Red;
style.BackgroundImageMode = GridBackgroundImageMode.StretchImage;
```

## 2. Using VB.NET

### [VB .NET]

```
Dim style As GridStyleInfo
style = grid(4, 3)
grid.BanneredRanges.Add(GridRangeInfo.FromTlhw(4, 3, 8, 3))
style.BackgroundImage =
GetImage("common\Images\Grid\BannerCells\back1.jpg")
style.Text = "back1.jpg"
style.TextColor = Color.Red
style.BackgroundImageMode = GridBackgroundImageMode.StretchImage
```

### 4.1.4.26 Merge Cells Feature

The Merge Cells feature merges two or more adjacent cells with the same value into one cell and displays the content in the merged cell. A single cell is created by combining two or more selected cells.



**Note:** The data in the merged cells will be displayed on the first cell of the merged range.

To use merge cells, you need to set **Model.Options.MergeCellsMode** and **MergeCell** properties of the cells to select the required merge behavior for cells.

1. The **GridMergeCellsMode** enumeration specifies behavior of the merged cells in a grid. Following is the list of options under this enumeration:

- **None-**

Merge cells behavior is disabled.

- **OnDemandCalculation-**

The number of cells to be merged are calculated before the merged cells are displayed and the results are saved. Floating cells will only be recalculated if the width or content of the cells change.

- **BeforeDisplayCalculation-**

The number of cells to be merged are always calculated before cells are displayed.

- **MergeRowsInColumn-**  
Enables merging of neighboring cells among rows in the same column.
  - **MergeColumnsInRow-**  
Enables merging of neighboring cells among columns in the same row.
  - **SkipHiddenCells-**  
Skips hidden rows and columns while merging the cells. This means that the hidden rows or columns in the grid are not considered during the merge process.
2. The **GridMergeCellDirection** enumeration specifies the merge behavior for an individual cell when merging cells feature has been enabled. Here is the list of options offered:
- **None-**  
Merging cells is disabled.
  - **ColumnsInRow-**  
Merges with neighboring columns in the same row.
  - **RowsInColumn-**  
Merges with neighboring rows in the same column.

The following code examples illustrate how to set the MergeCellsMode and MergeCell properties.

a. Using C#

[C#]

```
this.gridControl1.Model.Options.MergeCellsMode =
GridMergeCellsMode.OnDemandCalculation |
GridMergeCellsMode.MergeRowsInColumn;

// Merge cells in column 2.
this.gridControl1.ColStyles[2].MergeCell =
GridMergeCellDirection.RowsInColumn;
```

b. Using VB.NET

[VB .NET]

```
Me.gridControl1.Model.Options.MergeCellsMode =
GridMergeCellsMode.OnDemandCalculation Or
GridMergeCellsMode.MergeRowsInColumn
```

```
' Merge cells in column 2.  
Me.gridControl1.ColStyles(2).MergeCell =  
GridMergeCellDirection.RowsInColumn
```

The following screen shots illustrate the merge cells feature in Essential Grid.

A	B	C	D	E
0	0			
0	0		Test	
0	0		Test	
0	0		Test	
0	0		Test	
0	0		Test	
1	0		Test	
1	0		Test	
1	0		Test	
1	0		Test	
1	0			
1	0			
1	0			

*Figure 197: Before Merging*

	A	B	C	D	E
0	0				
	0			Test	
	0				
	0				
	0				
	0				
1	0				
	0				
	0				
	0				
	0				
	0				
	0				
	0				

*Figure 198: After Merging*

#### **4.1.4.27 Real-time Applications**

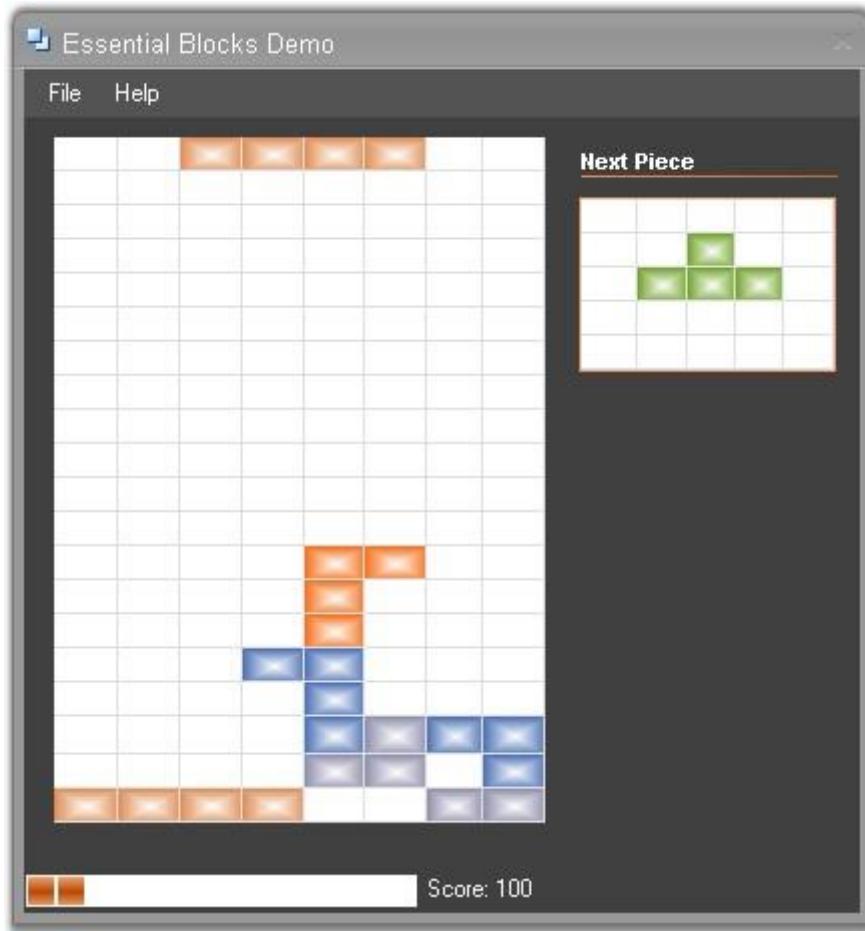
This section provides information on the real-time applications of Essential Grid. It includes the following topics.

##### **4.1.4.27.1 Gaming Applications**

Essential Grid is used to implement gaming applications.

##### **Example**

Let us consider the Tetris game application which is implemented by using Essential Grid. This game application makes use of arrow keys to move the blocks and change the block shape. It also provides an option to display the next block.



*Figure 199: Tetris game implemented by using Grid*

A sample demonstrating this feature is available under the following sample installation path.

**<Install Location>\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Windows\Samples\2.0\Product Showcase\Essential Blocks Demo**

#### 4.1.4.27.2 MS Excel 2007-like UI

Essential Grid control provides support to implement the Microsoft Excel 2007-like User Interface (UI). It contains a **Name Box** that shows the current selection range, and a **Formula Bar** which supports the formula cells. The row and column headers of the selected range are highlighted on the UI. This feature is similar to Excel.

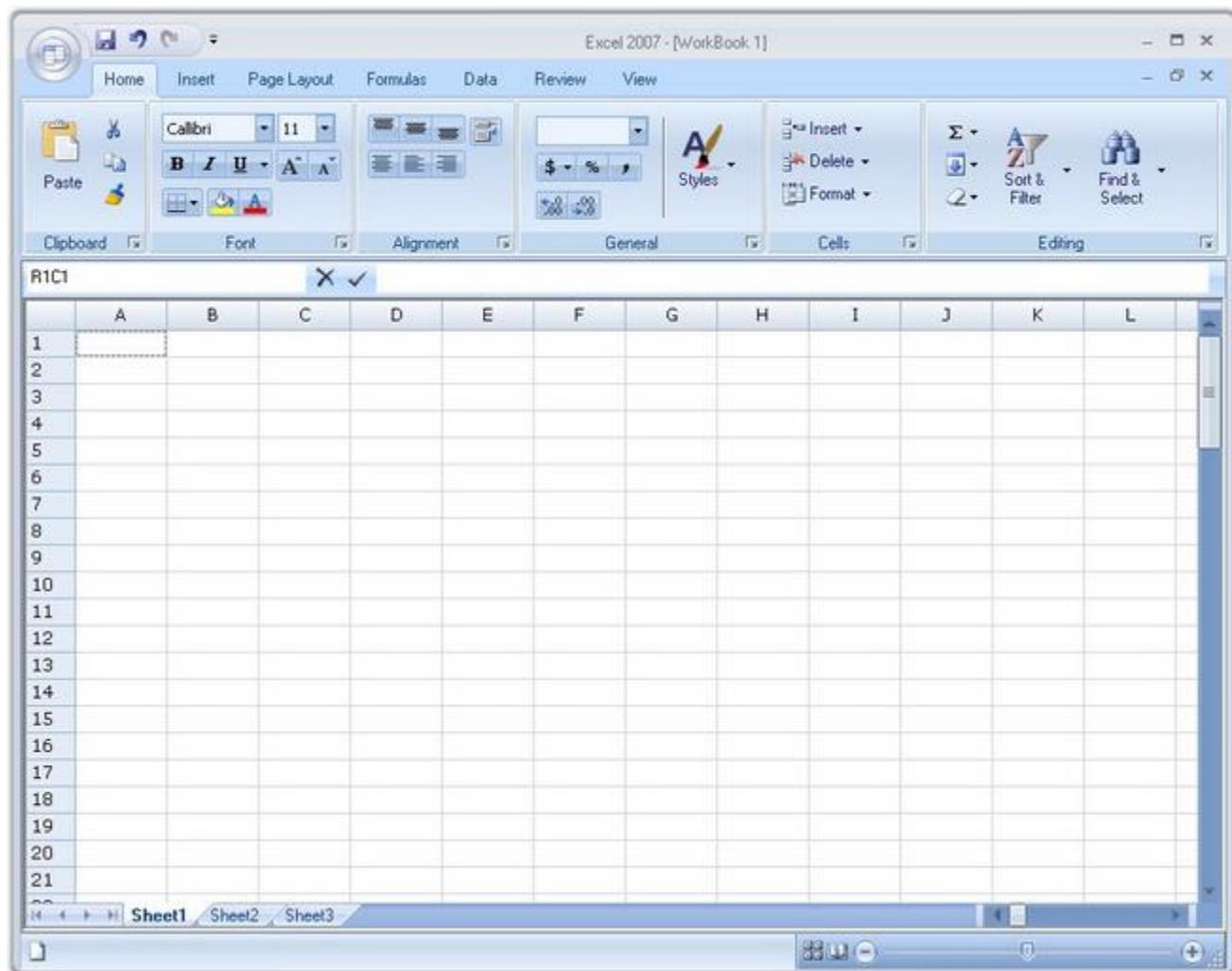


Figure 200: Microsoft Excel 2007-like UI implemented by using Grid

A sample demonstrating this feature is available under the following sample installation path.

**<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Windows\Samples\2.0\Product Showcase\Excel Like UI Demo**

#### 4.1.4.27.3 Grid Folder Browser

The Essential Grid can be used to develop a powerful **TreeView** control owing to its flexibility. The tree nodes can be created through a custom **TreeCell** cell type. The **GridStaticCellModel** class is inherited to create this cell type. The plus/minus buttons of the tree nodes are selected by using the **ImageIndex** property.

## Example

Let us consider the Grid Folder Browser sample which is available under the following installation path.

**<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Windows\Samples\2.0\Product Showcase\Grid Folder Browser Demo**

This sample operates the grid in virtual mode in order to populate the data dynamically on demand, i.e., when the tree is expanded. The QueryCellInfo, QueryColCount and QueryRowCount events must be handled in order to implement a virtual grid. These events provide basic information about the number of rows and columns and the values of the data.

The following code examples illustrates how to set the data from the data source.

### 1. Using C#

```
[C#]

void GridQueryCellInfo(object sender, GridQueryCellInfoEventArgs e)
{
    if (e.RowIndex > 0 && e.ColIndex > 0)
    {
        e.Style.CellValue = externalData[e.RowIndex - 1].Items[e.ColIndex - 1];
        if (e.ColIndex == 1)
        {
            e.Style.CellType = "TreeCell";
            e.Style.Tag = externalData[e.RowIndex - 1].IndentLevel;
            e.Style.ImageIndex = (int) externalData[e.RowIndex - 1].ExpandState;
        }
    }
    e.Handled = true;
}
```

### 2. Using VB.NET

```
[VB.NET]

Private Sub GridQueryCellInfo(ByVal sender As Object, ByVal e As
```

```
GridQueryCellInfoEventArgs)
    If e.RowIndex > 0 AndAlso e.ColumnIndex > 0 Then
        e.Style.CellValue = externalData(e.RowIndex - 1).Items(e.ColumnIndex - 1)
        If e.ColumnIndex = 1 Then
            e.Style.CellType = "TreeCell"
            e.Style.Tag = externalData(e.RowIndex - 1).IndentLevel
            e.Style.ImageIndex = CInt(Fix(externalData(e.RowIndex - 1).ExpandState))
        End If
    End If
    e.Handled = True
End Sub
```

The implementation uses a CollapsibleDataSource class. This class makes use of a custom collection to hold a list of SampleData objects (Consider each of these objects as a row in the underlying grid). Each row carries information on a specific folder. Each SampleData object has an IndentValue property, an ExpandState property, and an Items string array that holds the different column values for this row. The column values display the folder details like the name of the folder, folder size and so on. This class also contains the InsertData method which retrieves the data of the inner subtree and inserts the data into the collection when a node is expanded.

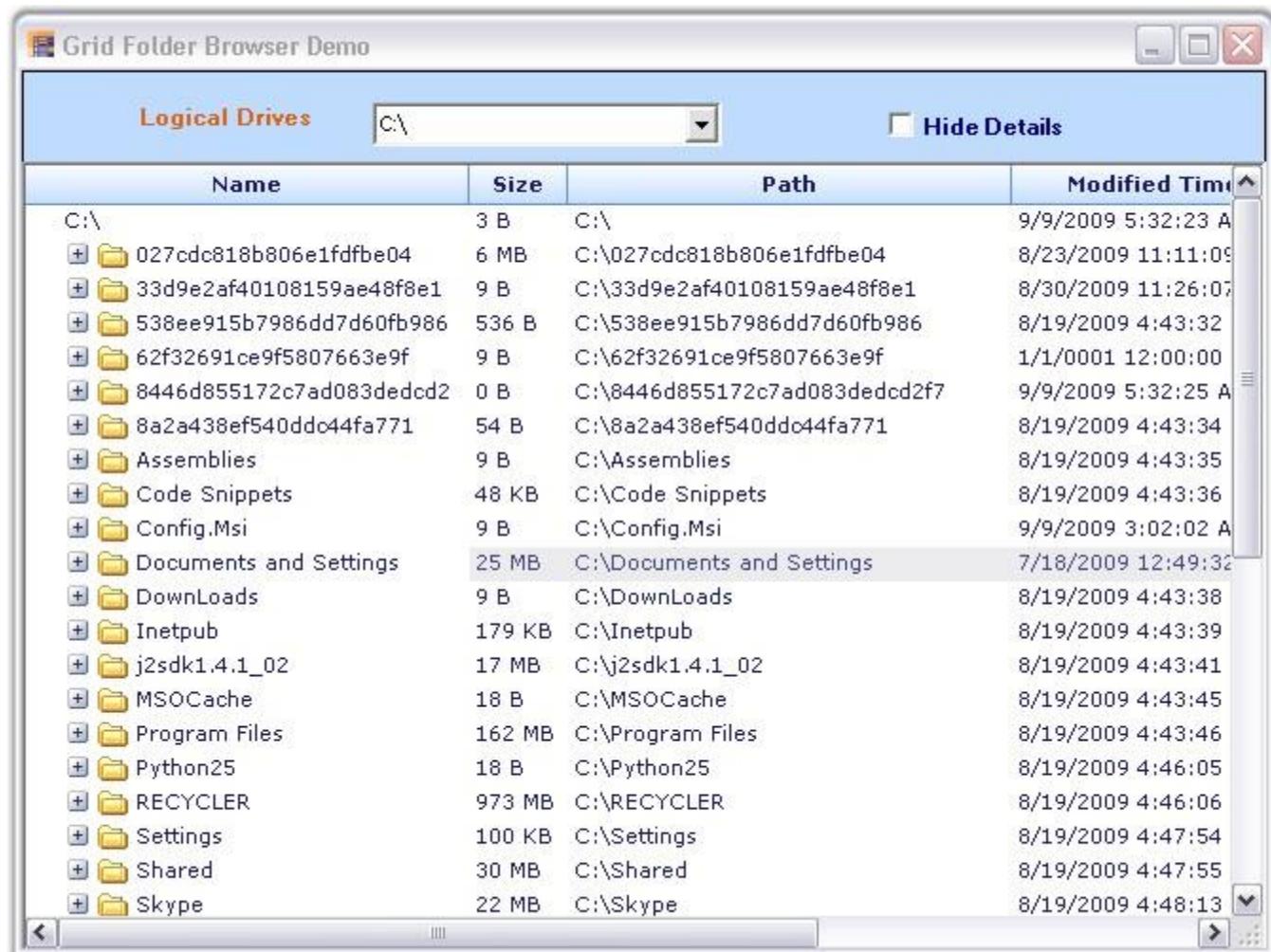
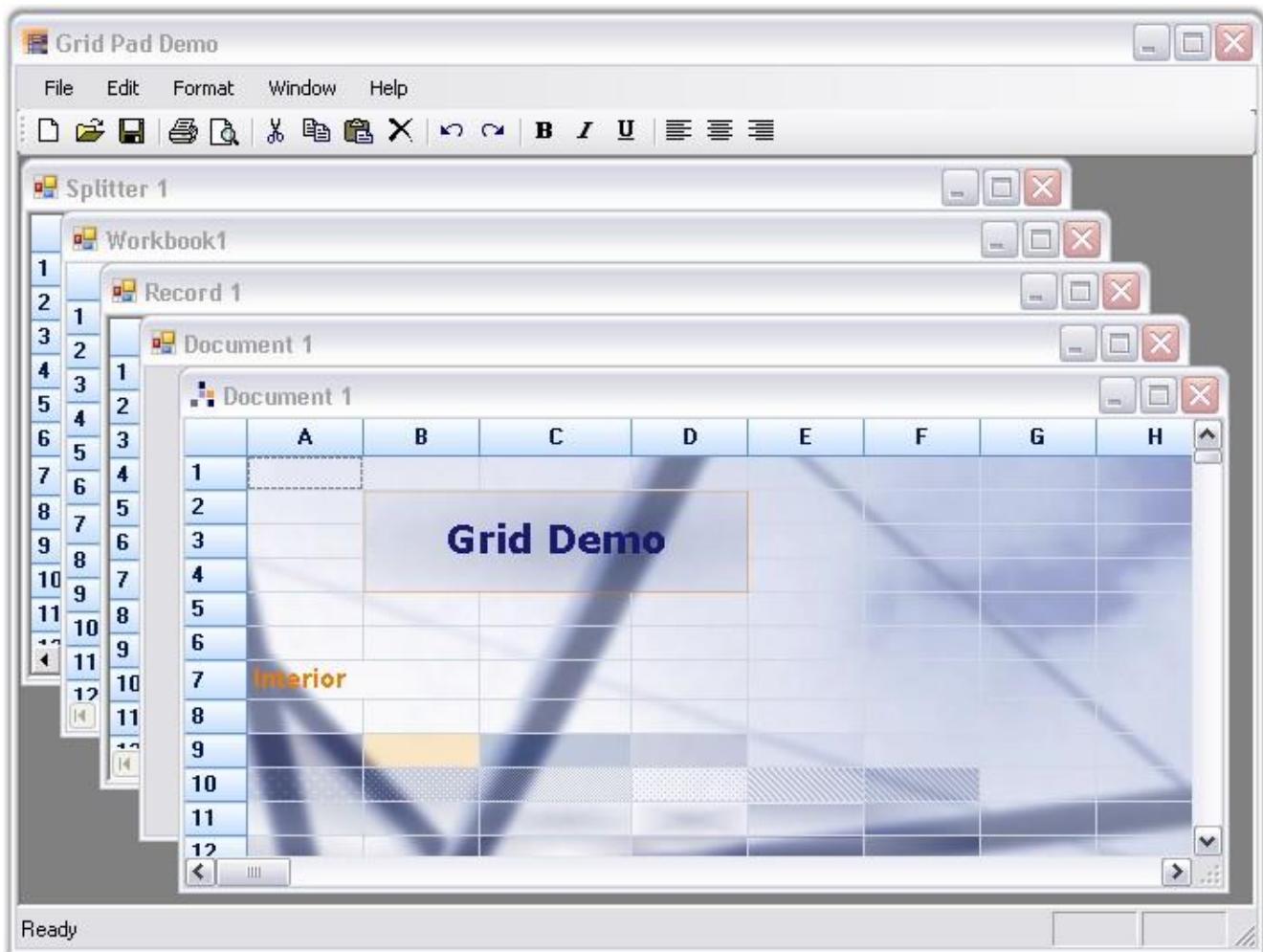


Figure 201: Grid Folder Browser Demo

#### 4.1.4.28 Multiple Document Interface (MDI) Support

Essential Grid supports Multiple Document Interface (MDI) by enabling users to work with multiple grid controls simultaneously. Here, multiple windows reside under a single parent window.



*Figure 202: Multiple Document Interface Illustrated*

A sample demonstrating this feature is available under the following sample installation path.

**<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Windows\Samples\2.0\Product Showcase\Grid Pad Demo**

#### **4.1.4.29 Clipboard Support**

This section discusses the clipboard support provided by Essential Grid. It includes the following topics.

#### 4.1.4.29.1 Managing Clipboard Operations with GridModelCutPaste

The **GridModelCutPaste** class manages Cut, Copy and Paste operations for a grid. You can access this class from a grid with the **Grid.Model.CutPaste** property. This class provides many properties and functions.

Here is the list of properties and methods:

- **ClipboardFlags**-This property gets or sets various properties of GridDragDropFlags class, which specifies how the clipboard operations like cut, copy, and paste should be handled.

The following code examples illustrate how to set ClipboardFlags by using the GridDragDropFlags.Styles property:

- Using C#

**[C#]**

```
this.gridControl1.Model.CutPaste.ClipboardFlags =  
GridDragDropFlags.Styles;
```

- Using VB.NET

**[VB .NET]**

```
Me.gridControl1.Model.CutPaste.ClipboardFlags =  
GridDragDropFlags.Styles
```

- **CanCopy**-This method checks whether there are selected ranges of cells that can be copied to clipboard or if the current cell's contents can be copied. The return type of this method is Boolean. If it returns true, it indicates that the selected range of cells or the current cell's contents can be copied to the clipboard. If it is false, it indicates that the selected range of cells or the current cell's contents cannot be copied to the clipboard.



**Note:** Any content copied is pasted to the Clipboard by default.

The following code examples are used to call the CanCopy method:

- Using C#

**[C#]**

```
this.gridControl1.Model.CutPaste.CanCopy();
```

- Using VB.NET

**[VB.NET]**

```
Me.gridControl1.Model.CutPaste.CanCopy()
```

- **Copy**-This method copies the contents of selected cells and the current cell's contents to the clipboard.

The following code examples are used to call the Copy method:

- Using C#

**[C#]**

```
this.gridControl1.Model.CutPaste.Copy();
```

- Using VB.NET

**[VB.NET]**

```
Me.gridControl1.Model.CutPaste.Copy()
```

- **CopyRange(GridRangeInfo range)**-This method copies the contents of a specified range of cells to the clipboard. The range of cells to be copied is given to the method as a parameter. For example, if the range is specified to be (2,2), the selection is restricted to the cell with row index 2 and column index 2 of the grid.

The following code examples show how to call this method:

- Using C#

**[C#]**

```
this.gridControl1.Model.CutPaste.CopyRange(GridRangeInfo.Cell(2, 2));
```

- Using VB.NET

**[VB.NET]**

```
Me.gridControl1.Model.CutPaste.CopyRange(GridRangeInfo.Cell(2, 2))
```

- **CopyTextToClipboard(GridRangeInfo range)**-This method copies the formatted text of a specified range of cells to clipboard. The range of cells to be copied is given to the method as a parameter. For example, if the range is specified to be (1,2,1,4), the

selection starts from the cell (1,2) – with row index 1 and column index 2 to the cell (1,4) – with row index 1 and column index 4 of the grid.

The following code examples show how to call this method:

- Using C#

**[C#]**

```
this.gridControl1.Model.CutPaste.CopyTextToClipboard(GridRangeInfo.Cells(1, 2, 1, 4));
```

- Using VB.NET

**[VB .NET]**

```
Me.gridControl1.Model.CutPaste.CopyTextToClipboard(GridRangeInfo.Cells(1, 2, 1, 4))
```

- **CopyCellsToClipboard(GridRangeInfoList list, bool bLoadBaseStyles)**-This method copies the style information of a specified range of cells to clipboard. The range of cells to be copied is given to the method as the first parameter. The second parameter represents a Boolean value. The base style will be copied along with default settings, if it is set to true. Only the default settings that were initialized to the cell are copied if it is set to false.

The following code examples show how to call this method:

- Using C#

**[C#]**

```
GridRangeInfoList list = new GridRangeInfoList();
list.Add(GridRangeInfo.Cell(2, 2));
this.gridControl1.Model.CutPaste.CopyCellsToClipboard(list, true);
```

- Using VB.NET

**[VB .NET]**

```
Dim list As New GridRangeInfoList()
list.Add(GridRangeInfo.Cell(2, 2))
Me.gridControl1.Model.CutPaste.CopyCellsToClipboard(list, True)
```

- **CanCut**-This method checks if there are selected ranges that can be cut or if the current cell's contents can be cut. The return type of this method is Boolean. If it returns true, it indicates that the content in the selected range of cells or the current cell's content can be cut. This method returns false, when no selected range is available to cut.



**Note:** Any content cut is pasted to the clipboard by default.

The following code examples show how to call this method:

- Using C#

**[C#]**

```
this.gridControl1.Model.CutPaste.CanCut();
```

- Using VB.NET

**[VB .NET]**

```
Me.gridControl1.Model.CutPaste.CanCut()
```

- **Cut**-This method cuts the content of the selected cells and the current cell, and pastes them to the clipboard.

The following code examples show how to call this method:

- Using C#

**[C#]**

```
this.gridControl1.Model.CutPaste.Cut();
```

- Using VB.NET

**[VB .NET]**

```
Me.gridControl1.Model.CutPaste.Cut()
```

- **CutRange(GridRangeInfo rangelist)**-This method cuts the content of a specified range of cells and pastes it to the clipboard. The range of cells to be cut is specified as a parameter.

The following code examples show how to call this method:

- Using C#

[C#]

```
this.gridControl1.Model.CutPaste.CutRange(GridRangeInfo.Row(4), false);
```

o Using VB.NET

[VB .NET]

```
Me.gridControl1.Model.CutPaste.CutRange(GridRangeInfo.Row(4), False)
```

- **CanPaste**-This method checks for the most recent content in the clipboard that can be pasted into the grid. The return type of this method is Boolean. If it returns true, it indicates that the contents in the clipboard can be pasted into the grid. If there is no content available in the clipboard to paste, this method returns false.

The following code examples show how to call this method:

o Using C#

[C#]

```
this.gridControl1.Model.CutPaste.CanPaste();
```

o Using VB.NET

[VB .NET]

```
Me.gridControl1.Model.CutPaste.CanPaste()
```

- **Paste**-This method pastes the content from the clipboard into the grid at the current selected range or current cell.



**Note:** *It is not mandatory to call this method after CanPaste method. If there is no content in the clipboard to be pasted, this method will not respond.*

The following code examples show how to call this method:

o Using C#

[C#]

```
this.gridControl1.Model.CutPaste.Paste();
```

o Using VB.NET

[VB.NET]

```
Me.gridControl1.Model.CutPaste.Paste()
```

#### 4.1.4.29.2 Clipboard Events

Essential Grid exposes various events to manage the clipboard content while performing cut, paste and copy operations. Following is the list of clipboard events:

- **ClipboardCanCopy**-This event is to be handled when the CanCopy method of grid is called.
- **ClipboardCanPaste**-This event is to be handled when the CanPaste method of grid is called.
- **ClipboardCanCut**-This event is to be handled when the CanCut method of grid is called.
- **ClipboardCopy**-This event is to be handled when the Copy method of grid is called.
- **ClipboardCut**-This event is to be handled when the Cut method of grid is called.
- **ClipboardPaste**-This event is to be handled when the Paste method of grid is called.
- **ClipboardPasted**-This event is to be handled after a paste operation.

#### 4.1.4.30 Performance

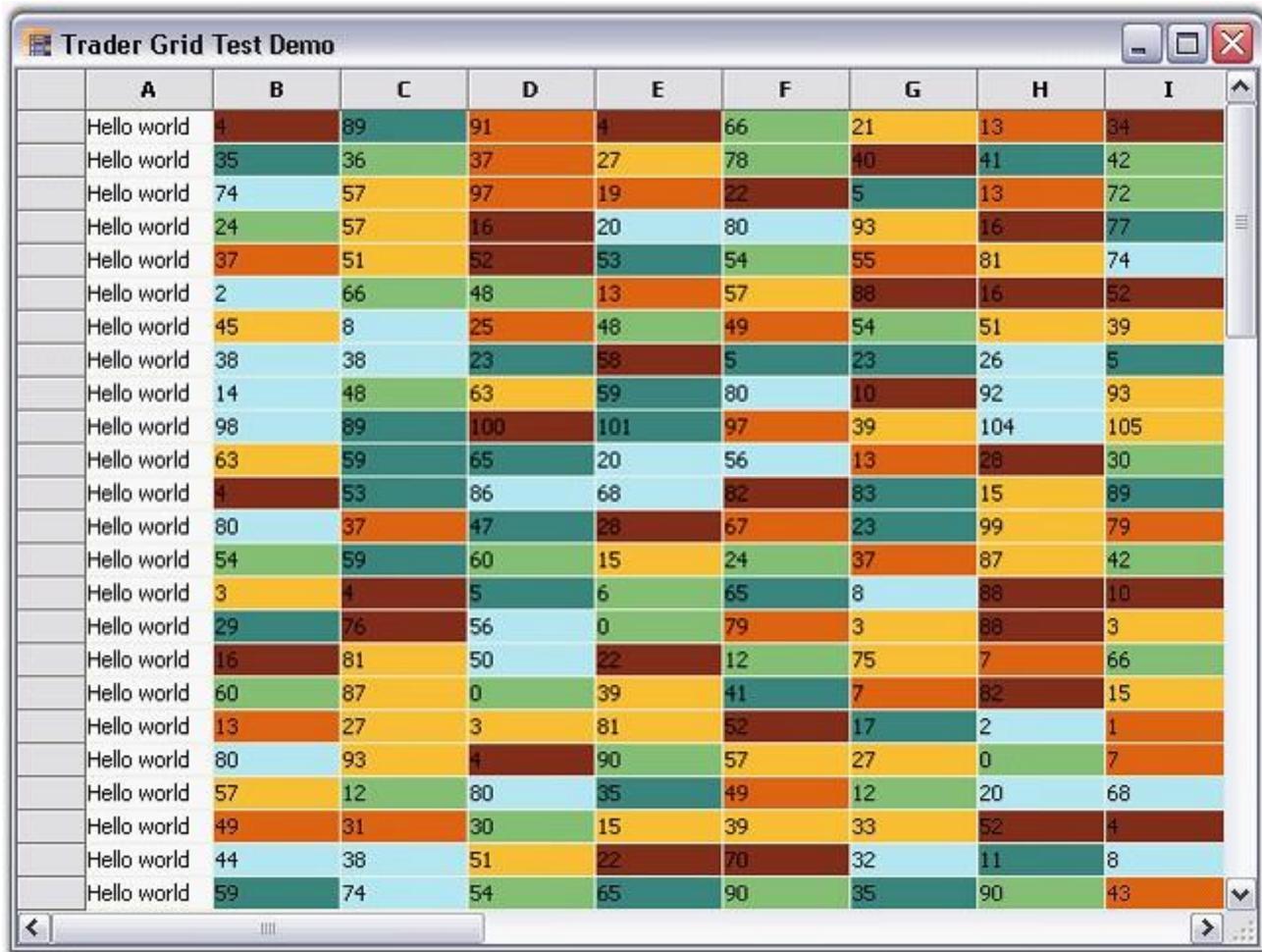
Essential Grid control has an extremely high performance standard. It can handle high frequency updates and work with large amounts of data without its performance being affected.

##### 4.1.4.30.1 High Frequency Real Time Updates

Essential Grid supports frequent updates that occur in random cells across the grid while keeping CPU usage to a minimum level.

**Example:**

In this example, a timer changes the cell in the short intervals, inserts and removes rows. This example draws cell changes directly to the graphics context instead of performing an Invalidate. It also shows you draw text using the GDI instead of the GDI+ and how to optimize updates for inserting and removing rows. You can start multiple instances without slowing down your machine. You can confirm the same by viewing the TaskManager CPU usage while the sample runs.



The screenshot shows a Windows application window titled "Trader Grid Test Demo". The main area contains a 20x10 grid of cells, each containing a numerical value. The cells are colored using a gradient scheme: dark red for low values, light yellow for high values, and various shades of green, blue, and orange in between. The values themselves are mostly single-digit numbers, ranging from 0 to 99. The grid has a header row labeled A through I across the top. The first column is labeled "Hello world" in all its cells. The application has standard Windows-style window controls (minimize, maximize, close) at the top right and scroll bars on the right and bottom edges.

Figure 203: Trader Grid

A sample demonstrating this feature is available under the following sample installation path.

**<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Windows\Samples\2.0\Performance\Trader Grid Test Demo**

#### 4.1.4.30.2 Data Handling

Essential Grid control supports large amount of data without a performance hit.

##### Example:

This example will step you through following three ways of populating an Essential Grid:

- The first technique just loops through the cells and uses an indexer on the Grid control to set the values.
- The second uses the PopulateValues method that optimally places data from a data source into a grid range.
- The third technique uses an Essential Grid in a virtual manner.

You can specify the size of the grid that is to be populated while running the sample and then you can try all the three methods to compare the performance. However, the .NET Framework JIT slows the first population owing to one-time jittering of the code.

##### 1. Using Indexer

This technique loops through the cells and uses an indexer on the Grid control to set the values.

###### a. Using C#

###### [C#]

```
for (int i = 0; i < this.numArrayRows; ++i)
    for (int j = 0; j < this.numArrayCols; ++j)
        this.gridControl1[i + 1, j + 1].CellValue = this.intArray[i, j];
```

###### b. Using VB.NET

###### [VB .NET]

```
For i As Integer = 0 To Me.numArrayRows - 1
    For j As Integer = 0 To Me.numArrayCols - 1
        Me.gridControl1(i + 1, j + 1).CellValue = Me.intArray(i, j)
    Next j
Next i
```

##### 2. Populating Values

PopulateValues method is used to move values from a given data source into the specified grid range. The first parameter specifies range of destination cells where the data is to be copied and the second parameter specifies the data source to the destination cells.

a. Using C#

[C#]

```
gridControl1.Model.PopulateValues(GridRangeInfo.Cells(top, left,
bottom, right), this.intArray);
```

b. Using VB.NET

[VB .NET]

```
gridControl1.Model.PopulateValues(GridRangeInfo.Cells(top, left,
bottom, right), Me.intArray)
```

### 3. Implementing Virtual Mode

Three events need to be handled in order to implement a virtual mode. They perform the following actions:

- Determine number of rows
- Determine number of columns
- Pass value to a cell from a data source.

a. Using C#

[C#]

```
// Determine number of rows.
this.gridControl1.QueryRowCount += new
GridRowColCountEventHandler(GridQueryRowCount);
private void GridQueryRowCount(object sender, GridRowColCountEventArgs
e)
{
    e.Count = this.numArrayRows;
    e.Handled = true;
}

// Determine the number of columns.
this.gridControl1.QueryColCount += new
GridRowColCountEventHandler(GridQueryColCount);
private void GridQueryColCount(object sender, GridRowColCountEventArgs
e)
```

```
{  
    e.Count = this.numArrayCols;  
    e.Handled = true;  
}  
  
// Pass value to a cell from a given data source.  
this.gridControl1.QueryCellInfo += new  
GridQueryCellInfoEventHandler(QueryCellInfoHandler);  
private void GridQueryCellInfo(object sender,  
GridQueryCellInfoEventArgs e)  
{  
    if(e.ColIndex > 0 && e.RowIndex > 0)  
    {  
        // By using indexers, pass value to a cell from a given data  
        // source.  
        e.Style.CellValue = this.intArray[e.RowIndex - 1, e.ColIndex -  
1];  
        e.Handled = true;  
    }  
}
```

b. Using VB.NET

**[VB.NET]**

```
' Determine number of rows.  
Private Me.gridControl1.QueryRowCount += New  
GridRowColCountEventHandler(AddressOf GridQueryRowCount)  
Private Sub GridQueryRowCount(ByVal sender As Object, ByVal e As  
GridRowColCountEventArgs)  
    e.Count = Me.numArrayRows  
    e.Handled = True  
End Sub  
  
// Determine the number of columns.  
Private Me.gridControl1.QueryColCount += New  
GridRowColCountEventHandler(AddressOf GridQueryColCount)  
Private Sub GridQueryColCount(ByVal sender As Object, ByVal e As  
GridRowColCountEventArgs)  
    e.Count = Me.numArrayCols  
    e.Handled = True  
End Sub  
  
// Pass value to a cell from a given data source.  
Private Me.gridControl1.QueryCellInfo += New
```

```
GridQueryCellInfoEventHandler(AddressOf QueryCellInfoHandler)
Private Sub GridQueryCellInfo(ByVal sender As Object, ByVal e As
GridQueryCellInfoEventArgs)
    If e.ColumnIndex > 0 AndAlso e.RowIndex > 0 Then

        // By using indexers, pass value to a cell from a given data
        // source.
        e.Style.CellValue = Me.intArray(e.RowIndex - 1, e.ColumnIndex - 1)
        e.Handled = True
    End If
End Sub
```

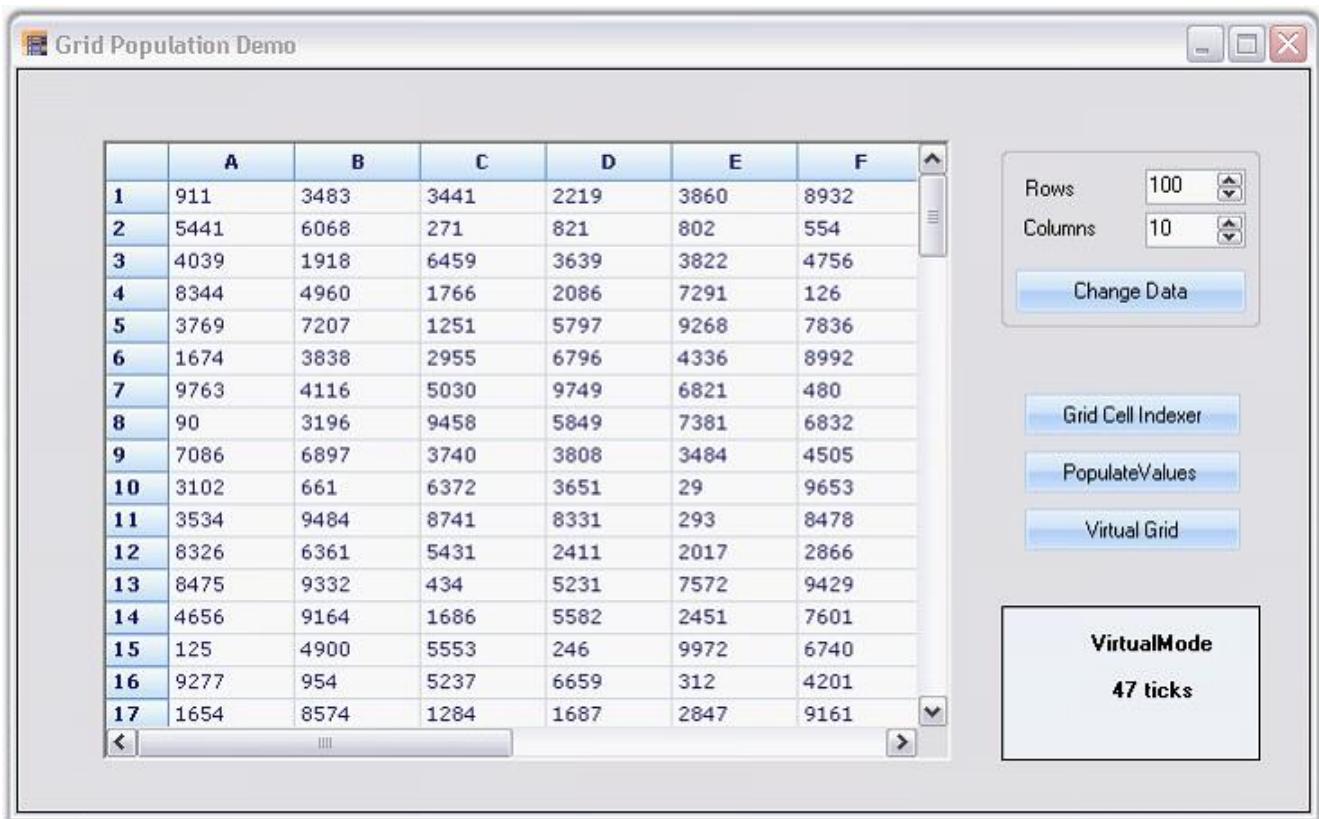


Figure 204: Grid Population

A sample demonstrating this feature is available under the following sample installation path.

<Install Location>\Syncfusion\EssentialStudio\Version  
Number\Windows\Grid.Windows\Samples\2.0\Performance\Trader Grid Test Demo

#### 4.1.4.31 Zooming options

This section discusses the following zooming options:

##### 4.1.4.31.1 Cell-level and grid-level

We can implement zooming functionality in Essential Grid to show a magnified image of the Grid for better visualization. A method named `ZoomGrid` is used for this purpose. It accepts the percentage of zoom as the parameter and then uses this value to set the font size and cell size for the grid. Zooming can be implemented at cell-level and at grid-level.

The preceding screen shot shows a grid that is zoomed to the grid-level.

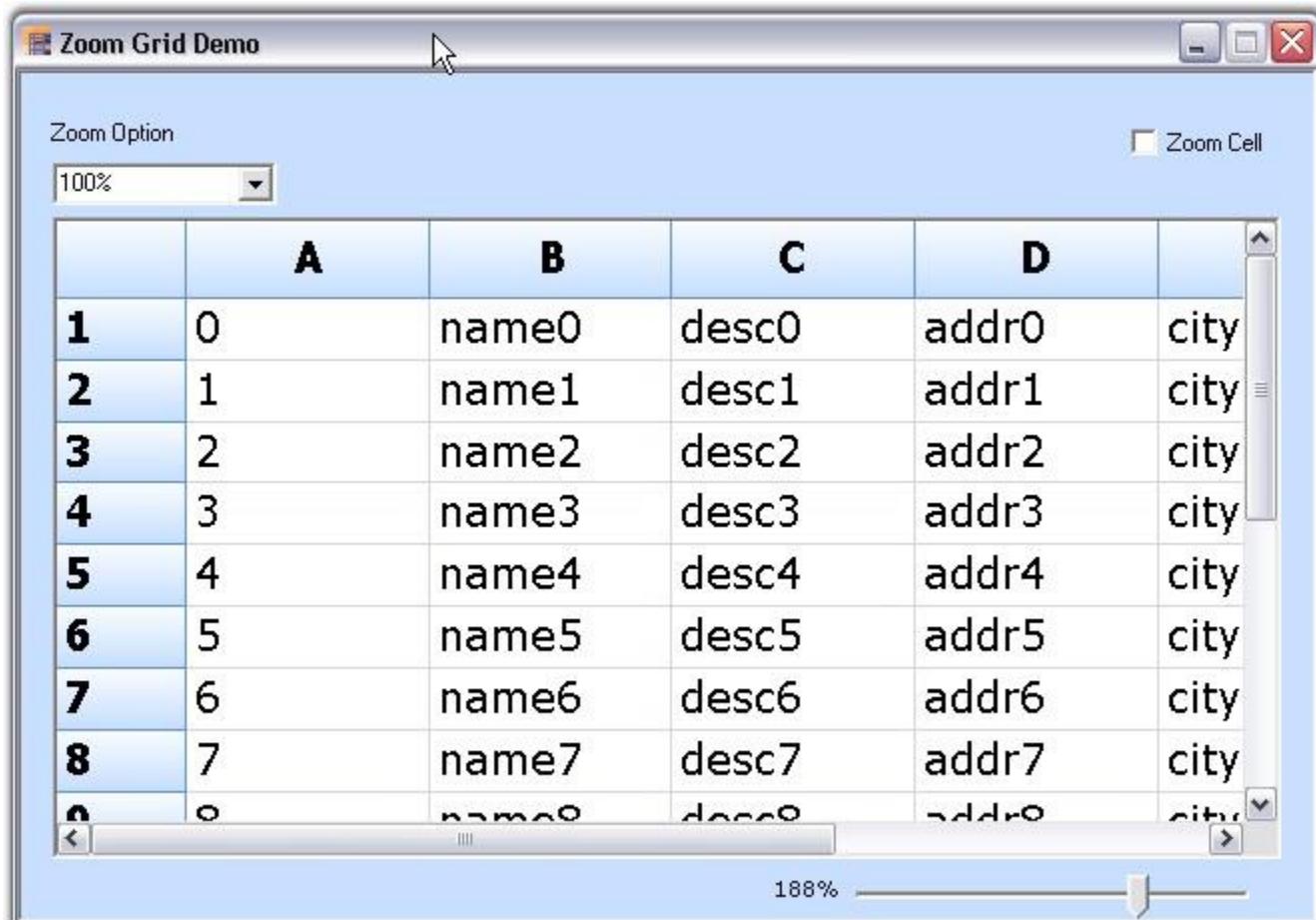


Figure 205: Grid zoomed at Grid Level

The preceding screen shot shows a grid that is zoomed to the cell-level.

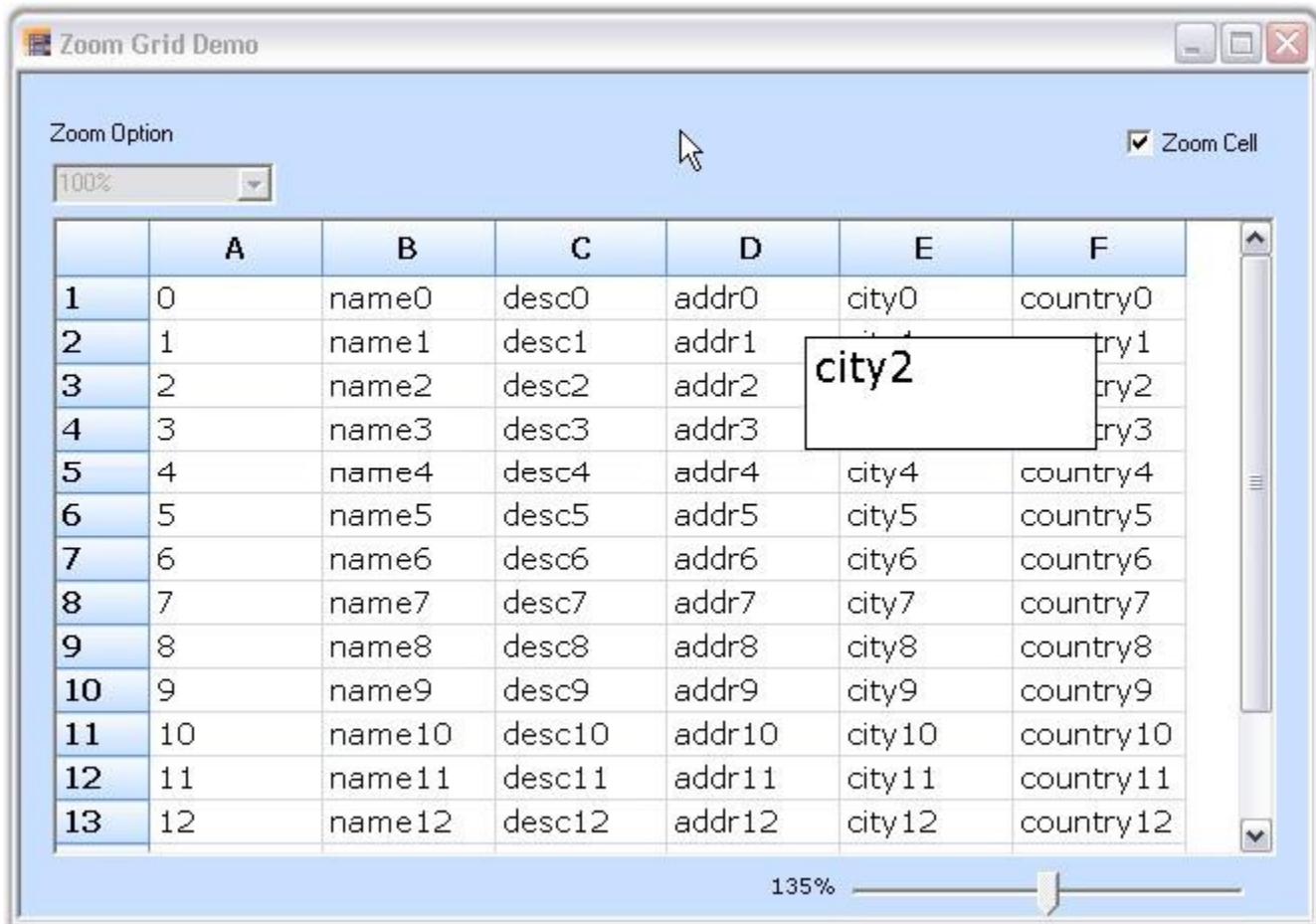


Figure 206: Grid zoomed at Cell Level

### Example

This example implements zooming functionality for Essential Grid at cell-level.

Follow the steps listed below:

1. Call the **ZoomGrid** method by passing the percentage of zoom as the parameter. The below code illustrates this.

```
[C#]

private void zoomGrid(float percent)
{
```

```
this.gridControl1.BeginUpdate();
currentPercent = percent;
for (int i = 0; i <= gridControl1.ColCount; i++)
{
    for (int j = 0; j <= gridControl1.RowCount; j++)
        this.gridControl1[j, i].Font.Size = fontSize * currentPercent;
    this.gridControl1.Model.ColWidths[i] = (int)(percent *
defColWidth);
}

this.gridControl1.ColWidths[0] = (int)(percent * headerColWd);
this.gridControl1.DefaultColWidth = (int)(percent * defColWidth);
this.gridControl1.DefaultRowHeight = (int)(percent * defRowHeight);
this.gridControl1.RowHeights[0] = (int)(percent * headerRowHt);
this.gridControl1.EndUpdate();
this.gridControl1.Refresh();
}
```

**[VB.NET]**

```
Private Sub zoomGrid(ByVal percent As Single)
    Me.label1.Text = Me.trackBar1.Value.ToString() & "%"
    Me.label1.Refresh()

    Me.gridControl1.BeginUpdate()
    currentPercent = percent
    For i As Integer = 0 To gridControl1.ColCount
        For j As Integer = 0 To gridControl1.RowCount
            Me.gridControl1(j, i).Font.Size = fontSize * currentPercent
        Next j
        Me.gridControl1.Model.ColWidths(i) = CInt(Fix(percent *
defColWidth))
    Next i

    Me.gridControl1.ColWidths(0) = CInt(Fix(percent * headerColWd))
    Me.gridControl1.DefaultColWidth = CInt(Fix(percent * defColWidth))
    Me.gridControl1.DefaultRowHeight = CInt(Fix(percent *
defRowHeight))
    Me.gridControl1.RowHeights(0) = CInt(Fix(percent * headerRowHt))
    Me.gridControl1.EndUpdate()
    Me.gridControl1.Refresh()
End Sub
```

The preceding code sets the font and cell size using the percent parameter.

- When the cell to be zoomed is clicked, handle the CellClick event to display it as a zoomed cell. Here we use a PictureBox to show the magnified view of the cell content.



**Note:** A PictureBox is a Microsoft's .NET Control used to display an image.

[C#]

```
// Code to show zoom window.
private System.Windows.Forms.PictureBox zoomWindow;
private void gridControl1_CellClick(object sender,
GridCellEventArgs e)
{
    if (e.RowIndex > 0 && e.ColumnIndex > 0)
    {
        if (checkBox1.Checked)
        {
            if (!zoomWindow.Visible)
                this.zoomWindow.Visible = true;
            Point p1 = new Point(0, 0);
            Size s = new Size(this.gridControl1.ColumnWidths[e.ColumnIndex] +
10, this.gridControl1.RowHeaders[e.RowIndex] + 5);
            s.Width += 50;
            s.Height += 30;
            Rectangle rect = new Rectangle(p1, s);
            zoomWindow.Size = s;

            Bitmap bmp = new Bitmap(s.Width, s.Height);
            Graphics g = Graphics.FromImage(bmp);
            GridStyleInfo style = gridControl1[e.RowIndex, e.ColumnIndex];
            float size = style.Font.Size;
            style.Font.Size = 15.5f;
            gridControl1.DrawString(g, e.RowIndex, e.ColumnIndex,
rect, style, true, true);
            g.Dispose();

            this.zoomWindow.Image = bmp;
            this.zoomWindow.BorderStyle = BorderStyle.FixedSingle;
            this.zoomWindow.Visible = true;
            Point pt =
this.gridControl1.ViewLayout.RowColToPoint(e.RowIndex, e.ColumnIndex,
GridCellSizeKind.VisibleSize);
            pt.Y += 60;
            zoomWindow.Location = pt;

            style.Font.Size = size;
        }
    }
}
```

```
        }
    }
else
{
    this.zoomWindow.Visible = false;
    MessageBox.Show("Not a record cell");
}
}
```

**[VB.NET]**

```
Private zoomWindow As System.Windows.Forms.PictureBox

' Code to show zoom window.
Private Sub gridControl1_CellClick(ByVal sender As Object, ByVal e As GridCellEventArgs)
    If e.RowIndex > 0 AndAlso e.ColumnIndex > 0 Then
        If checkBox1.Checked Then
            If (Not zoomWindow.Visible) Then
                Me.zoomWindow.Visible = True
            End If
            Dim p1 As Point = New Point(0, 0)
            Dim s As Size = New
Size(Me.gridControl1.ColumnWidths(e.ColumnIndex) + 10,
Me.gridControl1.RowHeaders(e.RowIndex) + 5)
            s.Width += 50
            s.Height += 30
            Dim rect As Rectangle = New Rectangle(p1, s)
            zoomWindow.Size = s

            Dim bmp As Bitmap = New Bitmap(s.Width, s.Height)
            Dim g As Graphics = Graphics.FromImage(bmp)
            Dim style As GridStyleInfo = gridControl1(e.RowIndex,
e.ColumnIndex)
            Dim size As Single = style.Font.Size
            style.Font.Size = 15.5F
            gridControl1.DrawString(g, e.RowIndex, e.ColumnIndex,
rect, style, True, True)
            g.Dispose()

            Me.zoomWindow.Image = bmp
            Me.zoomWindow.BorderStyle = BorderStyle.FixedSingle
            Me.zoomWindow.Visible = True
            Dim pt As Point =
Me.gridControl1.ViewLayout.RowColToPoint(e.RowIndex, e.ColumnIndex,
GridCellSizeKind.VisibleSize)
            pt.Y += 60
        End If
    End If
End Sub
```

```
zoomWindow.Location = pt

    style.Font.Size = size
End If
Else
    Me.zoomWindow.Visible = False
    MessageBox.Show("Not a record cell")
End If
End Sub
```

Now when you click any cell, it displays a picture box over the cell showing the cell content in a magnified manner.



**Note:** For more details on this feature, refer the following browser sample:

**<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Windows\Samples\2.0\Zoom and Scrolling\Zoom Grid Demo**

#### 4.1.4.32 Tile Image In Grid Cell

Essential **GridControl** supports Tile Image feature in Grid cell.

Set **BackgroundImageMode** property to **GridBackgroundImageMode.TileImage** to add title image in grid cell.

The following code illustrates how to add **Tile Image feature** in Grid cell.

```
[C#]
this.gridControl1[2, 2].BackgroundImageMode =
GridBackgroundImageMode.TileImage;
```

```
[VB]
Me.gridControl1(2, 2).BackgroundImageMode =
GridBackgroundImageMode.TileImage
```

When the code runs, the following image displays.



*Figure 207: Tile Image in grid cell*

## 4.2 Grid Data Bound Grid

The Grid Data Bound Grid has been designed to be used as a grid that is bound to a data source such as an ADO.NET dataset or data table. It can be used with either flat or hierarchical data sources; does not include grouping support. No data values are stored in a `GridDataBoundGrid` object. The Grid Data Bound Grid is more column-centric than the Grid control as it is data bound. Its **GridBoundColumns** collection property will maintain an array of `GridBoundColumn` objects that will hold information like **HeaderText**, **MappingName** and a **GridStyleInfo** object that holds the style properties for the column.

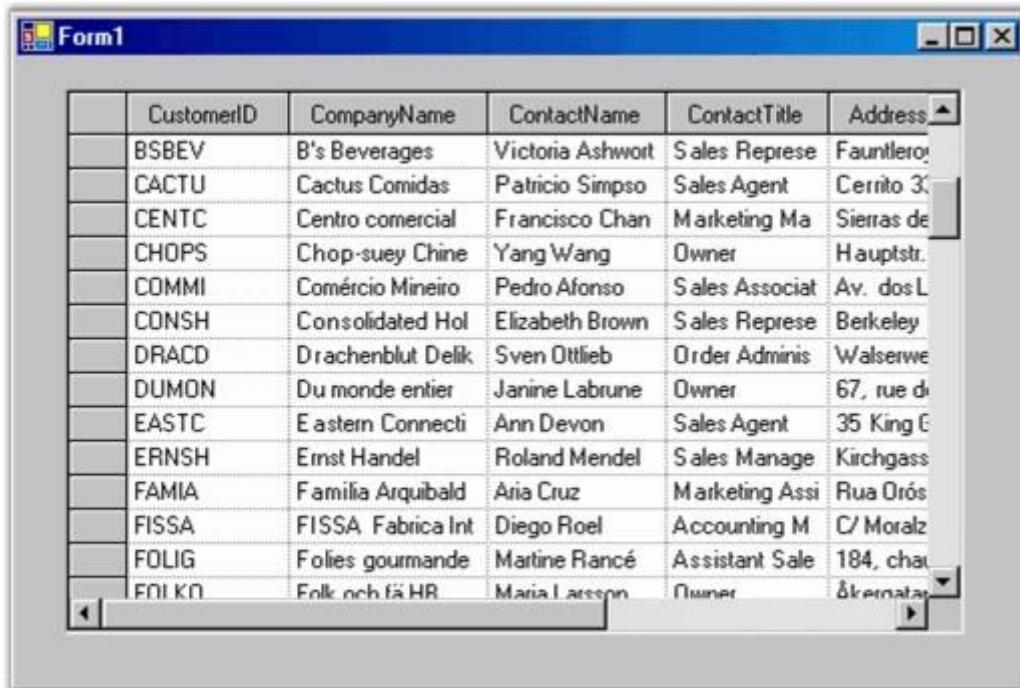


Figure 208: Grid Data Bound Grid showing the NorthWind.Customers Table

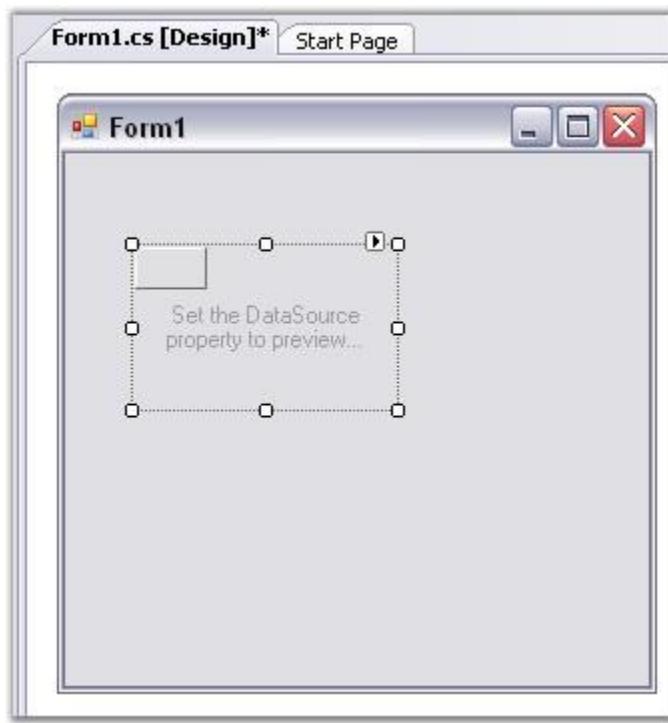
## 4.2.1 Creating Grid Data Bound Grid

This section will provide the step-by-step procedure to create a Grid Data Bound Grid through designer and through programmatical approach in a .NET application.

### 4.2.1.1 Through Designer

With the designer, all you have to do is drag the Grid Data Bound Grid control, resize it and then set the desired properties. The following steps illustrate this.

1. Drag a GridBoundDataGrid object from your toolbox onto the form.



*Figure 209: Grid Data Bound Grid dragged from the toolbox onto the Form*

2. Size and position it.
3. Click the Smart tag, expand Choose DataSource combo box and click Add Project DataSource.

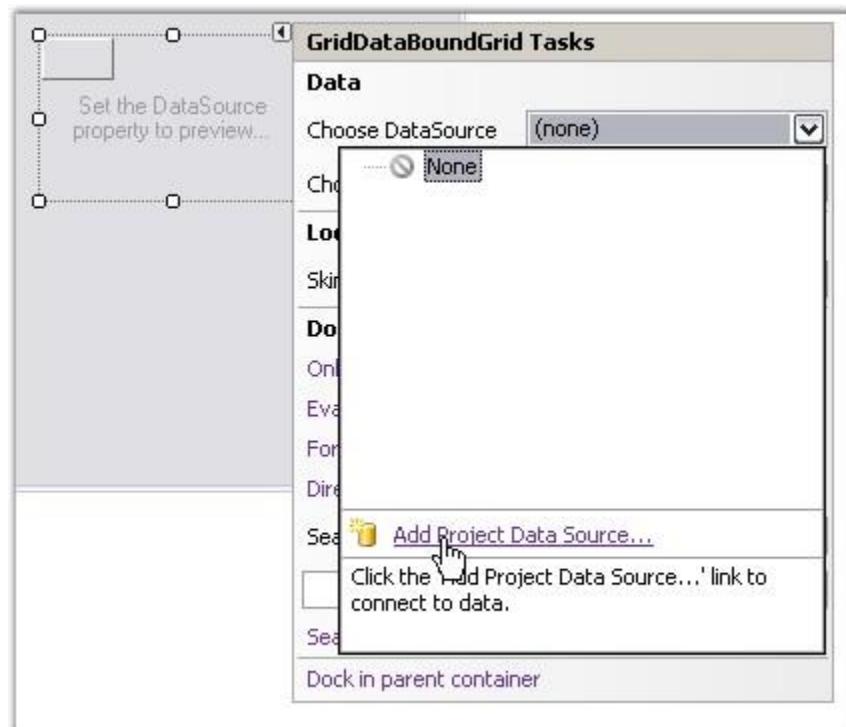


Figure 210: Add Project Data Source Option selected through Smart Tag

4. In the **Data Source Configuration** wizard, select **Database** and click **Next**.



Figure 211: Data Source Type set to "Database" in the Data Source Configuration Wizard Dialog Box

5. Select appropriate data connection and click **Next**. This example uses Northwind database which is available in the below path:

<Sample Install Location>\Syncfusion\EssentialStudio\x.x.x\Common\Data

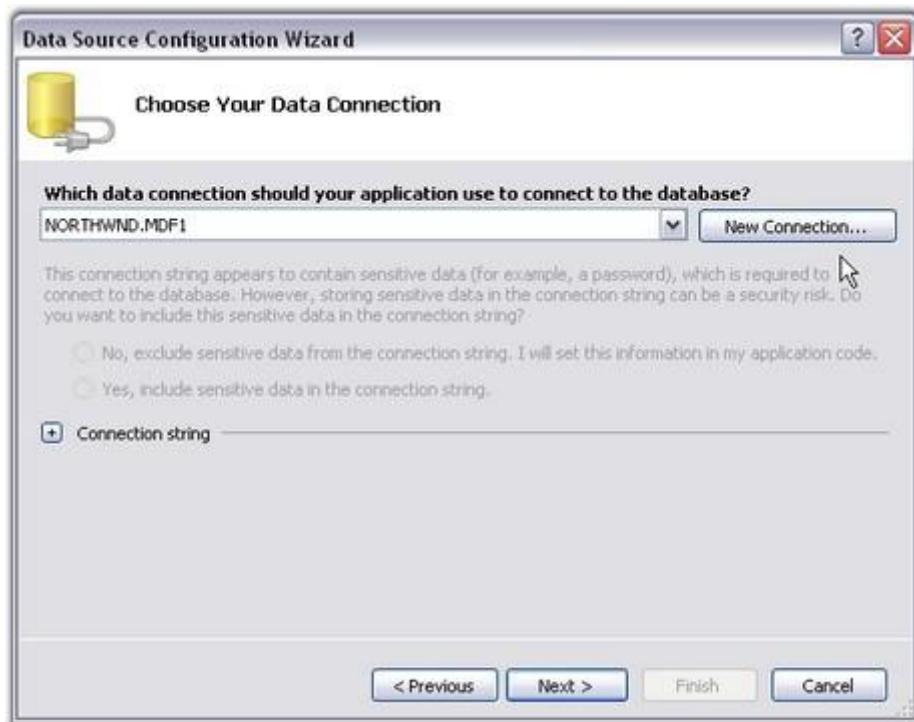
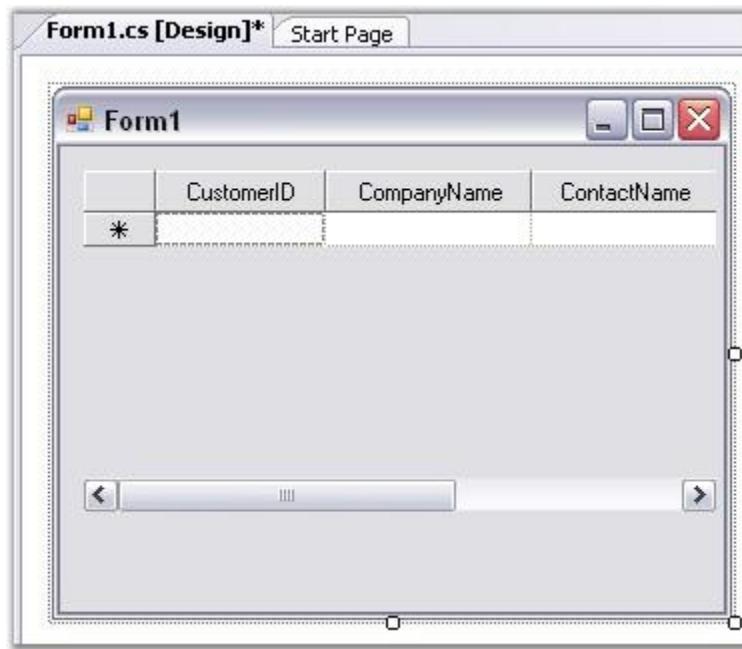


Figure 212: Data Connection Set

6. Select appropriate table and data using this wizard.



*Figure 213: Table and Data Selected*

7. To customize columns, open the **GridBoundColumns** collection by clicking that property in the Grid Data Bound Grid. With this editor, you can determine exactly, which columns of the data source are displayed in the grid. You can also set the **column** properties like **backcolor** and **font**

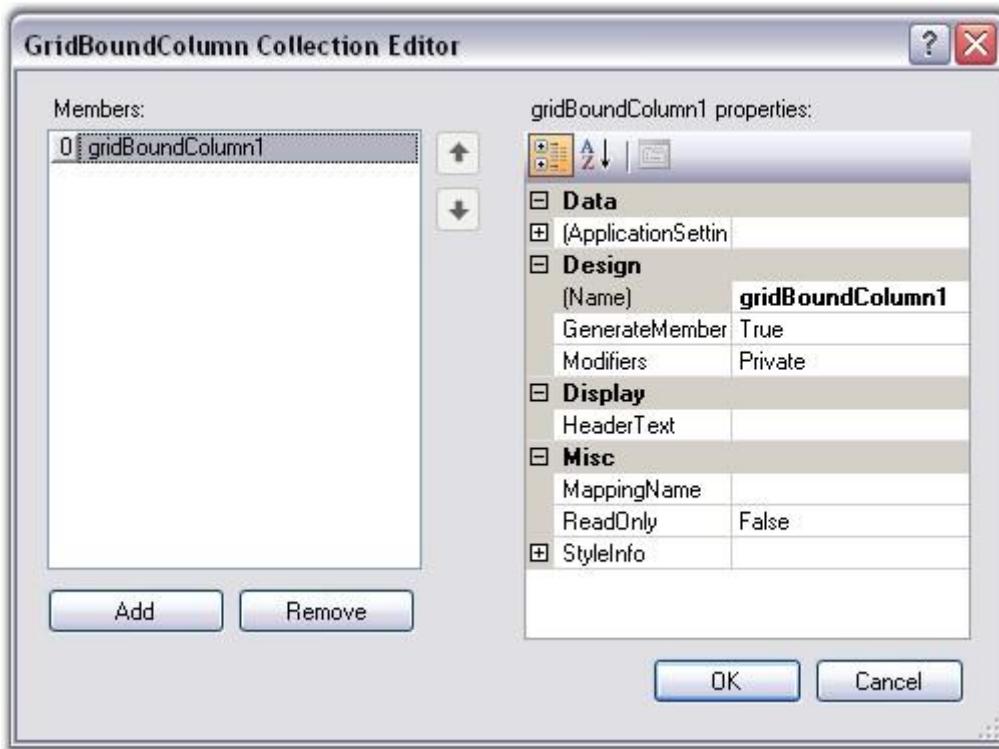


Figure 214: Customizing Columns by using the GridBoundColumn Collection Editor

- Run the application. Following is the output.

	CustomerID	CompanyName	ContactName
▶	ALFKI	Alfreds Futterkiste	Maria
	ANATR	Ana Trujillo Empar	Ana Trujillo
	ANTON	Antonio Moreno Ta	Antonio Moren
	AROUT	Around the Horn	Thomas Hardy
	BERGS	Berglunds snabbk	Christina Bergl
	BLAUS	Blauer See Delikat	Hanna Moos
	BLONP	Blondesddsl père	Frédérique Cib
	BOLID	Bólido Comidas pr	Martín Sommer

Figure 215: GridBoundColumn created Through Designer

Grid Data Bound Grid is added to the windows application and bound to a local data source. For more details, see [Grid Data Bound Grid tutorial](#).

#### 4.2.1.2 Through Code

Here are some code samples that will create a DataTable and bind it to a Grid Data Bound Grid. Once you have a **DataTable** object populated you can use the **GridDataBoundGrid.DataSource** property to implement the binding.

[C#]

```
DataTable myDataTable = new DataTable("MyDataTable");

// Declare the Data Column and Data Row variables.
 DataColumn myDataColumn;
 DataRow myDataRow;

// Create a new Data Column, set the Data Type and Column Name and add
// to the Data Table.
 myDataColumn = new DataColumn();
 myDataColumn.DataType = System.Type.GetType("System.Int32");
 myDataColumn.ColumnName = "id";
 myDataTable.Columns.Add(myDataColumn);

// Create a second column.
 myDataColumn = new DataColumn();
 myDataColumn.DataType = Type.GetType("System.String");
 myDataColumn.ColumnName = "item";
 myDataTable.Columns.Add(myDataColumn);

// Create new Data Row objects and add to the Data Table.
 for (int i = 0; i <= 10; i++)
{
    myDataRow = myDataTable.NewRow();
    myDataRow["id"] = i;
    myDataRow["item"] = "item " + i.ToString();
    myDataTable.Rows.Add(myDataRow);
}
this.GridDataBoundGrid1.DataSource = myDataTable;

// Size the columns.
this.GridDataBoundGrid1.Model.ColWidths[1] = 30;
this.GridDataBoundGrid1.Model.ColWidths[2] = 50;
```

[VB.NET]

```
Dim myDataTable As DataTable = New DataTable("MyDataTable")

' Declare the Data Column and Data Row variables.
Dim myDataColumn As DataColumn
Dim myDataRow As DataRow

' Create a new Data Column, set the Data Type and Column Name and add
to the Data Table.
myDataColumn = New DataColumn()
myDataColumn.DataType = System.Type.GetType("System.Int32")
myDataColumn.ColumnName = "id"
myDataTable.Columns.Add(myDataColumn)

' Create a second column.
myDataColumn = New DataColumn()
myDataColumn.DataType = Type.GetType("System.String")
myDataColumn.ColumnName = "item"
myDataTable.Columns.Add(myDataColumn)

' Create new Data Row objects and add to the Data Table.
Dim i As Integer
For i = 0 To 10
    myDataRow = myDataTable.NewRow
    myDataRow("id") = i
    myDataRow("item") = "item " & i
    myDataTable.Rows.Add(myDataRow)
Next i

Me.gridDataBoundGrid1.DataSource = myDataTable

' Size the columns.
Me.gridDataBoundGrid1.Model.ColWidths(1) = 30
Me.gridDataBoundGrid1.Model.ColWidths(2) = 50
```

## 4.2.2 Concepts and Features

This section discuss the use cases for the Grid Data Bound Grid. The cases range from simple binding to a DataTable through Master-Detail and ends with some hierarchical binding samples. It also includes other topics like filtering, sorting, and accessing data in the grid.

### 4.2.2.1 Binding to an ArrayList

You can bind an **ArrayList** that holds objects with public properties. Given below is an example, which substantiates this point.

1. We first define an object called **Person** with two public properties: **FirstName** and **LastName**.
2. We then create an **ArrayList** holding a collection of these objects.
3. To bind this **ArrayList** to a Grid Data Bound Grid, we set the grid's **DataSource** property after dropping a Grid Data Bound Grid onto the form.

Given below are the code samples for this.

[C#]

```
// Create the Person object class.
public class Person
{
    private string lname;
    private string fname;

    public Person(string fname, string lname)
    {
        this.fname = fname;
        this.lname = lname;
    }

    public string FirstName
    {
        get{return fname;}
        set{fname = value;}
    }
    public string LastName
    {
        get{return lname;}
        set{lname = value;}
    }
}

// Code within your Form Load event handler.
private void Form1_Load(object sender, System.EventArgs e)
{
    ArrayList al = new ArrayList();
    al.Add(new Person("John", "Smith"));
```

```
al.Add(new Person("Mary", "Tucker"));
al.Add(new Person("Sue", "Gaskins"));
al.Add(new Person("John", "Jacobs"));
al.Add(new Person("Sam", "Garfunkel"));
al.Add(new Person("George", "Shepherd"));
al.Add(new Person("Becky", "Dunsford"));

this.gridDataBoundGrid1.DataSource = al;
}
```

**[VB.NET]**

```
' Create the Person object class.
Public Class Person
    Private fname As String
    Private lname As String

    Public Sub New(ByVal fname As String, ByVal lname As String)
        Me.fname = fname
        Me.lname = lname
    End Sub

    Public Property LastName()
        Get
            Return lname
        End Get
        Set(ByVal Value)
            lname = Value
        End Set
    End Property

    Public Property FirstName()
        Get
            Return fname
        End Get
        Set(ByVal Value)
            fname = Value
        End Set
    End Property
End Class

' Code within your Form Load event handler.
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    Dim al As ArrayList = New ArrayList()
```

```
al.Add(New Person("John", "Smith"))
al.Add(New Person("Mary", "Tucker"))
al.Add(New Person("Sue", "Gaskins"))
al.Add(New Person("John", "Jacobs"))
al.Add(New Person("Sam", "Garfunkel"))
al.Add(New Person("George", "Shepherd"))
al.Add(New Person("Becky", "Dunsford"))
Me.GridView1.DataSource = al
End Sub
```

### **See Also**

#### 4.2.2.1.1 ArrayList Class with IBindingList Support

Any changes that you make to the grid will be posted back to the ArrayList. Keep in mind that you cannot add new items to the ArrayList through the grid. Instead, make sure that your data source supports the IBindingList interface and that it implements an appropriate **AddNew** method. The IBindingList support determines whether you can add new items and sort items as well as do searching for other basic aspects of list behavior.



*Figure 216: Grid Data Bound Grid Bound to a Simple ArrayList*

Here is a minimal implementation of an ArrayList-derived class that also supports IBindingList. If you add this class to the above code and in the Form-Load change the ArrayList to PersonList, then you will be able to add new entries to your underlying PersonList data source by using the AppendRow at the bottom of the grid. Notice that the implementation of **IBindingList.AddNew** as well as the **IBindingList.AllowNew** property will indicate that new rows are allowed.

#### 4.2.2.2 Binding to a DataTable

Binding to a DataTable is a very simple and straight-forward process. After defining a DataTable, you must set the **GridDataBoundGrid.DataSource** property to the table. Then you can easily use the Data Tab of your toolbox in Visual Studio to generate DataTables. Here we will add a simple People table using the code to illustrate how you can dynamically create a DataTable.

Creating a DataTable from code is a two-step process. You must first add DataColumn objects to the DataTable.Columns collection and then you must add DataRow objects to the DataTable.Rows collection. Given below is the code that does this. The code will assume that you have dropped a Grid Data Bound Grid onto the form.

[C#]

```
private void Form1_Load(object sender, System.EventArgs e)
{
    this.gridDataBoundGrid1.DataSource = ReturnATable();
}

private DataTable ReturnATable()
{
    DataTable table = new DataTable("People");

    // Add two columns.
    table.Columns.Add(new DataColumn("FirstName"));
    table.Columns.Add(new DataColumn("LastName"));

    // Add some rows.
    DataRow dr = table.NewRow();
    dr["FirstName"] = "John";
    dr["LastName"] = "Smith";
    table.Rows.Add(dr);

    dr = table.NewRow();
    dr["FirstName"] = "Mary";
    dr["LastName"] = "Tucker";
    table.Rows.Add(dr);

    dr = table.NewRow();
    dr["FirstName"] = "Sue";
    dr["LastName"] = "Gaskins";
    table.Rows.Add(dr);

    dr = table.NewRow();
    dr["FirstName"] = "John";
```

```
dr["LastName"] = "Jacobs";
table.Rows.Add(dr);

dr = table.NewRow();
dr["FirstName"] = "Sam";
dr["LastName"] = "Garfunkel";
table.Rows.Add(dr);

dr = table.NewRow();
dr["FirstName"] = "George";
dr["LastName"] = "Shepherd";
table.Rows.Add(dr);

dr = table.NewRow();
dr["FirstName"] = "Becky";
dr["LastName"] = "Dunsford";
table.Rows.Add(dr);

return table;
}
```

**[VB.NET]**

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    Me.GridDataBoundGrid1.DataSource = ReturnATable()
End Sub

Private Function ReturnATable()
    Dim table As DataTable = New DataTable("People")

    ' Add two columns.
    table.Columns.Add(New DataColumn("FirstName"))
    table.Columns.Add(New DataColumn("LastName"))

    ' Add some rows.
    Dim dr As DataRow = table.NewRow()
    dr("FirstName") = "John"
    dr("LastName") = "Smith"
    table.Rows.Add(dr)

    dr = table.NewRow()
    dr("FirstName") = "Mary"
    dr("LastName") = "Tucker"
    table.Rows.Add(dr)
```

```
dr = table.NewRow()
dr("FirstName") = "Sue"
dr("LastName") = "Gaskins"
table.Rows.Add(dr)

dr = table.NewRow()
dr("FirstName") = "John"
dr("LastName") = "Jacobs"
table.Rows.Add(dr)

dr = table.NewRow()
dr("FirstName") = "Sam"
dr("LastName") = "Garfunkel"
table.Rows.Add(dr)

dr = table.NewRow()
dr("FirstName") = "George"
dr("LastName") = "Shepherd"
table.Rows.Add(dr)

dr = table.NewRow()
dr("FirstName") = "Becky"
dr("LastName") = "Dunsford"
table.Rows.Add(dr)

Return table
End Function
```

#### 4.2.2.3 Accessing Values in the Grid Data Bound Grid and in the Data Source

To access values in the Grid Data Bound Grid, use the indexer and retrieve the value from the **GridStyleInfo** object.

[C#]

```
// Get value at (row, col).
object myValue = this.gridDataBoundGrid1[row, col].CellValue;
```

[VB.NET]

```
' Get Value at (row, col).
```

```
Dim myValue as Object = Me.GridDataBoundGrid1(row, col).CellValueget  
value at row, col
```

If you want to retrieve the values that are based on column names, use the methods in the **GridDataBoundGrid.Binder** object to switch the name for a column index.

**[C#]**

```
// Specify the field name.  
int nField = this.gridDataBoundGrid1.Binder.NameToField("FirstName");  
  
// Call Binder.FieldToColIndex method to retrieve the column index for  
// the ColumnName specified.  
int col = this.gridDataBoundGrid1.Binder.FieldToColIndex(nField);  
  
// Get Value at (row, col).  
object myValue = this.gridDataBoundGrid1[row, col].CellValue;
```

**[VB .NET]**

```
' Specify the field name.  
Dim nField As Integer =  
Me.gridDataBoundGrid1.Binder.NameToField("FirstName")  
  
' Call Binder.FieldToColIndex method to retrieve the column index for  
// the ColumnName specified.  
Dim col As Integer =  
Me.gridDataBoundGrid1.Binder.FieldToColIndex(nField)  
  
' Get Value at (row, col).  
Dim myValue As Object = Me.gridDataBoundGrid1(row, col).CellValue
```

#### 4.2.2.4 Using the CurrencyManager

Sometimes the grid itself will not contain all the data in the data table. For example, your data table may have 12 columns in it but, you are able to add only three **GridBoundColumns** to display exactly three of the twelve columns. In this case, the table has nine more columns than the grid. How do you get all the values in the columns that are not in the grid?

Your first inclination might be to grab the row number from the grid and try to use it as an index in the datatable. But, the relationship between the grid's row number and the data table row numbers will break if your grid has been sorted. So, instead of using the row number as an index in the data table, you should use it as an index in the **CurrencyManager** List.

Given below are some code samples.

**[C#]**

```
// Assuming the grid is bound to a Data Table.  
CurrencyManager cm =  
(CurrencyManager)this.grid.BindingContext[this.grid.DataSource,  
this.grid.DataMember];  
DataRowView drv = cm.List[1] as DataRowView;  
  
// Access row 2 of the grid.  
if(drv != null)  
{  
    DataRow dr = drv.Row;  
    Console.WriteLine(dr["FirstName"].ToString());  
}
```

**[VB .NET]**

```
' Assuming grid is bound to a Data Table.  
Dim cm As CurrencyManager =  
CType(Me.grid.BindingContext(Me.grid.DataSource, Me.grid.DataMember),  
CurrencyManager)  
Dim drv As DataRowView = cm.List(1)  
  
' Access row 2 of the grid.  
If Not (drv Is Nothing) Then  
    Dim dr As DataRow = drv.Row  
    Console.WriteLine(dr("FirstName").ToString())  
End If
```

#### 4.2.2.5 Filtering a Grid Data Bound Grid

We will use an example to illustrate the filtering procedure for the grid.

Assume that you are binding a grid to a **DataTable**. In this case, you will have to use the **DataView.RowFilter** property to restrict the rows that appear in your Grid Data Bound Grid. The syntax for the **RowFilter** clause is very similar to the SQL WHERE clause. You will be able to see a full description of this in the .NET Frameworks online help for the  **DataColumn.Expression**, which uses the same syntax.

Here are some samples.

RowFilter String	Result
[City Area] = Center	Shows only rows where the City Area column is center.
rate > 10	Shows only rows where the rate column is greater than 10.
Name LIKE a*	Shows only rows where the Name column begins with *a.
Name LIKE *a	Shows only rows where the Name column ends with *a.

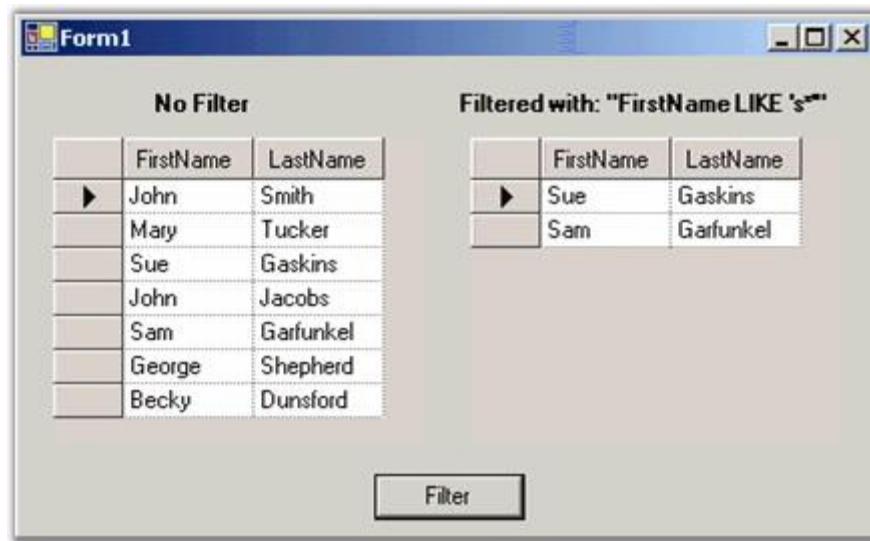


Figure 217: Grid on Right is Filtered Version of Grid on Left

The filtered grid is created by setting the **RowFilter** property of it to default view. If you change the RowFilter property then the grid's contents will change to reflect the new filter.

[C#]

```
// Assuming the grid is bound to a Data Table.  
DataView dv =  
((DataTable)this.gridDataBoundGrid1.DataSource).DefaultView;  
dv.RowFilter = "FirstName LIKE 's*'";
```

**[VB.NET]**

```
' Assuming the grid is bound to a Data Table.  
Dim dv As DataView = CType(Me.gridDataBoundGrid1.DataSource,  
DataTable).DefaultView  
dv.RowFilter = "FirstName LIKE 's*'"
```

You can use the Essential Grid's **GridFilterBar** class to automatically add a row of drop-down cells at the top of a simple (non-hierarchical) Grid Data Bound Grid that can be used to filter the grid to display only rows that match values from the drop-down. For example, when you have a grid with a Grid Filter Bar, if one of your columns is City and you want to see all the rows where City is 'Boston' for example and then you will have to drop the combo box at the top of the City column, and select Boston. The grid will then display only those rows with Boston in the City column. Adding a Grid Filter Bar takes only two lines of code.

**[C#]**

```
// Add a Filter Bar to the data bound grid.  
GridFilterBar filterBar = new  
Syncfusion.Windows.Forms.Grid.GridFilterBar();  
filterBar.WireGrid(gridDataBoundGrid1);
```

**[VB.NET]**

```
' Add a Filter Bar to the data bound grid.  
Dim filterBar As GridFilterBar = New  
Syncfusion.Windows.Forms.Grid.GridFilterBar()  
filterBar.WireGrid(GridDataBoundGrid1)
```

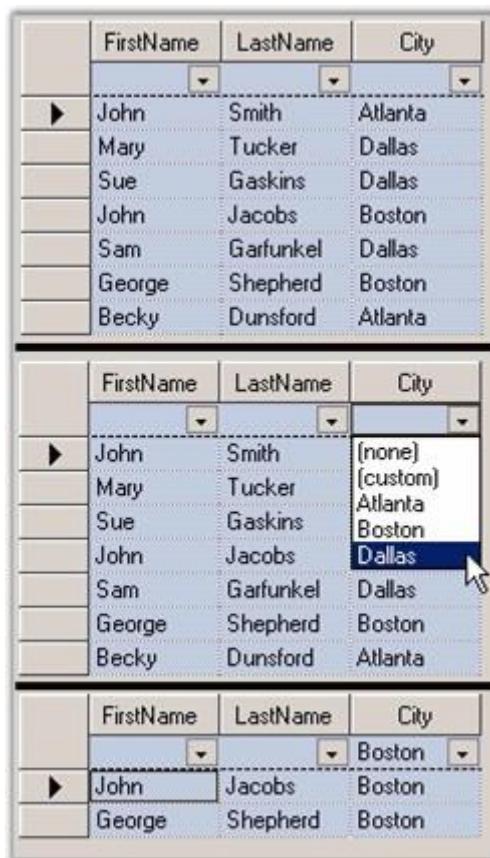


Figure 218: Top Grid is Before Grid Filter Bar; Middle Shows Selecting to Filter on City = Boston; Bottom Grid is Filtered Grid

### Filter By DisplayMember

Grid Data Bound Grid filters the data records by the value member of the columns. This default behavior can be customized in order to accomplish filtering by display member instead. This can be achieved by deriving custom filter from the GridFilterBar class where in you can customize the GetFilterFromRow method to replace the display strings in the filter with the value strings.

The Filter By DisplayMember feature performs this sort of customization and lets you filter the grid data by display member instead of value member.

Following code example illustrates how to enable this filter.

[C#]

```
GridDataBoundGridFilterBarExt filterBar = new
GridDataBoundGridFilterBarExt();
filterBar.WireGrid(gridDataBoundGrid1);
```

**[VB.NET]**

```
Dim filterBar As GridDataBoundGridFilterBarExt = New
GridDataBoundGridFilterBarExt()
filterBar.WireGrid(gridDataBoundGrid1)
```

	ProductID	ProductName	Category	Supplier
▶	1	Chai	(none)	Charlotte Cooper
	2	Chang	(custom)	Charlotte Cooper
	3	Aniseed Syrup	Beverages	Charlotte Cooper
	4	Chef Anton's Cajun S	Condiments	Charlotte Cooper
	5	Chef Anton's Gumbo	Confections	Shelley Burke
	6	Grandma's Boysenbe	Dairy Products	Shelley Burke
	7	Uncle Bob's Organic	Grains/Cereals	Regina Murphy
	8	Northwoods Cranberr	Meat/Poultry	Regina Murphy
	9	Mishi Kobe Niku	Produce	Regina Murphy
	10	Ikura	Seafood	Regina Murphy
	11	Queso Cabrales	Dairy Products	Yoshi Nagase
	12	Queso Manchego La	Dairy Products	Yoshi Nagase

Figure 219: Filtering Data Records in the Grid by the Display Member



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\{Version Number}\Windows\Grid.Windows\Samples\2.0\Data Bound\Filter By DisplayMember Demo

#### 4.2.2.6 GridBoundColumns and Controlling the Column Format

To control the properties of a column in your Grid Data Bound Grid, you must use a **GridBoundColumn** class object. You can also explicitly add a GridBoundColumn object to the **GridDataBoundGrid.GridBoundColumns** collection for each column that you want to see in the grid or you can let the **GridDataBoundGrid.Binder** class generate these columns for you.

Here are the code samples that will explicitly add GridBoundColumns. Note that you can add these GridBoundColumns at design-time provided that you properly set the **MappingName** property for each column.

[C#]

```
private void Form1_Load(object sender, System.EventArgs e)
{
    this.gridDataBoundGrid1.DataSource = ReturnATable();
    this.gridDataBoundGrid1.EnableAddNew = false;
    this.gridDataBoundGrid1.BackColor = Color.FromArgb(0xcc, 0xd4, 0xe6);

    // Create a GridBoundColumn for each displayed column.
    GridBoundColumn gbc = new GridBoundColumn();
    gbc.MappingName = "FirstName";

    // Must set to column mapping name.
    gbc.HeaderText = "Name";

    // Set some style properties.
    gbc.StyleInfo.BackColor = Color.FromArgb(0xC0, 0xC9, 0xdb);
    gbc.StyleInfo.TextColor = Color.Blue;

    // Add the column to the GridBoundColumns collection.
    this.gridDataBoundGrid1.GridBoundColumns.Add(gbc);

    // Repeat for each column.
    gbc = new GridBoundColumn();
    gbc.MappingName = "LastName";
    gbc.HeaderText = "FamilyName";
    gbc.StyleInfo.Font.Bold = true;
    this.gridDataBoundGrid1.GridBoundColumns.Add(gbc);

    gbc = new GridBoundColumn();
    gbc.MappingName = "City";
    gbc.HeaderText = "City";
    this.gridDataBoundGrid1.GridBoundColumns.Add(gbc);

    // Need to initialize the GridBoundColumns so that their settings
    // will replace the currently set values.
    this.gridDataBoundGrid1.Binder.InitializeColumns();

    // Resize the column headers.

    this.gridDataBoundGrid1.Model.ColWidths.ResizeToFit(GridRangeInfo.Row(0),
        GridResizeToFitOptions.NoShrinkSize);
}
```

[VB.NET]

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Me.gridDataBoundGrid1.DataSource = ReturnATable()
    Me.gridDataBoundGrid1.EnableAddNew = False
    Me.gridDataBoundGrid1.BackColor = Color.FromArgb(&HCC, &HD4, &HE6)

    ' Create a GridBoundColumn for each displayed column.
    Dim gbc As New GridBoundColumn()
    gbc.MappingName = "FirstName"

    ' Must set to column mapping name.
    gbc.HeaderText = "Name"

    ' Set some style properties.
    gbc.StyleInfo.BackColor = Color.FromArgb(&HC0, &HC9, &HDB)
    gbc.StyleInfo.TextColor = Color.Blue

    ' Add the column to the GridBoundColumns collection.
    Me.gridDataBoundGrid1.GridBoundColumns.Add(gbc)

    ' Repeat for each column.
    gbc = New GridBoundColumn()
    gbc.MappingName = "LastName"
    gbc.HeaderText = "FamilyName"
    gbc.StyleInfo.Font.Bold = True
    Me.gridDataBoundGrid1.GridBoundColumns.Add(gbc)

    gbc = New GridBoundColumn()
    gbc.MappingName = "City"
    gbc.HeaderText = "City"
    Me.gridDataBoundGrid1.GridBoundColumns.Add(gbc)

    ' Need to initialize the GridBoundColumns so their settings will
    ' replace the currently set values.
    Me.gridDataBoundGrid1.Binder.InitializeColumns()

    ' Resize the column headers.
    Me.gridDataBoundGrid1.Model.ColWidths.ResizeToFit(GridRangeInfo.Row(0), GridResizeToFitOptions.NoShrinkSize)

' Form1_Load
End Sub
```

#### 4.2.2.6.1 Using the `GridDataBoundGrid.Binder` Class

Instead of adding your own `GridBoundColumns`, you can directly reference the internal columns that are generated by the `GridDataBoundGrid.Binder` class when the data source is set in the grid. You can also use these internal columns to set the style info for the columns or `HeaderText` or any of the other properties of the `GridBoundColumn`. Here is a `Form_Load` handler that will display the same grid as above, but using the internal columns.

##### [C#]

```
private void Form1_Load(object sender, System.EventArgs e)
{
    this.gridDataBoundGrid1.DataSource = ReturnATable();
    this.gridDataBoundGrid1.EnableAddNew = false;
    this.gridDataBoundGrid1.BackColor = Color.FromArgb(0xcc, 0xd4, 0xe6);

    // Set the first column.
    GridModelDataBinder binder = this.gridDataBoundGrid1.Binder;
    GridBoundColumn gbc = binder.InternalColumns["FirstName"];
    gbc.HeaderText = "Name";
    gbc.StyleInfo.BackColor = Color.FromArgb(0xC0, 0xC9, 0xdb);
    gbc.StyleInfo.TextColor = Color.Blue;

    // Set the second column.
    gbc = binder.InternalColumns["LastName"];
    gbc.HeaderText = "FamilyName";
    gbc.StyleInfo.Font.Bold = true;

    // Just use the default third column... no changes.
    // Resize the column headers.

    this.gridDataBoundGrid1.Model.ColWidths.ResizeToFit(GridRangeInfo.Row(0),
    GridResizeToFitOptions.NoShrinkSize);
}
```

##### [VB .NET]

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As
System.EventArgs)
    Me.gridDataBoundGrid1.DataSource = ReturnATable()
    Me.gridDataBoundGrid1.EnableAddNew = False
    Me.gridDataBoundGrid1.BackColor = Color.FromArgb(&HCC, &HD4, &HE6)

    ' Set the first column.
```

```
Dim binder As GridModelDataBinder = Me.gridDataBoundGrid1.Binder
Dim gbc As GridBoundColumn = binder.InternalColumns("FirstName")
gbc.HeaderText = "Name"
gbc.StyleInfo.BackColor = Color.FromArgb(&HC0, &HC9, &HDB)
gbc.StyleInfo.TextColor = Color.Blue

' Set the second column.
gbc = binder.InternalColumns("LastName")
gbc.HeaderText = "FamilyName"
gbc.StyleInfo.Font.Bold = True

' Just use the default third column... no changes.
' Resize the column headers.
Me.gridDataBoundGrid1.Model.ColWidths.ResizeToFit(GridRangeInfo.Row(0), GridResizeToFitOptions.NoShrinkSize)

' Form1_Load
End Sub
```

#### 4.2.2.7 Changing Column Order in a Grid Data Bound Grid

The simplest way to change the column order in a Grid Data Bound Grid is to use the **GridDataBoundGrid.Model.Cols.MoveRange** method. This method will rearrange the columns that are based on from and to and count the parameters passed into it.

##### [C#]

```
// Move columns 4 and 5, to column 1.
this.gridDataBoundGrid1.Model.Cols.MoveRange(4, 2, 1);
```

##### [VB .NET]

```
' Move columns 4 and 5, to column 1.
Me.GridDataBoundGrid1.Model.Cols.MoveRange(4, 2, 1)
```

#### 4.2.2.8 Using a Master-Details Relation

To define a simple Master-Details relation, you must have two tables. The first is a Master table that has a column whose values are also included in a second table, referred to as the Details table. You must display these two tables in two grids, which are called as the Master grid and the Details grid. As you click a row in the Master grid, the rows displayed in the Details Grid will be restricted to only those rows whose common value matches the value in the selected Master grid row.

Here is a screen shot showing you a Master-Detail grid pair using the NorthWind Customers table as the Master table, and the NorthWind Orders table as the details table. As you click on a customer in the customers grid, the orders for that customer will appear in the second grid.

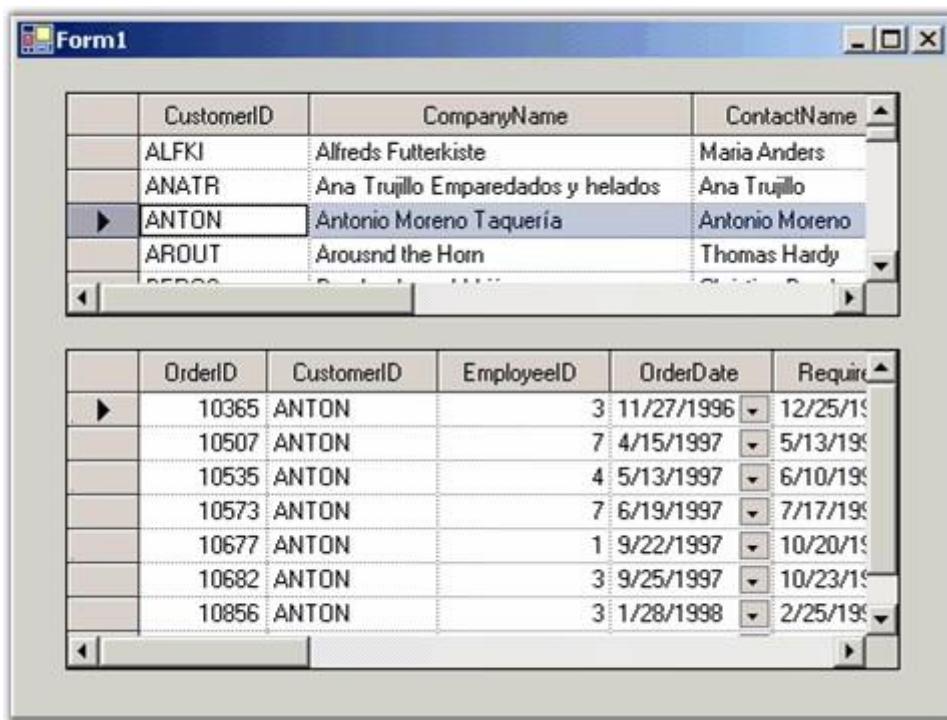


Figure 220: Top Grid is Master Listing All Customers, Bottom Table is Details, Displaying All Orders for Selected Customer

Here is the code that implements this Master-Detail form. In the designer, two DataAdapters (one for each table), Customers and Orders are added to the form. A **DataSet** will be generated as well. Also, the two Grid Data Bound Grids are positioned on the form. The Form\_Load event listed below will set up all the data binding between the DataSet that is holding the two tables and the two grids.

```
[C#]

private void Form1_Load(object sender, System.EventArgs e)
{
    // Fill the Data Set with two tables.
```

```
this.sqlDataAdapter1.Fill(this.dataSet11.Customers);
this.sqlDataAdapter2.Fill(this.dataSet11.Orders);

// Add a Data Relation to the Data Set.
DataRelation dr = new DataRelation("CustomersToOrders",
this.dataSet11.Customers.Columns["CustomerID"],
this.dataSet11.Orders.Columns["CustomerID"]);
this.dataSet11.Relations.Add(dr);

// Set up the data sources.
this.masterGrid.DataSource = this.dataSet11.Tables["Customers"];
this.detailsGrid.DataSource = Me.DataSet11.Tables["Customers"];
this.detailsGridDataMember = "CustomersToOrders";
}
```

**[VB.NET]**

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    ' Fill the Data Set with two tables.
    Me.SqlDataAdapter1.Fill(Me.DataSet11.Customers)
    Me.SqlDataAdapter2.Fill(Me.DataSet11.Orders)

    ' Add a Data Relation to the Data Set.
    Dim dr As DataRelation = New DataRelation("CustomersToOrders",
    Me.DataSet11.Customers.Columns("CustomerID"),
    Me.DataSet11.Orders.Columns("CustomerID"))
    Me.DataSet11.Relations.Add(dr)

    ' Set up the data sources.
    Me.masterGrid.DataSource = Me.DataSet11.Tables("Customers")
    Me.detailsGrid.DataSource = Me.DataSet11.Tables("Customers")
    Me.detailsGridDataMember = "CustomersToOrders"
End Sub
```

#### 4.2.2.9 Foreign Key Columns: Showing One Value but Saving Another

Very often a table will have a column that displays an ID key defined in another table. In your grid, you may like to have this foreign key mapped to some meaningful value, which is referenced from a different column in this other table. The key column in the foreign table is referred to as the **ValueMember** and the meaningful column is referred to as the **DisplayMember**.

Essential Grid will handle this work for you and it can all be done through the designer. For the example discussed below we use code to handle most of the steps.

In the designer, drag two of the Grid Data Bound Grids onto a form. Use one grid to show the foreign key combobox and the other to show the raw data for the primary table. Once the grids are in place, the code, which is given below will create the tables for this sample and then the code in the Form\_Load will hook up the foreign key combobox. In our sample, we have set the combobox button to display only the current row.

[C#]

```
private void Form1_Load(object sender, System.EventArgs e)
{
    this.gridDataBoundGrid1.DataSource = PrimaryTable();
    this.gridDataBoundGrid1.EnableAddNew = false;

    // Column 2 of the grid is the foreign key combo.
    GridBoundColumn gbc =
this.gridDataBoundGrid1.Binder.InternalColumns[1];
    gbc.StyleInfo.CellType = "ComboBox";
    gbc.StyleInfo.DataSource = ForeignKeyTable();
    gbc.StyleInfo.DisplayMember = "Name";
    gbc.StyleInfo.ValueMember = "CustID";
    gbc.StyleInfo>ShowButtons = GridShowButtons.Show.CurrentRow;
    gbc.StyleInfo.HorizontalAlignment = GridHorizontalAlignment.Left;

    // Display the primary table in a second grid without the foreign
key column.
    this.gridDataBoundGrid2.DataSource =
this.gridDataBoundGrid1.DataSource;
    this.gridDataBoundGrid2.EnableAddNew = false;
    this.gridDataBoundGrid2.Enabled = false;
}

private DataTable PrimaryTable()
{
    DataTable dt = new DataTable("PrimaryTable");
    dt.Columns.Add(new DataColumn("ID", typeof(int)));
    dt.Columns.Add(new DataColumn("CustID", typeof(int)));
    dt.Columns.Add(new DataColumn("Address"));
    dt.Columns.Add(new DataColumn("City"));
    for(int i = 0; i < 10; ++i)
    {
        DataRow dr = dt.NewRow();
        dr[0] = i;
    }
}
```

```
        dr[1] = i % 4;
        dr[2] = string.Format("address{0}", i);
        dr[3] = string.Format("city{0}", i);
        dt.Rows.Add(dr);
    }
    return dt;
}

private DataTable ForeignKeyTable()
{
    // Two columns CustID (Value Member) and Name (Display Member).
    DataTable dt = new DataTable("ForeignKeyTable");
    dt.Columns.Add(new DataColumn("CustID", typeof(int)));
    dt.Columns.Add(new DataColumn("Name"));
    for(int i = 0; i < 4; ++i)
    {
        DataRow dr = dt.NewRow();
        dr[0] = i;
        dr[1] = string.Format("Name{0}", i);
        dt.Rows.Add(dr);
    }
    return dt;
}
```

**[VB.NET]**

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As
System.EventArgs)
    Me.gridDataBoundGrid1.DataSource = PrimaryTable()
    Me.gridDataBoundGrid1.EnableAddNew = False

    ' Col 2 of the grid is the foreign key combo.
    Dim gbc As GridBoundColumn =
    Me.gridDataBoundGrid1.Binder.InternalColumns(1)
    gbc.StyleInfo.CellType = "ComboBox"
    gbc.StyleInfo.DataSource = ForeignKeyTable()
    gbc.StyleInfo.DisplayMember = "Name"
    gbc.StyleInfo.ValueMember = "CustID"
    gbc.StyleInfo>ShowButtons = GridShowButtons.Show.CurrentRow
    gbc.StyleInfo.HorizontalAlignment = GridHorizontalAlignment.Left

    ' Just display the primary table in a second grid without the
    ' foreign key column.
    Me.gridDataBoundGrid2.DataSource = Me.gridDataBoundGrid1.DataSource
    Me.gridDataBoundGrid2.EnableAddNew = False
```

```

' Don't allow clicking it.
Me.gridDataBoundGrid2.Enabled = False

' Form1_Load
End Sub

Private Function PrimaryTable() As DataTable
    Dim dt As New DataTable("PrimaryTable")
    dt.Columns.Add(New DataColumn("ID", GetType(Integer)))
    dt.Columns.Add(New DataColumn("CustID", GetType(Integer)))
    dt.Columns.Add(New DataColumn("Address"))
    dt.Columns.Add(New DataColumn("City"))

    Dim i As Integer
    While i < 10
        Dim dr As DataRow = dt.NewRow()
        dr(0) = i
        dr(1) = i Mod 4
        dr(2) = String.Format("address{0}", i)
        dr(3) = String.Format("city{0}", i)
        dt.Rows.Add(dr)
        i += 1
    End While
    Return dt
End Function

' Primary Table
End Function

Private Function ForeignKeyTable() As DataTable
    ' Two columns CustID (Value Member) and Name (Display Member).
    Dim dt As New DataTable("ForeignKeyTable")
    dt.Columns.Add(New DataColumn("CustID", GetType(Integer)))
    dt.Columns.Add(New DataColumn("Name"))

    Dim i As Integer
    While i < 4
        Dim dr As DataRow = dt.NewRow()
        dr(0) = i
        dr(1) = String.Format("Name{0}", i)
        dt.Rows.Add(dr)
        i = i + 1
    End While
    Return dt
End Function

' Foreign Key Table

```

```
End Function
```

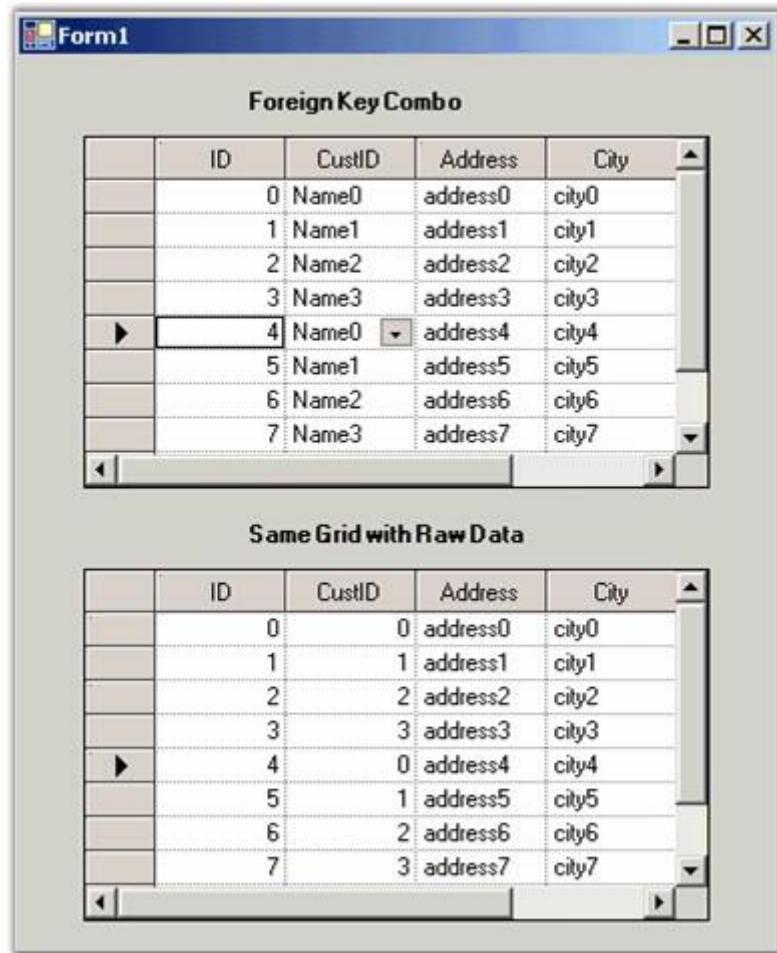


Figure 221: CustID Column of the Top Row Displays Name Field Instead of Raw CustID Field

#### 4.2.2.10 Sorting

Sorting feature available in Grid Data Bound Grid control allows the user to arrange items in a sequence and/or in different sets. **SortBehaviour** property under the control allows you to sort a column in a data bound grid when the column header cell is clicked. Implementation of this property will rearrange the cell data in the clicked column. This property can be set either using code or designer. By default, it is set to perform sorting on a double-click.

**GridSortBehavior** is an enumeration that defines the sorting behavior options/values.

Following is the list of options/values that can be assigned to the SortBehaviour property:

- **SingleClick**: Sort column when user clicks once.
- **DoubleClick**: Sort column when user double-clicks.
- **None**: No sorting when user clicks.

The following code example illustrates sorting of columns on a single click.

[C#]

```
this.gridDataBoundGrid1.SortBehavior = GridSortBehavior.SingleClick;
```

[VB .NET]

```
Me.gridDataBoundGrid1.SortBehavior = GridSortBehavior.SingleClick
```

	Name	State	Zip
▶	Name00006	North Caroli	1
	Name00015	South Caroli	5
	Name00007	Washington	10
	Name00001	South Caroli	11
	Name00003	North Caroli	11
	Name00017	Nevada	14
	Name00018	Washington	21
	Name00013	Nevada	26
	Name00019	Washington	30
	Name00005	Washington	31
	Name00011	North Caroli	31
	Name00009	South Caroli	42
	Name00014	South Caroli	42
	Name00000	South Caroli	45
	Name00012	Washington	48
	Name00004	South Caroli	51
	Name00016	Nevada	52
	Name00008	South Caroli	61
	Name00010	North Caroli	62

Figure 222: Sorting Column

#### 4.2.2.11 Sort by DisplayMember

By default, sorting is done in a Grid Data Bound Grid through the **IBindingList**.



**Note:** *IBindingList interface provides the features required to support both complex and simple scenarios when binding to a data source.*

**Sort** method relies on the data source for the grid and by default, sorting is done based on the value members present in the data source and not based on the display member. We can implement Sort By DisplayMember feature in Grid Data Bound Grid. The code for foreign key column can be added to the View of the data table so that the sort behavior can be redirected to use the foreign key column linked to the combo box column, when the user sorts the combo box column.

#### Example:

The following code example implements a solution for sorting a column by its display member instead of its value member. Here the foreign key column is added to the View of the data to redirect the sort behavior to use the foreign key column.

To accomplish this, two handlers—the **CellClick** event and the **QueryCellInfo** event have been used. In the CellClick event, the display member is set to the existing mapping name in the sortName (which will be the value member) so that the sorting is done by display member.

#### [C#]

```
string sortName = column.MappingName;
if (column.MappingName == "SupplierID")
    sortName = "CompanyName";
else if (column.MappingName == "CategoryID")
    sortName = "CategoryName";
```

#### [VB .NET]

```
Dim sortName As String = column.MappingName
If column.MappingName = "SupplierID" Then
    sortName = "CompanyName"
ElseIf column.MappingName = "CategoryID" Then
    sortName = "CategoryName"
End If
```

A DataView is created by using the **List** property under the **CurrencyManager** class.

#### [C#]

```
CurrencyManager cm = BindingContext[grid.DataSource, grid.DataMember]
as CurrencyManager;
```

```
DataGridView dv = cm.List as DataGridView;
```

**[VB.NET]**

```
Dim cm As CurrencyManager = TryCast(BindingContext(Grid.DataSource,  
Grid.DataMember), CurrencyManager)  
Dim dv As DataView = TryCast(cm.List, DataView)
```

The DataView sort is applied to this with the sortName.

**[C#]**

```
if (dv.Sort == sortName)  
{  
    dv.Sort = sortName + " DESC";  
}  
else  
    dv.Sort = sortName;
```

**[VB.NET]**

```
If dv.Sort = sortName Then  
    dv.Sort = sortName & " DESC"  
Else  
    dv.Sort = sortName  
End If
```



**Note:** *CurrencyManager manages a list of binding objects when the data source uses the IBindingList interface.*

In the QueryCellInfo handler, the sorting icon is drawn with respect to sorting

**[C#]**

```
if (dv.Sort == sortName)  
    e.Style.Tag = ListSortDirection.Ascending;  
else if (dv.Sort == sortName + " DESC")  
    e.Style.Tag = ListSortDirection.Descending;
```

**[VB.NET]**

```
If dv.Sort = sortName Then  
    e.Style.Tag = ListSortDirection.Ascending
```

```

ElseIf dv.Sort = sortName & " DESC" Then
    e.Style.Tag = ListSortDirection.Descending
End If

```

**Sort By DisplayMember Demo**

	ProductID	ProductName	Category	Supplier
▶	38	Côte de Blaye	Beverages	Aux joyeux ecclésiastiques
	39	Chartreuse verte	Beverages	Aux joyeux ecclésiastiques
	34	Sasquatch Ale	Beverages	Bigfoot Breweries
	35	Steeleye Stout	Beverages	Bigfoot Breweries
	67	Laughing Lumberjack Lager	Beverages	Bigfoot Breweries
	11	Queso Cabrales	Dairy Produ	Cooperativa de Quesos 'Las
	12	Queso Manchego La Pastora	Dairy Produ	Cooperativa de Quesos 'Las
	58	Escargots de Bourgogne	Seafood	Escargots Nouveaux
	1	Chai	Beverages	Exotic Liquids
	2	Chang	Beverages	Exotic Liquids
	3	Aniseed Syrup	Condiments	Exotic Liquids
	62	Tarte au sucre	Confections	Forêts d'éables
	31	Gorgonzola Telino	Dairy Produ	Formaggi Fortini s.r.l.
	32	Mascarpone Fabioli	Dairy Produ	Formaggi Fortini s.r.l.
	72	Mozzarella di Giovanni	Dairy Produ	Formaggi Fortini s.r.l.
	59	Raclette Courdavault	Dairy Produ	Gai pâturage
	60	Camembert Pierrot	Dairy Produ	Gai pâturage

Figure 223: Sort by Display Member

A sample demonstrating this feature is available under the following sample installation path.

`<Install Location>\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Windows\Samples\2.0\Data Bound\Sort By DisplayMember Demo`

#### 4.2.2.12 Data Relations

This section illustrates the following topics.

##### 4.2.2.12.1 Nested Drop-down Grids

Nested Drop-down grids are used to represent multi-level data in a grid. For example, if a bank wants to load all the accounts of an enrolled user in a grid control for a financial project, and some of the accounts have subaccounts with options to be selected under each subaccount, which need to be loaded/ shown as a subelement to that account, Nested Drop-down grids can be used to represent the data. The data can be distributed in parent (primary) grid, child grid, and so on. Grid Data Bound Grid control can display hierarchical data using Nested Drop-down grids.

**Example:**

In the code example below, the parent (primary) grid is the 'Customers' table from the NorthWind database. On clicking a row of this table, the 'Orders' table will be displayed in a new grid providing details on orders placed by the customers. On clicking any of the rows in Orders table, another grid named 'Order\_Details' table is displayed providing details on the order details of the selected row in the Orders table.

This example has a derived GridDataBoundGrid class called the **GridHierDataBoundGrid** used for all the grids to be displayed. In the constructor for this class, the tables for parent and child are to be passed.

**[C#]**

```
// Parent table to Child table.  
// Create the outermost grid for the customers table-uses  
GridHierDataBoundGrid class.  
this.customerGrid1 = new GridHierDataBoundGrid(this,  
this.dataSet11.Customers,  
this.dataSet11.Orders, this.orderGrid2, new  
QueryFilterStringEventHandler(ProvideOrdersFilterStrings),  
new QueryFormatGridEventHandler(ProvideOrderFormat), true);
```

**[VB .NET]**

```
' Parent table to Child table.  
' Create the outermost grid for the customers table-uses  
GridHierDataBoundGrid class.  
Me.customerGrid1 = New GridHierDataBoundGrid(Me,  
Me.dataSet11.Customers, Me.dataSet11.Orders, Me.orderGrid2, New  
QueryFilterStringEventHandler(AddressOf ProvideOrdersFilterStrings),  
New QueryFormatGridEventHandler(AddressOf ProvideOrderFormat), True)
```

Finally, to specify a relationship between a parent table and a child table, an event handler must be passed for the **QueryFilterString** event. The event should specify the FilterString that defines the relationship between the parent table and the child table.

**[C#]**

```
// Parent table to Child table.
private void ProvideOrdersFilterStrings(object sender,
QueryFilterStringEventArgs e)
{
    if (this.customerGrid1.Model[e.Row, e.Column + 1].Text != "")

        // Add 1 to get to Customer ID.
        e.FilterString = string.Format("CustomerID = '{0}'",
            this.customerGrid1.Model[e.Row, e.Column + 1].Text);
}
```

**[VB.NET]**

```
' Parent table to Child table.
Private Sub ProvideOrdersFilterStrings(ByVal sender As Object, ByVal e
As QueryFilterStringEventArgs)
    If Me.customerGrid1.Model(e.Row, e.Column + 1).Text <> "" Then

        ' Add 1 to get to Customer ID.
        e.FilterString = String.Format("CustomerID = '{0}'",
            Me.customerGrid1.Model(e.Row, e.Column + 1).Text)
    End If
End Sub
```

The screenshot shows a Windows application window containing two grids. The top grid has columns for CustomerID, CompanyName, ContactName, ContactTitle, and Address. The bottom grid, which is nested within the rows of the top grid, has columns for OrderID, CustomerID, EmployeeID, OrderDate, RequiredDate, ShippedDate, ShipVia, and Freight. The data in both grids is from the Northwind database.

	CustomerID	CompanyName	ContactName	ContactTitle	Address			
-	ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57			
▼	ANSTR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222			
▼	10308	ANSTR	King	9/18/1996	10/16/1996	9/24/1996	3	1.
▼	10625	ANSTR	Leverling	8/8/1997	9/5/1997	8/14/1997	1	43.
▼	10759	ANSTR	Leverling	11/28/1997	12/26/1997	12/12/1997	3	11.
▼	10926	ANSTR	Penney	3/1/1998	4/1/1998	3/11/1998	2	29.
▼	BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Bouchers			
▼	BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd.			
▼	BSBEV	B's Beverages	Victoria Ashworth	Sales Representative	Fauntleroy Circus			
▼	CACTU	Cactus Comidas para llevar	Patricia Simpson	Sales Agent	Cerrito 333			
▼	CENTC	Centro comercial Moctezuma	Francisco Chang	Marketing Manager	Sierras de Granada 9993			

Figure 224: Nested Drop-down Grids

A sample demonstrating this feature is available under the following sample installation path.

**<Install Location>\Syncfusion\EssentialStudio\Version  
Number\Windows\Grid.Windows\Samples\2.0\Data Bound\GDBG Drop Grid Demo**

#### 4.2.2.12.2 Multiple Nested Relations

Grid Data Bound Grid control supports multiple nested relations. A relation can be added in the data source and the data source can be set to the Grid Data Bound Grid. Then the name of the relation can be passed through the **Grid.Binder.AddRelation** function to show a hierarchical pattern.

##### Example:

This following code example illustrates the display of a DataSet with multiple nested relations. The sample displays the NorthWind's 'Category', 'Products' and the 'Orders\_Details' table, and allows you to expand and collapse the order details for each order and the products for each category. After adding a relation in the dataset and setting the DataSource to the grid, the name of the relation is passed through the Grid.Binder.AddRelation function in order to show a hierarchical pattern.

##### [C#]

```
GridHierarchyLevel hlCategory_Products =  
gridBinder.AddRelation("Category_Products");  
GridHierarchyLevel hlProducts_OrderDetails =  
gridBinder.AddRelation("Products_OrderDetails");
```

##### [VB .NET]

```
Dim hlCategory_Products As GridHierarchyLevel =  
gridBinder.AddRelation("Category_Products")  
Dim hlProducts_OrderDetails As GridHierarchyLevel =  
gridBinder.AddRelation("Products_OrderDetails")
```

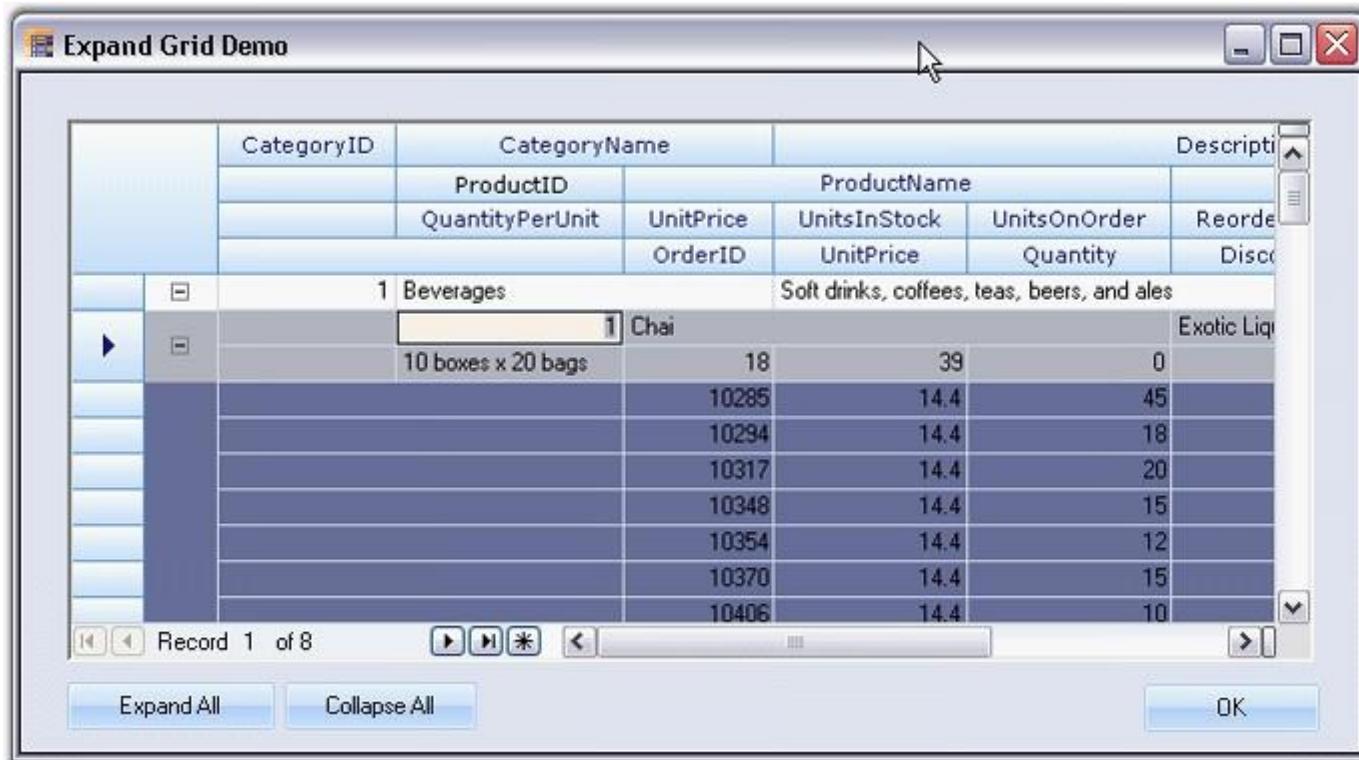


Figure 225: Expand Grid

A sample demonstrating this feature is available under the following sample installation path.

**<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Windows\Samples\2.0\Data Bound\Hierarchy\Expand Grid Demo**

#### 4.2.2.12.3 Hierarchical Grid with Tree Lines

Grid Data Bound Grid supports display of hierarchical grid with tree lines. This can be achieved by setting the **ShowTreeLines** property to *true*.

[C#]

```
this.gridDataBoundGrid1.ShowTreeLines = true;
```

[VB .NET]

```
Me.gridDataBoundGrid1.ShowTreeLines = True
```

With the ShowTreeLines property set to true, there is no separate column allotted for the plus/minus buttons, instead the indented text in the first column can be seen.

	ParentName	parentID	ParentDec
	ChildName	childID	ParentID
	GrandChildName	grandChildI	ChildID
▶	GreatGrandChildN	greatGrand	GrandChildI
▶	GreatGreatGrandC	greatGreat	GreatGrandC
▶	parentName0	0	desc0
▶	ChildName5	5	0
▶	ChildName7	7	0
▶	ChildName17	17	0
▶	parentName1	1	desc1
▶	ChildName9	9	1
▶	ChildName11	11	1
▶	ChildName18	18	1
▶	parentName2	2	desc2
▶	parentName3	3	desc3
▶	parentName4	4	desc4

Figure 226: Hierarchical Grid with Tree Lines

A sample demonstrating this feature is available under the following sample installation path.

**<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Windows\Samples\2.0\Data Bound\Hierarchy\GDBG Tree Lines Demo**

#### 4.2.2.12.4 ExpandAll and CollapseAll Methods

##### 1. ExpandAll

Using this method will enable the user to view the expanded nodes in the Grid Data Bound Grid, i.e., the parent, child and subsequent sublevel nodes can be viewed.

The following code illustrates how to set this method for Grid Data Bound Grid:

##### a. Using C#

```
[C#]
```

```
this.gridDataBoundGrid1.ExpandAll();
```

b. Using VB.NET

[VB.NET]

```
Me.gridDataBoundGrid1.ExpandAll()
```

ParentName	parentID	ParentDec
ChildName	childID	ParentID
GrandChildName	grandChildI	ChildID
GreatGrandChildN	greatGrand	GrandChildI
GreatGreatGrandC	greatGreat	GreatGrandC
parentName0	0	desc0
└ ChildName3	3	0
└ GrandChild	9	3
└ GreatGra	48	9
└ GreatGra	84	9
└ GreatG	176	84
└ GrandChild	16	3
└ GreatGra	53	16
└ GreatG	112	53
└ GreatG	180	53
└ GreatG	203	53
└ GreatGra	70	16

Figure 227: Grid Data Bound Grid with Expanded Nodes

## 2. CollapseAll

Using this method will enable the user to view the collapsed nodes in the Grid Data Bound Grid, i.e., only the parent node can be viewed with its corresponding expansion icon.

The following code illustrates how to set this method for Grid Data Bound Grid:

a. Using C#

[C#]

```
this.gridDataBoundGrid1.CollapseAll();
```

b. Using VB.NET

[VB.NET]

```
Me.gridDataBoundGrid1.CollapseAll()
```

ParentName	parentID	ParentDec
ChildName	childID	ParentID
GrandChildName	grandChildI	ChildID
GreatGrandChildN	greatGrand	GrandChildI
GreatGreatGrandC	greatGreat	GreatGrandC
+ parentName0	0	desc0
+ parentName1	1	desc1
+ parentName2	2	desc2
+ parentName3	3	desc3
+ parentName4	4	desc4

Figure 228: Grid Data Bound Grid with Collapsed Nodes

#### 4.2.2.13 Multi Row Record

The Grid Data Bound Grid has support for displaying a single record in multiple rows.

CustomerID	CompanyName		ContactTitle	ContactName	
	Address		City		
	PostalCode	Country			
ALFKI	Alfreds Futterkiste		Sales Representative	Maria Anders	
	Obere Str. 57		Berlin		
12209	Germany		030-0074321	030-0076545	
ANATR	Ana Trujillo Emparedados y helados		Owner	Ana Trujillo	
	Avda. de la Constitución 2222		México D.F.		
05021	Mexico		(5) 555-4729	(5) 555-3745	
ANTON	Antonio Moreno Taquería		Owner	Antonio Moreno	
	Mataderos 2312		México D.F.		
05023	Mexico		(5) 555-3932		
AROUT	Around the Horn		Sales Representative	Thomas Hardy	
	120 Hanover Sq.		London		
WA11DP	UK		(171) 555-7788	(171) 555-6750	
SEEDOO	Sakar International		General Trading	Customer Support	

*Figure 229: Multi Row Record*



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Windows\Samples\2.0\Data Bound\Multi Row Record Demo

### Example

Using the following code example, you can switch the display of the records from the NorthWind's Customers table between displaying a single row per record and multiple rows per record. The Binder.LayoutColumns function can be used to break the records into multiple rows. The record can be broken by inserting a "." in the LayoutColumns() function of the GridHierarchyLevel class.

#### [C#]

```
GridModel gridModel = gridDataBoundGrid1.Model;
GridModelDataBinder binder = gridDataBoundGrid1.Binder;

// "." indicates a new row.
binder.LayoutColumns(new string[] {"CustomerID", "CompanyName",
"ContactTitle", "ContactName", ".", "Address", "City", ".",
"PostalCode", "Country", "Phone", "Fax", "Region"});
```

#### [VB .NET]

```
Dim gridModel As GridModel = gridDataBoundGrid1.Model
Dim binder As GridModelDataBinder = gridDataBoundGrid1.Binder

' "." indicates a new row.
binder.LayoutColumns(New String() {"CustomerID", "CompanyName",
"ContactTitle", "ContactName", ".", "Address", "City", ".",
"PostalCode",
"Country", "Phone", "Fax", "Region"})
```

### 4.2.2.14 Record Navigation Bar

It is possible to display a Grid Data Bound Grid within a Grid Record Navigation control. This combination will give you a look similar to Microsoft Access.



	CustomerID	CompanyName	ContactName	ContactTitle	Address
	ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57
	ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222
	ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312
	AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.
	BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8
	BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57
	BLONP	Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber
▶	BOLID	Bólido Comidas preparadas	Martín Sommer	Owner	C/ Araquil, 67
	BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Bouchers
	BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd.
	BSBEV	B's Beverages	Victoria Ashworth	Sales Representative	Fauntleroy Circus
	CACTU	Cactus Comidas para llevar	Patricia Simpson	Sales Agent	Cerrito 333
	CENTC	Centro comercial Moctezuma	Francisco Chang	Marketing Manager	Sierras de Granada 9993
	CHOPS	Chop-suey Chinese	Yang Wang	Owner	Hauptstr. 29
	COMMI	Comércio Mineiro	Pedro Afonso	Sales Associate	Av. dos Lusíadas, 23

Figure 230: Record Navigation



**Note:** For more details, refer the following browser sample:

```
<Install Location>\Syncfusion\EssentialStudio\Version
Number\Windows\Grid.Windows\Samples\2.0\Data Bound\Record Navigation Data Bound
Grid Demo
```

## Example

The following sample displays a Grid Data Bound Grid within a Grid Record Navigation control. This sample was created using the designer.

## Steps

1. Create an SqlDataAdapter and connect to the Customers table of the NorthWind database.
2. Drag Grid Record Navigation control onto the form.

**Note:** A DataSet is generated.

1. Create an SqlDataAdapter and connect to the Customers table of the NorthWind database.
2. Drag Grid Record Navigation control onto the form.

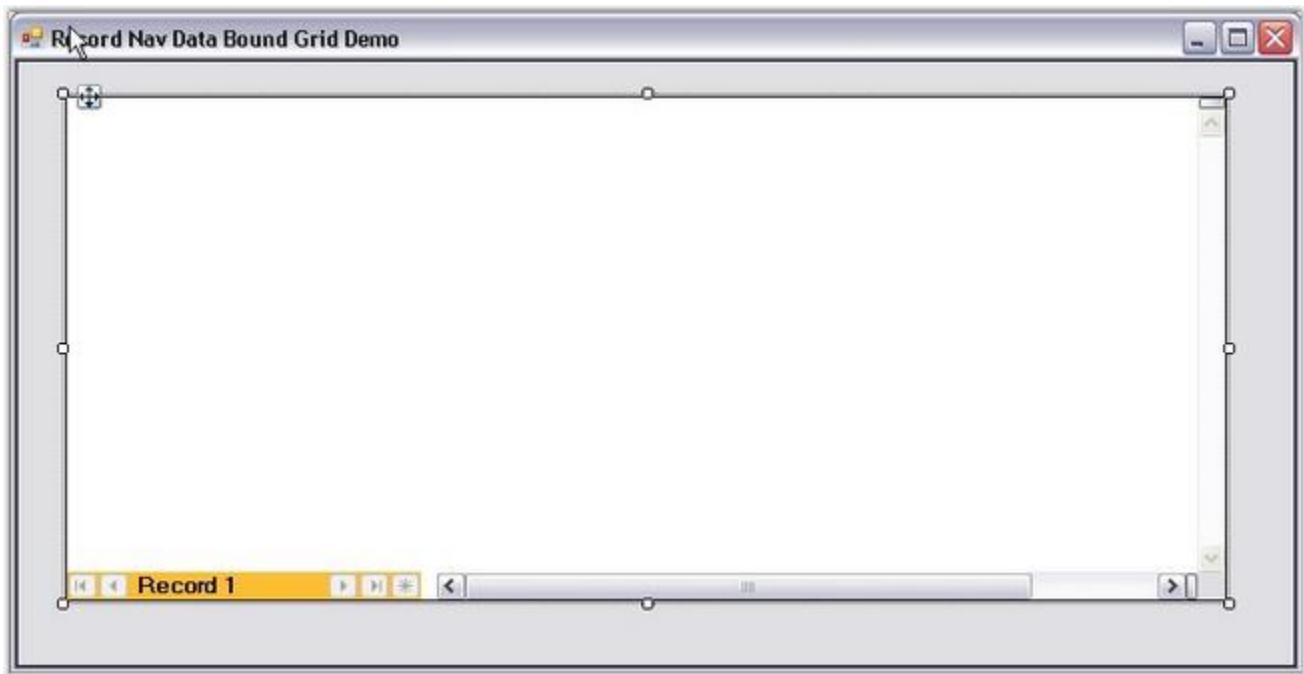


Figure 231: Grid Record Navigation Control dragged onto the Form

3. Drag Grid Data Bound Grid onto the Grid Record Navigation control.

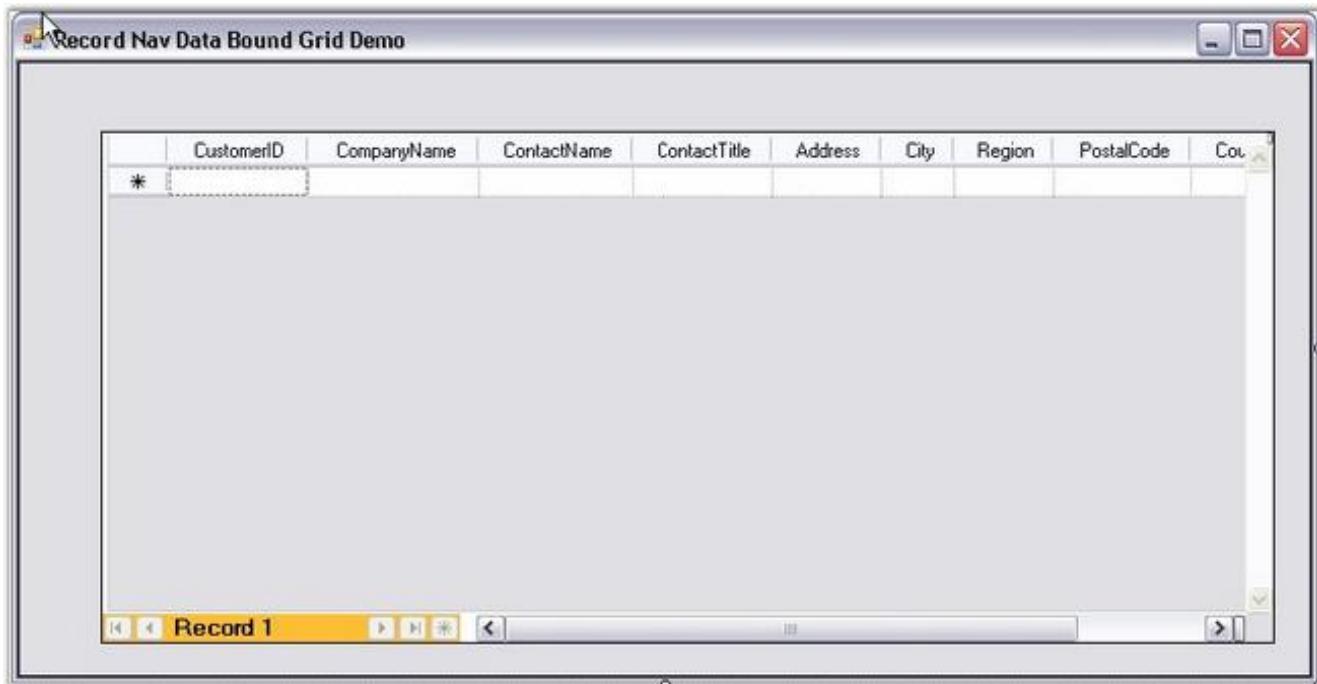


Figure 232: Grid Data Bound Grid dragged onto the Grid Record Navigation Control

 Note: Records can be displayed by typing in the NavigationBar.



	CustomerID	CompanyName	ContactName	ContactTitle	Address
	ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57
	ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222
	ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312
	AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.
	BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8
	BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57
	BLONP	Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber
▶	BOLID	Bólido Comidas preparadas	Martín Sommer	Owner	C/ Araquil, 67
	BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Bouchers
	BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd.
	BSBEV	B's Beverages	Victoria Ashworth	Sales Representative	Fauntleroy Circus
	CACTU	Cactus Comidas para llevar	Patricia Simpson	Sales Agent	Cerrito 333
	CENTC	Centro comercial Moctezuma	Francisco Chang	Marketing Manager	Sierras de Granada 9993
	CHOPS	Chop-suey Chinese	Yang Wang	Owner	Hauptstr. 29
	COMMI	Comércio Mineiro	Pedro Afonso	Sales Associate	Av. dos Lusíadas, 23

Figure 233: Records displayed in the Grid Data Bound Grid added to the Grid Record Navigation Control

A Grid Data Bound Grid is displayed within a Grid Record Navigation control.

#### 4.2.2.15 Multiple Headers

Grid Data Bound Grid supports display of multiple row and column headers. Additional row headers can be added along side the existing header by setting Model.Rows.HeaderCount and additional column headers can be added below the existing column header by setting the Model.Cols.HeaderCount property.

The following code example illustrates how to display multiple row and column headers.

```
[C#]

int extraRowHeaders = 1;
int extraColHeaders = 1;

// Initialize extra row and column headers.
this.gridDataBoundGrid1.Model.Rows.HeaderCount = extraRowHeaders;
this.gridDataBoundGrid1.Model.Cols.HeaderCount = extraColHeaders;
```

**[VB.NET]**

```
Dim extraRowHeaders As Integer = 1
Dim extraColHeaders As Integer = 1

' Initialize extra row and column headers.
Me.gridDataBoundGrid1.Model.Rows.HeaderCount = extraRowHeaders
Me.gridDataBoundGrid1.Model.Cols.HeaderCount = extraColHeaders
```

The resultant output is shown below.

	CustomerID	CompanyName
<b>Covered Cell - Group One</b>		
	ALFKI	Alfreds Futterkiste
◀	ANATR	Ana Trujillo Empare
	ANTON	Antonio Moreno Taq
	AROUT	Around the Horn
	BERGS	Berglunds snabbköp
	BLAUS	Blauer See Delikate
	BLONP	Blondesddsl père et
	BOLID	Bólido Comidas pre
	BONAP	Bon app'
▶	BOTTM	Bottom-Dollar Marke
	BSBEV	B's Beverages
	CACTU	Cactus Comidas par
	CENTC	Centro comercial M
	CHOPS	Chop-suey Chinese
	COMMI	Comércio Mineiro

Figure 234: Multiple Headers

#### 4.2.2.16 Grid Data Bound Grid Performance

Essential Grid Data Bound Grid can handle large amount of data without a performance hit.

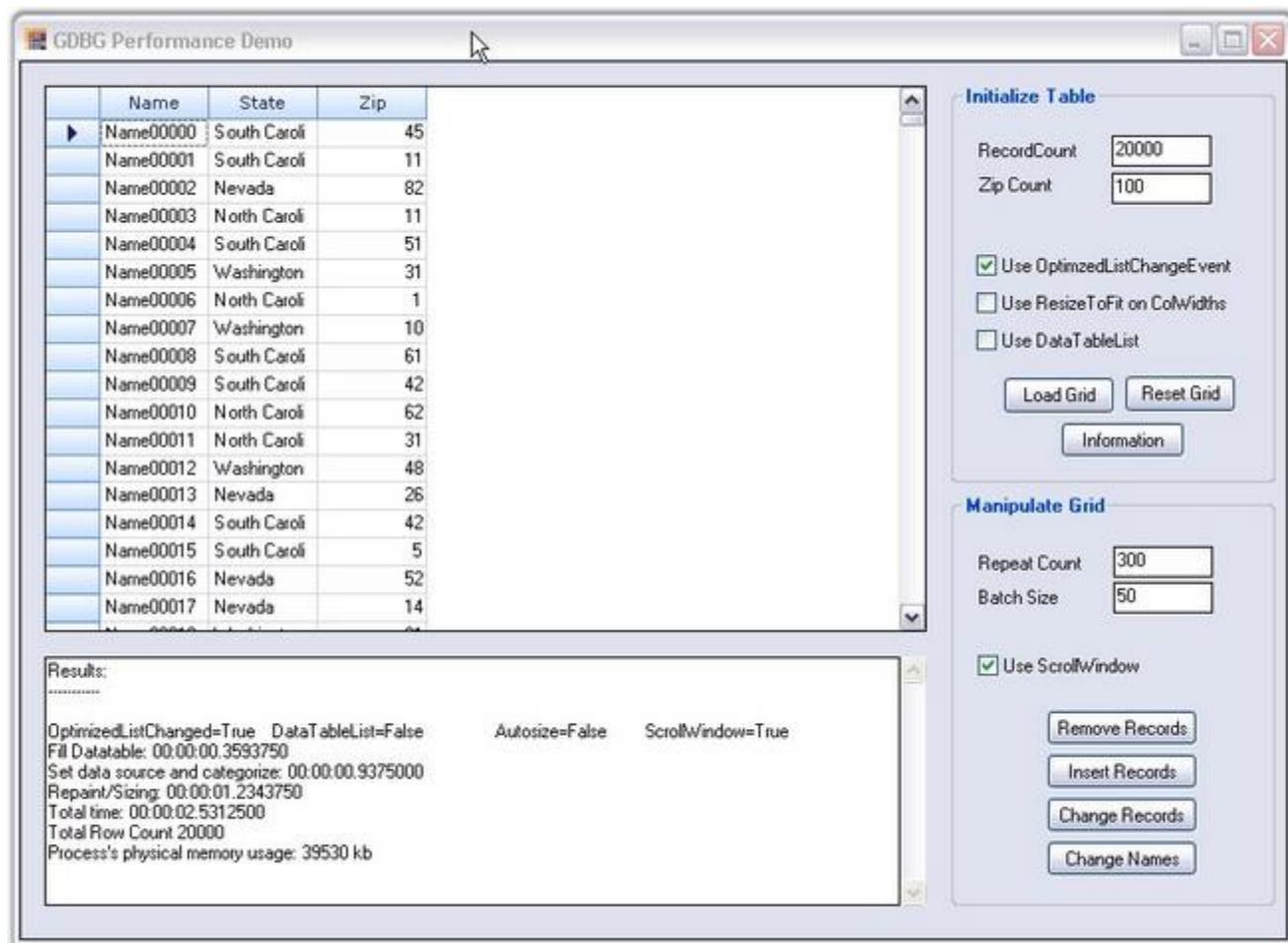


Figure 235: Grid Data Bound Grid

For more details, refer the sample under the following path from our sample browser:

**<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Windows\Samples\2.0\Data Bound\Grid Performance Demo**

### Example

The Grid can be loaded by specifying the number of record and using the options-Use OptimizedListChangeEvent, Use ResizeToFit on ColWidths and Use DataTableList.

In the Initialize Table group box

- **Use OptimizedListChangeEvent**—Selecting this option ensures that the grid data is updated using the `IBindingList.ListChanged` instead of the `CurrencyManager` change events.
- **Use ResizeToFit on ColWidths**—Selecting this option ensures column width is resized to fit the cell content after the data is loaded.
- **Use DataTableList**—Selecting this option ensures that the `Syncfusion.Collections.DataTableWrapperList` is used as the data source instead of the data table. The `DataTableWrapperList` is an `IBindingList` collection that wraps a data table and provides optimized access to the rows of the data table in turn improving the performance when inserting records into an existing table holding many records.

In the Manipulate Grid group box

- The Repeat Count and the Batch Size can be specified in order to check the performance in batch updates.
- Selecting the Use ScrollWindow check box invalidates only the inserted or removed rows instead of invalidating the whole Grid.
- The Insert Records, Remove Records and Change Records buttons let you check the performance when inserting, removing or changing the records in the underlying data table. Once the data is loaded after the batch update, you will be able to see the performance and the memory usage in a text box, as shown below.



Figure 236: Grid Performance Check

#### 4.2.2.17 Events

The important events in Grid control and Grid Data Bound Grid are as follows.

##### Current Cell Related Events

- **CurrentCellAcceptedChanges**—Occurs when the grid accepts changes made to the active current cell.

- **CurrentCellMoved**-Occurs when current cell is moved to the new position.
- **CurrentCellMoving**-Occurs when the current cell is about to be moved to a new position.
- **CurrentCellMoveFailed**-Occurs when the current cell movement to a new position fails.
- **CurrentCellActivating**-Occurs before the grid activates the specified cell as current cell.
- **CurrentCellActivated**-Occurs after the grid activates the specified cell as current cell.
- **CurrentCellActivateFailed**-Occurs after the grid fails to activate a specific cell as current cell.
- **CurrentCellDeleting**-Occurs when the user presses the Delete key on an active current cell.
- **CurrentCellStartEditing**-Occurs before the current cell switches into editing mode.
- **CurrentCellChanging**-Occurs when the user wants to modify contents of the current cell.
- **CurrentCellChanged**-Occurs when the user changes contents of the current cell.
- **CurrentCellDeactivating**-Occurs before the grid deactivates the current cell.
- **CurrentCellShowedDropDown**-Occurs after the drop-down is displayed.
- **CurrentCellCloseDropDown**-Occurs when the drop-down in the current cell is closed.
- **CurrentCellShowingDropDown**-Occurs when the drop-down is about to be shown.
- **CurrentCellValidating**-Occurs when the grid validates content of the active current cell.
- **CurrentCellValidated**-Occurs when the grid has successfully validated the content of the active current cell.
- **CurrentCellAcceptedChanges**-Occurs when the grid accepted changes made to the active current cell.
- **CurrentCellConfirmChangesFailed**-Occurs when the grid could not save changes made to the active current cell.
- **CurrentCellRejectedChanges**-Occurs when the grid rejects changes made to the active current cell.
- **CurrentCellEditingComplete**-Occurs when the grid completes editing mode, i.e., when the active current cell exits the editing mode.
- **CurrentCellDeactivated**-Occurs after the grid deactivates current cell.
- **CurrentCellDeactivateFailed**-Occurs after the grid fails to deactivate the current cell.
- **CurrentCellControlGotFocus**-Occurs when the current cell has switched to in-place editing and the control associated with the current cell has received the focus.
- **CurrentCellControlLostFocus**-Occurs when the current cell has switched to in-place editing and the control associated with the current cell has lost the focus.
- **CurrentCellKeyPress**-Occurs before GridCellRendererBase.OnKeyPress is called.
- **CurrentCellKeyDown**-Occurs before GridCellRendererBase.OnKeyDown is called.
- **CurrentCellKeyUp**-Occurs before GridCellRendererBase.OnKeyUp is called.

#### Mouse Related Events

- **CellButtonClicked**-Occurs when the user has clicked on a button element inside a cell renderer.
- **CellClick**-Occurs when user clicks inside a cell.
- **CellDoubleClick**-Occurs when the user double-clicks inside a cell.
- **CellMouseDown**-Occurs before the OnMouseDown method of a cell's renderer is called.
- **CellMouseUp**-Occurs before the OnMouseUp method of a cell's renderer is called.

- **CellMouseHoverEnter**-Occurs before the OnMouseHoverEnter method of a cell's renderer is called.
- **CellMouseHoverLeave**-Occurs before the OnMouseHoverLeave method of a cell's renderer is called.
- **CheckBoxClick**-Occurs when the user selects a check box in a cell.
- **PushButtonClick**-Occurs when the user clicks a push button of the PushButton cell.

#### Other Events

- **Click**-Occurs when control is clicked once.
- **DoubleClick**-Occurs when control is double clicked.
- **DragDrop**-Occurs when drag-drop operation is completed.
- **DragEnter**-Occurs when an object is dragged into the control's bounds.
- **DragLeave**-Occurs when the object is dragged out of the control's bound.
- **DragOver**-Occurs when the object is dragged over the control.
- **GotFocus**-Occurs when control receives focus.
- **LostFocus**-Occurs when control loses focus.
- **MouseDown**-Occurs when the mouse pointer is over the control and a mouse button is pressed.
- **MouseUp**-Occurs when the mouse pointer is over the control and a mouse button is released.
- **MouseEnter**-Occurs when mouse pointer enters the control.
- **MouseLeave**-Occurs when mouse pointer leaves the control.
- **QueryCanOleDragRange**-Occurs when the user hovers the mouse over the edge of a selected range.
- **ResizingColumns**-Occurs when user is resizing a selected range of columns.
- **ResizingRows**-Occurs when user is resizing a selected range of rows.

### 4.2.2.18 Enables Migration of .Net Grid to Essential Grid

Look-up table that Enables Migration of .Net Grid to Essential Grid

The following section contains document that enables users to migrate **.Net Grid to Essential Grid**. Most of the properties, events, methods have common name in both the grid. So it is not included in the following table. Since the API of both the grid is different, following document contains only common features that can be implemented with single line of code.

#### Equivalent Properties available

<b>.Net Grid</b>	<b>Essential Grid</b>	<b>Description</b>
AllowDrop	AllowDrop	Gets or sets a value indicating

		whether the control can accept data that the user drags into it.
AllowUserToAddRows	EnableAddNew	Gets or sets a value indicating whether the option to add rows is displayed.
AllowUserToDeleteRows	EnableRemove	Gets or sets a value indicating whether it allows delete the rows.
AllowUserToResizeColumns	ResizeRowsBehavior	Gets or sets a value indicating whether the it allows dragging of selected columns for rearranging.
AllowUserToResizeRows	ResizeRowsBehavior	Gets or sets a value indicating wheather row is resizable.
ColumnCount	Model.ColCount	Gets or sets the number of columns displayed
ColumnHeadersHeight	Model.RowHeights[0]	Gets or sets the width of the row.
ColumnHeadersVisible	Properties.ColHeaders	Gets or sets a value indicating whether the column header row is displayed.
HorizontalScrollingOffset	HScrollIncrement	Gets or sets the number of pixels by which the control is scrolled horizontally.
GridColor	Properties.GridLineColor	Gets or sets the color of the grid lines separating the cells.
MultiSelect	AllowSelection	Gets or sets a value indicating whether more than one cell, row, or column can be selected.
RowCount	Model.RowCount	Gets or sets the number of

		rows displayed.
RowHeadersVisible	Properties.RowHeaders	Gets or sets a value indicating whether the column that contains row headers is displayed.
RowHeadersWidth	Model.ColWidths[0]	Gets or sets the width of the column.
VerticalScrollingOffset	VScrollIncrement	Gets the number of pixels by which the control is scrolled vertically.
IsCurrentCellInEditMode	CurrentCell.IsEditing	Gets a value indicating whether the currently active cell is being edited.
RightToLeft	RightToLeft	Gets or sets a value indicating whether control's elements are aligned to support locales using right-to-left fonts.

#### Equivalent Events available

.Net Grid	Essential Grid	Description
BackgroundColorChanged	BackColorChanged	Occurs when the value of the <b>System.Windows.Forms.Control.BackColor</b> property changes.
CellMouseEnter	CellMouseHoverEnter	Occurs when the mouse pointer hovers over a cell.
CellMouseLeave	CellMouseHoverLeave	Occurs when the mouse pointer leaves a cell.
CellPainting	CellDrawn	Occurs when a cell needs to be drawn.
ColumnWidthChanged	Model.ColWidthsChanged	Occurs when column width changes.
DataSourceChanged	Binder.DataSourceChanged	Occurs when the <b>DataSource</b> property is changed.

RowsRemoved	Model.RowsRemoved	Occurs when a row/ rows are deleted.
SelectionChanged	Model.SelectionChanged	Occurs when current selection changes.
SelectionChanged	Model.RowHeightsChanged	Occurs when row height changes.

## Methods

.Net Grid	Essential Grid	Description
ClearSelection()	Selections.Clear()	Clears the current selection by unselecting all selected cells.
InvalidateCell()	Model.InvalidateRange()	Invalidates the specified cell forcing it to be repainted.
InvalidateColumn()	Model.InvalidateRange()	Invalidates the specified column forcing it to be repainted.
InvalidateRow	Model.InvalidateRange()	Invalidates the specified row forcing it to be repainted.
RefreshEdit	CurrentCell.Refresh()	Refreshes the value of the current cell

### 4.2.2.19 Field Chooser for Grid Data Bound Grid

This feature enables you to customize the view of the grid without modifying the database. The *FieldChooser* class of a *GridDataBoundGrid* has been implemented to add or remove columns from a grid.

## Use Case Scenarios

This feature will be useful when you want to remove certain columns (which cannot be deleted) from the grid.

## Methods

Table 9: Methods Table

Method	Description	Parameters	Type	Return Type	Reference links
WireGrid	Used to wire the <i>FieldChooser</i> .	Overloads: ( Arg1)	In GridWindowsForm	Example: GridDataboundGrid1.WireGrid(GridDataboundGrid).	NA
UnWireGrid	Used to unwire the <i>FieldChooser</i> .	NA	In GridWindowsForm	Example: GridDataboundGrid1.Unwired().	NA

**Sample Link**

You can find a sample for this feature in the following location:

..\\AppData\\Local\\Syncfusion\\EssentialStudio\\9.4.0.49\\Windows\\Grid.Windows\\Samples\\2.0\\Data Bound\\GDBG FieldChooser Demo

**Adding Field Chooser for Grid Data Bound Grid**

1. To add field chooser, pass data bound grid as the parameter of the *WireGrid* method.

The following code illustrates this:

[C#]

```
GridDataBoundFieldChooser fChooser = new
GridDataBoundFieldChooser();
fChooser.WireGrid(this.GridDataBoundGrid1);
```

[VB]

```
Dim fchooser As GridDataBoundFieldChooser = New
GridDataBoundFieldChooser()
fchooser.WireGrid(Me.GridDataBoundGrid1)
```

2. When the code runs, the entire grid will open.
3. Right click on a column header and select the **Field Chooser** menu item to view the **Field Chooser dialog**.

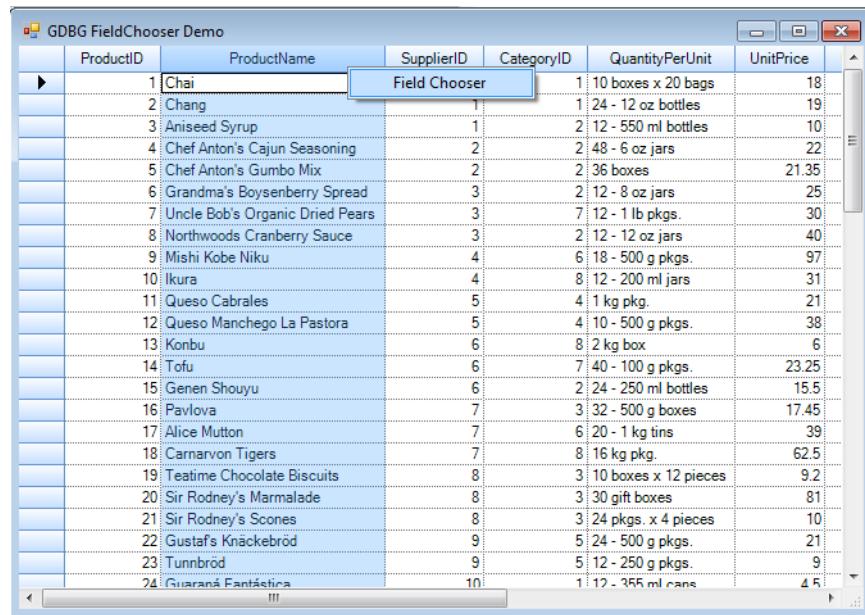


Figure 237: Field Chooser

4. This dialog will list all the column names with check boxes adjacent to them.

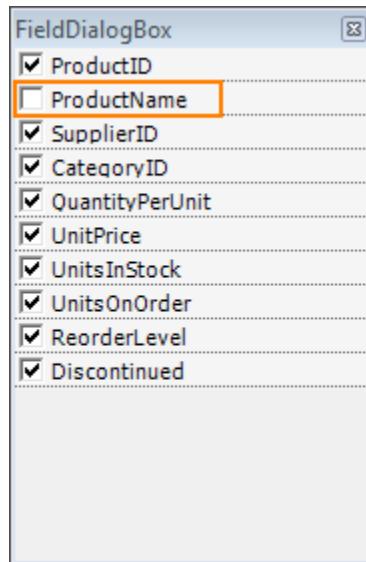
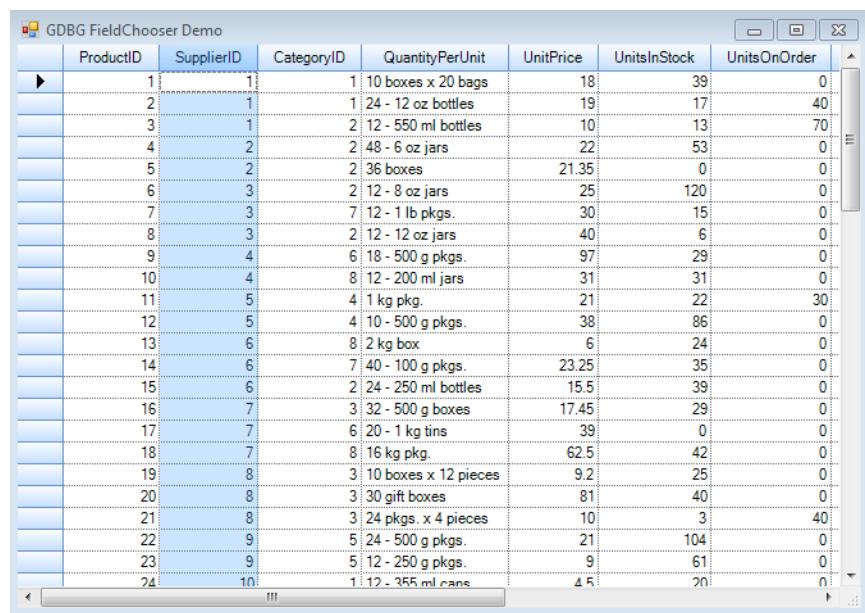


Figure 238: FieldDialogBox

5. Select the checkboxes of the columns you want to be displayed in the grid.
6. The grid will have only the columns which are selected in the **Field Chooser** dialog.



The screenshot shows a Windows application window titled "GDBG FieldChooser Demo". Inside, there is a data grid control displaying a table of product information. The columns are labeled: ProductID, SupplierID, CategoryID, QuantityPerUnit, UnitPrice, UnitsInStock, and UnitsOnOrder. The data consists of 24 rows, each representing a different product with its details. The grid has a light blue header and white body, with horizontal and vertical scroll bars visible.

	ProductID	SupplierID	CategoryID	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder
▶	1	1	1	10 boxes x 20 bags	18	39	0
	2	1	1	24 - 12 oz bottles	19	17	40
	3	1	2	12 - 550 ml bottles	10	13	70
	4	2	2	48 - 6 oz jars	22	53	0
	5	2	2	36 boxes	21.35	0	0
	6	3	2	12 - 8 oz jars	25	120	0
	7	3	7	12 - 1 lb pkgs.	30	15	0
	8	3	2	12 - 12 oz jars	40	6	0
	9	4	6	18 - 500 g pkgs.	97	29	0
	10	4	8	12 - 200 ml jars	31	31	0
	11	5	4	1 kg pkg.	21	22	30
	12	5	4	10 - 500 g pkgs.	38	86	0
	13	6	8	2 kg box	6	24	0
	14	6	7	40 - 100 g pkgs.	23.25	35	0
	15	6	2	24 - 250 ml bottles	15.5	39	0
	16	7	3	32 - 500 g boxes	17.45	29	0
	17	7	6	20 - 1 kg tins	39	0	0
	18	7	8	16 kg pkg.	62.5	42	0
	19	8	3	10 boxes x 12 pieces	9.2	25	0
	20	8	3	30 gift boxes	81	40	0
	21	8	3	24 pkgs. x 4 pieces	10	3	40
	22	9	5	24 - 500 g pkgs.	21	104	0
	23	9	5	12 - 250 g pkgs.	9	61	0
	24	10	1	12 - 355 ml cans	45	20	0

Figure 239: Customized Grid

## 4.3 Grid Grouping Control

Grid Grouping control supports grouping with summaries, filters and expression columns; can be used with flat data sources, hierarchical data sources or data sources with multiple nested tables.

### 4.3.1 Creating Grid Grouping Control

This section will provide the step-by-step procedure to create a Grid Grouping control through designer and through programmatical approach in a .NET application.

### 4.3.1.1 Through Designer

For details, see [Tutorials Lesson 1: Grid Grouping Control Designer](#).

### 4.3.1.2 Through Code

This section will give you a step by step tutorial on creating a Grid Grouping control through code. You can bind the Grid Grouping control either to an MDB file or to a data source that has been created manually.

In this lesson, you will learn about the following topics.

#### 4.3.1.2.1 Binding a Grid Grouping Control to an MDB File

This example illustrates how to bind a Grid Grouping control to an MDB file at run time. It uses OleDbConnection and OleDbAdapter objects to get connected to a data source that exposes OLE DB interface. Try a similar approach to connect to a database from MS SQL Server.

1. Include the required namespace.

[C#]

```
using Syncfusion.Windows.Forms.Grid.Grouping;
using System.Data.OleDb;
```

[VB .NET]

```
Imports Syncfusion.Windows.Forms.Grid.Grouping
Imports System.Data.OleDb
```

2. Create an instance of Grid Grouping control and specify its size.

[C#]

```
private Syncfusion.Windows.Forms.Grid.Grouping.GridGroupingControl
gridGroupingControl1;
```

```
this.gridGroupingControl1 = new  
Syncfusion.Windows.Forms.Grid.Grouping.GridGroupingControl();  
this.gridGroupingControl1.Size = new System.Drawing.Size(160, 200);
```

**[VB.NET]**

```
Private gridGroupingControl1 As  
Syncfusion.Windows.Forms.Grid.Grouping.GridGroupingControl  
  
Me.gridGroupingControl1 = New  
Syncfusion.Windows.Forms.Grid.Grouping.GridGroupingControl()  
Me.gridGroupingControl1.Size = New System.Drawing.Size(160, 200)
```

3. Set up the Data Source.

**[C#]**

```
// Create a Connection Object.  
OleDbConnection connection = new  
OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data  
Source=C:\\Data\\NWIND.MDB");  
  
// Create a Data Adapter.  
OleDbDataAdapter adapter = new OleDbDataAdapter("SELECT * FROM  
Customers", connection);  
  
// Create and fill a Data Set.  
DataSet dtSet = new DataSet();  
adapter.Fill(dtSet);
```

**[VB.NET]**

```
' Create a Connection Object.  
Dim Connection As OleDbConnection = New  
OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data  
Source=C:\\Data\\NWIND.MDB")  
  
' Create a Data Adapter.  
Dim Adapter As OleDbDataAdapter = New OleDbDataAdapter("SELECT * FROM  
Customers", Connection)  
  
' Create and fill a Data Set.  
Dim dtSet As DataSet = New DataSet()  
Adapter.Fill(dtSet)
```

4. Bind the grid grouping control to this data table by setting its **DataSource** property.

[C#]

```
this.gridGroupingControl1.DataSource = dtSet.Tables[0];
```

[VB .NET]

```
Me.GridGroupingControl1.DataSource = dtSet.Tables(0)
```

5. Finally add the grid grouping control to the form.

[C#]

```
this.Controls.Add(this.gridGroupingControl1);
```

[VB .NET]

```
Me.Controls.Add(Me.GridGroupingControl1)
```

6. When you run the application, the grid will look like the one in the following image.

CustomerID	CompanyName	ContactName
ALFKI	Alfreds Futterkiste	Maria Anders
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo
ANTON	Antonio Moreno Taquería	Antonio Moreno
AROUT	Around the Horn	Thomas Hardy
BERGS	Berglunds snabbköp	Christina Berglund
BLAUS	Blauer See Delikatessen	Hanna Moos
BLONP	Blondesddsl père et fils	Frédérique Citeaux
BOLID	Bólido Comidas preparadas	Martín Sommer
BONAP	Bon app'	Laurence Lebihan
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln

Figure 240: Binding a Grid Grouping control to an MDB File

#### 4.3.1.2.2 Bind a Grid Grouping Control to a Manual Data Source

Here are some code samples that will create a **DataTable** and bind it to a Grid Grouping control. Once you have a DataTable object populated you can use the **GridGroupingControl.DataSource** property to implement the binding.

1. Include the required namespace.

**[C#]**

```
using Syncfusion.Windows.Forms.Grid.Grouping;
```

**[VB .NET]**

```
Imports Syncfusion.Windows.Forms.Grid.Grouping
```

2. Create an instance of Grid Grouping control and specify its size.

**[C#]**

```
private Syncfusion.Windows.Forms.Grid.Grouping.GridGroupingControl  
gridGroupingControl1;  
  
this.gridGroupingControl1 = new  
Syncfusion.Windows.Forms.Grid.Grouping.GridGroupingControl();  
this.gridGroupingControl1.Size = new System.Drawing.Size(160, 200 );
```

**[VB .NET]**

```
Private gridGroupingControl1 As  
Syncfusion.Windows.Forms.Grid.Grouping.GridGroupingControl  
  
Me.gridGroupingControl1 = New  
Syncfusion.Windows.Forms.Grid.Grouping.GridGroupingControl()  
Me.gridGroupingControl1.Size = New System.Drawing.Size(160, 200 )
```

3. Set up the Data Source.

**[C#]**

```
DataTable myDataTable = new DataTable("MyDataTable");  
  
// Declare the Data Column and Data Row variables.  
 DataColumn myDataColumn;  
 DataRow myDataRow;  
  
// Create a new Data Column, set the Data Type and Column Name and add
```

```
to the Data Table.  
myDataColumn = new DataColumn();  
myDataColumn.DataType = System.Type.GetType("System.Int32");  
myDataColumn.ColumnName = "id";  
myDataTable.Columns.Add(myDataColumn);  
  
// Create a second column.  
myDataColumn = new DataColumn();  
myDataColumn.DataType = Type.GetType("System.String");  
myDataColumn.ColumnName = "item";  
myDataTable.Columns.Add(myDataColumn);  
  
// Create new Data Row objects and add to the Data Table.  
for (int i = 0; i <= 10; i++)  
{  
    myDataRow = myDataTable.NewRow();  
    myDataRow["id"] = i;  
    myDataRow["item"] = "item " + i.ToString();  
    myDataTable.Rows.Add(myDataRow);  
}
```

**[VB.NET]**

```
Dim myDataTable As DataTable = New DataTable("MyDataTable")  
  
' Declare the Data Column and Data Row variables.  
Dim myDataColumn As DataColumn  
Dim myDataRow As DataRow  
  
' Create a new Data Column, set the Data Type and Column Name and add  
to the Data Table.  
myDataColumn = New DataColumn()  
myDataColumn.DataType = System.Type.GetType("System.Int32")  
myDataColumn.ColumnName = "id"  
myDataTable.Columns.Add(myDataColumn)  
  
' Create a second column.  
myDataColumn = New DataColumn()  
myDataColumn.DataType = Type.GetType("System.String")  
myDataColumn.ColumnName = "item"  
myDataTable.Columns.Add(myDataColumn)  
  
' Create new Data Row objects and add to the Data Table.  
Dim i As Integer  
For i = 0 To 10  
    myDataRow = myDataTable.NewRow
```

```
myDataRow("id") = i  
myDataRow("item") = "item " & i  
myDataTable.Rows.Add(myDataRow)  
Next i
```

4. Bind the grid grouping control to this data table by setting its **DataSource** property.

[C#]

```
this.gridGroupingControl1.DataSource = myDataTable;
```

[VB .NET]

```
Me.GridGroupingControl1.DataSource = myDataTable
```

5. Finally, add the grid grouping control to the form.

[C#]

```
this.Controls.Add(this.gridGroupingControl1);
```

[VB .NET]

```
Me.Controls.Add(Me.GridGroupingControl1)
```

6. When you run the application, the grid will look like the one in the following image. You will be able to sort the data by clicking the header of the column you want to sort.

MyDataTable: 10 Items	
	id
*	
	2
	6
	4
	0
	9
	8
	7
	5
	item 28
	item 32
	item 39
	item 46
	item 56
	item 67
	item 74
	item 75

Figure 241: Binding a Grid Grouping control to a Manual Data Source

### 4.3.2 Feature Summary

Essential Grid Grouping control possesses advanced features such as Outlook style grouping and is highly optimized to handle large amount of data. This section lists the various features of the grid grouping control in brief.

- Filter by Display Member: Filter by Display Member, a new addition to Grid Helper Utilities. It allows the filtering of grid data by Display Member instead of Value Member. With this feature, the filter bar's drop-down lists the display member strings of the filtering column instead of its value member strings. You can use this feature with both Grid Data Bound Grid and Grid Grouping control.

	ProductID	ProductName	Category	Supplier
▶	1	Chai	(none)	Charlotte Cooper
	2	Chang	(custom)	Charlotte Cooper
	3	Aniseed Syrup	Beverages	Charlotte Cooper
	4	Chef Anton's Cajun S	Condiments	Shelley Burke
	5	Chef Anton's Gumbo	Confections	Shelley Burke
	6	Grandma's Boysenbe	Dairy Products	Regina Murphy
	7	Uncle Bob's Organic	Grains/Cereals	Regina Murphy
	8	Northwoods Cranberr	Meat/Poultry	Regina Murphy
	9	Mishi Kobe Niku	Produce	Regina Murphy
	10	Ikura	Seafood	Yoshi Nagase
	11	Queso Cabrales	Dairy Products	Yoshi Nagase
	12	Queso Manchego La	Dairy Products	Antonio del Valle Saavedra

Figure 242: Grid Data Bound Grid Filter



Figure 243: Grid Grouping control Filter

- Data Binding:** The grid grouping control supports variety of data sources which are used to automatically populate the grid with data. It has full ADO+ support and also allows any component that implements the **IList**, **IBindingList**, **ITypedList** or **IListSource** interface. It also supports unbound mode.

SupplierID	CompanyName	ContactName	ContactTitle
1	Exotic Liquids	Charlotte Cooper	Purchasing Manager
2	New Orleans Cajun Delights	Shelley Burke	Order Administrator
3	Grandma Kelly's Homestead	Regina Murphy	Sales Representative
4	Tokyo Traders	Yoshi Nagase	Marketing Manager
5	Cooperativa de Quesos 'Las Cabras'	Antonio del Valle Saavedra	Export Administrator
6	Mayumi's	Mayumi Ohno	Marketing Representative
7	Pavlova, Ltd.	Ian Devling	Marketing Manager
8	Specialty Biscuits, Ltd.	Peter Wilson	Sales Representative
9	PB Knäckebröd AB	Lars Peterson	Sales Agent

Figure 244: Data Binding Support

- Data Relations:** The grid can display data from multiple tables at once forming a hierarchical relationship. It also provides support to manually add relations.

	parentID	ParentName	ParentDec
-	0	parentName0	
	MyChildTable: 4 Items		
*	childID	Name	
▶	-	0	ChildName0
	MyGrandChildTable: 3 Items		
*	GrandChildID	Name	
	0	GrandChildName0	
	20	GrandChildName20	
	40	GrandChildName40	
+	5	ChildName5	
+	10	ChildName10	
+	15	ChildName15	
+	1	parentName1	

*Figure 245: Data Relations*

- **Grouping:** Grid data can be arranged based on matching field values to form groups. Essential Grid supports nested grouping by hierarchically combining the groups in different levels. The number of levels of grouping is unlimited.

Employees		Title	
	EmployeeID	LastName	FirstName
<b>Title: Inside Sales Coordinator - 1 Items</b>			
	8	Callahan	Laura
<b>Title: Sales Manager - 1 Items</b>			
	5	Buchanan	Steven
<b>Title: Sales Representative - 6 Items</b>			
	<b>Title: Vice President, Sales - 1 Items</b>		

*Figure 246: Nested Grouping support in the Grid Grouping Control*

- **Summaries:** The grid allows you to add summary rows which is used to display brief information about groups or specific columns. It is also possible to add user defined summaries.

ID	losses	School	Sport	ties	wins	year
217	9	Wake Forest	Football	0	2	2000
218	5	Wake Forest	Football	0	7	1999
219	8	Wake Forest	Football	0	3	1998
220	6	Wake Forest	Football	0	5	1997
221	8	Wake Forest	Football	0	3	1996
222	10	Wake Forest	Football	0	1	1995
223	8	Wake Forest	Football	0	3	1994
224	9	Wake Forest	Football	0	2	1993
225	4	Wake Forest	Football	0	8	1992
2085		3161				

Figure 247: Support to add Summaries

- Filters:** The grid grouping control has in-built support for FilterBar. The filter bar can be used to display a subset of records that satisfies a user-defined criteria.

	Address	City	CompanyName
		London	
[+]	120 Hanover Sq.	London	Around the Horn
[+]	Fauntleroy Circus	London	B's Beverages
[+]	Berkeley Gardens	London	Consolidated Holdings
[+]	35 King George	London	Eastern Connection
[+]	South House	London	North/South
[+]	90 Wadhurst Rd.	London	Seven Seas Imports

Figure 248: Filtering support in the Grid Grouping Control

- Sorting:** Grid Grouping control allows you to sort the data against an unlimited number of columns either in ascending or in descending order. When editing is done, the record values will be adjusted automatically to maintain the sorting order.

CustomerID	CompanyName	ContactName	ContactTitle
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner
AROUT	Around the Horn	Thomas Hardy	Sales Representative
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative
BLONP	Blondel père et fils	Frédérique Citeaux	Marketing Manager

Figure 249: Sorting support in the Grid Grouping Control

- **Expression Fields:** Expression fields facilitate the inclusion of columns with formula expressions.

Sport	ties	wins	year	Winning %	Lossing %
Basketball	0	15	2003	53.57	46.43
Football	0	6	2003	60.00	40.00
Baseball	0	54	2003	76.06	23.94
Basketball	0	13	2002	43.33	56.67
Basketball	0	11	2001	36.67	63.33
Basketball	0	10	2000	33.33	66.67
Basketball	0	20	1999	57.14	42.86
Basketball	0	18	1998	56.25	43.75
Basketball	0	23	1997	69.70	30.30

Figure 250: Expression Fields

- **Unbound Fields:** Grid Grouping control can have unbound fields with custom values using which you can evaluate the field values at runtime similar to Expression Fields.

	Contact ID	Contact Name	Unbound
1		ALFKI	<input checked="" type="checkbox"/>
2		ARCOS	<input type="checkbox"/>
3		BERGS	<input checked="" type="checkbox"/>
4		CHOPS	<input type="checkbox"/>
5		ERNSH	<input checked="" type="checkbox"/>
6		FRANK	<input type="checkbox"/>
7		KETHS	<input checked="" type="checkbox"/>
8		PALPS	<input type="checkbox"/>
9		RICHARD	<input checked="" type="checkbox"/>
10		SAMS	<input type="checkbox"/>

Figure 251: Unbound Fields in the Grid Grouping Control

- **Preview Rows:** It is possible to add a preview section for each group / record. The preview rows can be enabled when you want to display memo fields or some notes for a given group / record.

ID	FirstName	LastName	Address	City
1	Nancy	Davolio	507 - 20th Ave. E. Apt. 2A	Seattle
Education includes a BA in psychology from Colorado State University in 1970. She also completed a certificate in sales management at Seattle University in January 1992 and to vice president of sales in March 1993. Nancy is a member of the Sales Management Association.				
2	Andrew	Fuller	908 W. Capital Way	Tacoma
Andrew received his BTS commercial in 1974 and a Ph.D. in international marketing from the University of Washington in 1981. He is currently responsible for sales in the West. Andrew is a member of the Sales Management Association.				
3	Janet	Leverling	722 Moss Bay Blvd.	Kirkland
Janet has a BS degree in chemistry from Boston College (1984). She has also completed a certificate in sales management at Seattle University in January 1992 and to vice president of sales in March 1993. Janet is a member of the Sales Management Association.				

Figure 252: Preview Rows in the Grid Grouping Control

- **Frozen Columns:** This feature will allow you to lock specific columns so that they will always be visible on the screen no matter how far you scroll to the right or down.

parentID	ParentName	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8
0	parentName0	76	20	68	44	59	69
MyChildTable: 4 Items							
childID	Name	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8
0	ChildName0	9	36	2	89	6	
5	ChildName5	76	92	0	10	42	
10	ChildName10	34	90	19	92	37	
15	ChildName15	62	97	71	14	98	
1	parentName1	65	28	9	91	4	10
2	parentName2	90	7	68	66	89	98
3	parentName3	42	30	3	90	40	25
4	parentName4	8	74	41	70	52	78

Figure 253: Frozen Columns in the Grid Grouping Control

- **Stacked MultiHeaders:** This feature allows you to create additional header rows that span across visible grid columns. You can group the columns under each header row. It also supports drag / drop of these header rows.

Orders: 830 Items						
Header One			Header Two			
OrderID	CustomerID	EmployeeID	OrderDate	Freight	RequiredDate	
10248	VINET	5	7/4/1996	32.38	8/1/1996	
10249	TOMSP	6	7/5/1996	11.61	8/16/1996	
10250	HANAR	4	7/8/1996	65.83	8/5/1996	
10251	VICTE	3	7/8/1996	41.34	8/5/1996	
10252	SUPRD	4	7/9/1996	51.3	8/6/1996	
10253	HANAR	3	7/10/1996	58.17	7/24/1996	
10254	CHOPS	5	7/11/1996	22.98	8/8/1996	

Figure 254: Stacked MultiHeaders

- **Multi Row Record:** With Grid Grouping control, you can make a single record to span across several rows.

	ID	Photo	FirstName	Address		Region
			LastName	City	PostalCode	Country
1	1		Nancy	507 - 20th Ave. E. Apt. 2A		WA
			Davolio	Seattle	98122	USA
Education includes a BA in psychology from Colorado State University in 1970. She also co Toastmasters International.						
2	2		Andrew	908 W. Capital Way	WA	
			Fuller	Tacoma	98401	USA

Figure 255: Spanning Records over Multiple Rows

- **Designer Support:** The Grid Grouping control offers rich design time support to customize the appearance and behavior of the various grid elements.

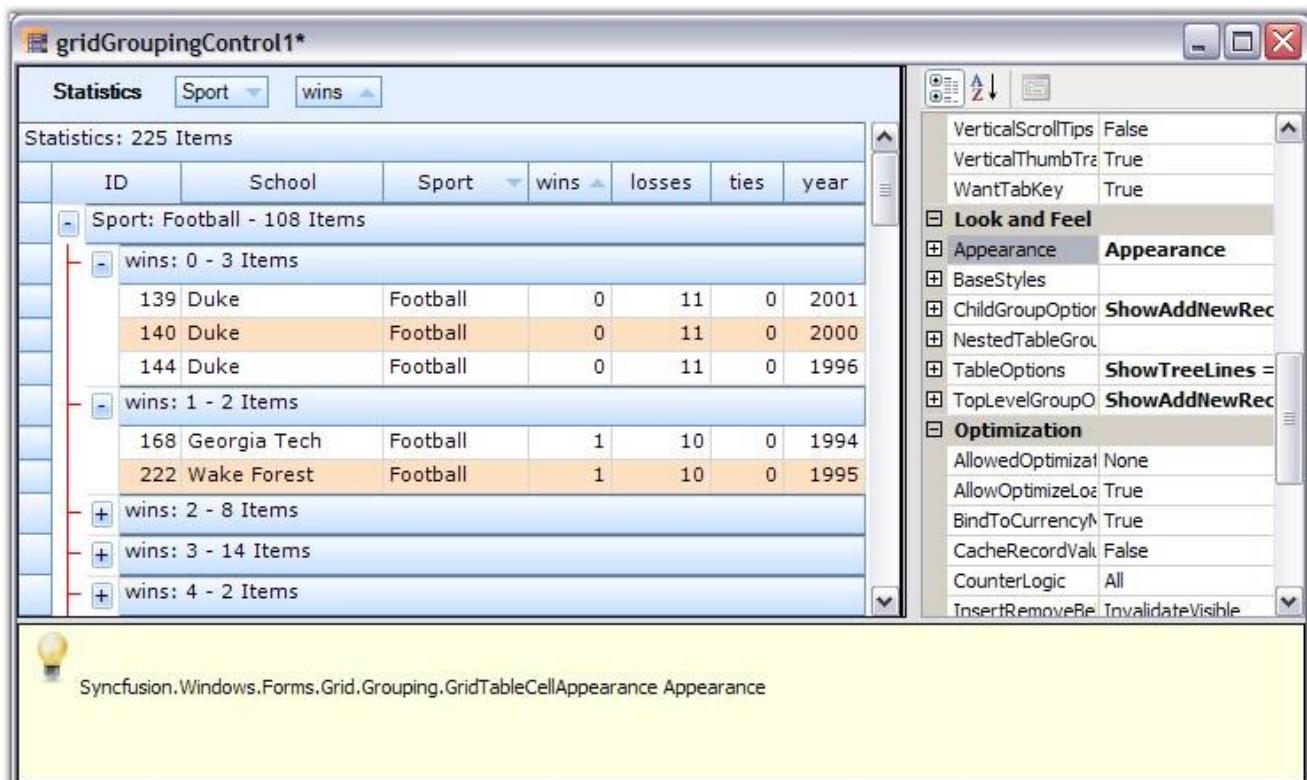


Figure 256: Design Time Support in the Grid Grouping Control

- Appearance:** The grid provides design time and run time options to customize the appearance of its elements. The appearance settings include various options such as background color, font, text color, alignment, and so on.

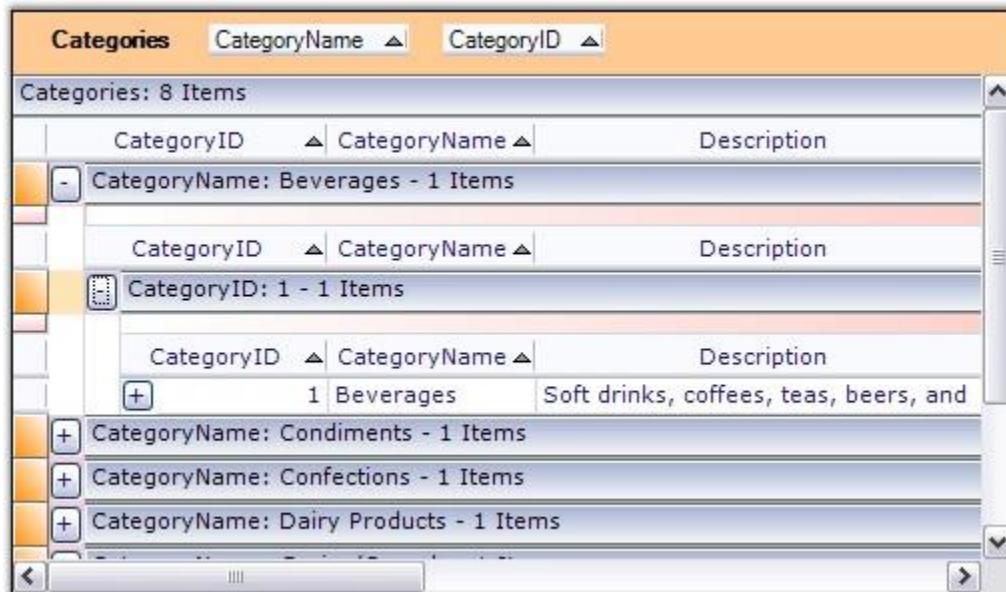


Figure 257: Customized Appearance of the Grid Grouping Control

- **Look and Feel:** Through this look and feel technology, you can provide a common appearance to all the grid elements such as headers, drop down buttons, etc. Here, the appearance denotes not only the way the cell controls get painted but also the way in which they respond to user actions like MouseDown, MouseHover, and so on.
- **Navigation:** Grid Grouping control includes built-in Navigation and Page Bars that let users navigate the records with ease.

CustomerID	CompanyName	ContactName
ALFKI	Alfreds Futterkiste	Maria Anders
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo
ANTON	Antonio Moreno Taquería	Antonio Moreno
AROUT	Around the Horn	Thomas Hardy
BERGS	Berglunds snabbköp	Christina Berglund
BLAUS	Blauer See Delikatessen	Hanna Moos
BLONP	Blondesddsl père et fils	Frédérique Citeaux
BOLID	Bólido Comidas preparadas	Martín Sommer
BONAP	Bon app'	Laurence Lebihan

Figure 258: Navigation Support

- **Optimizations:** Grid Grouping control is highly optimized to handle very high refresh and update scenarios. To increase performance, it also supports Virtual Mode.

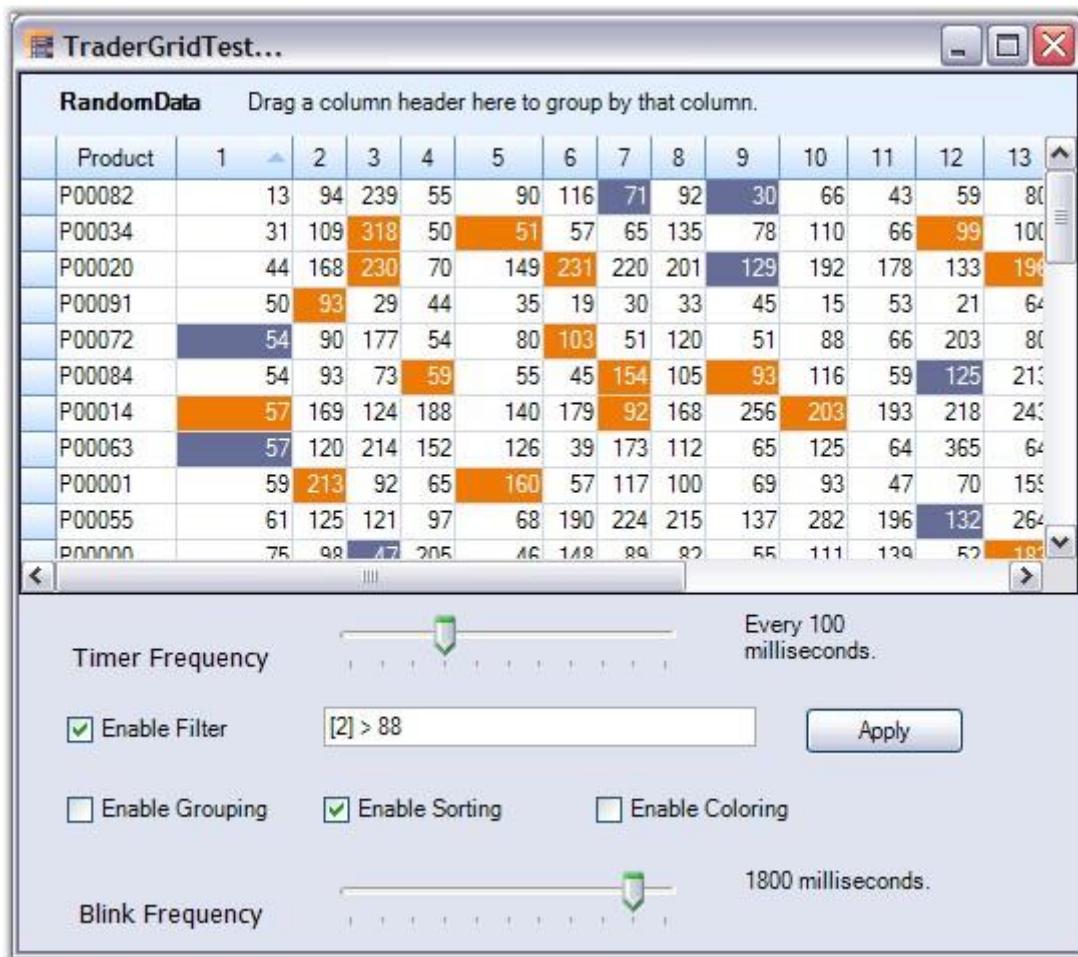


Figure 259: Increased Optimization Support

- **Serialization:** The grid schema information can be serialized and deserialized. Grid Grouping control supports several serialization formats such as XML and XLS.

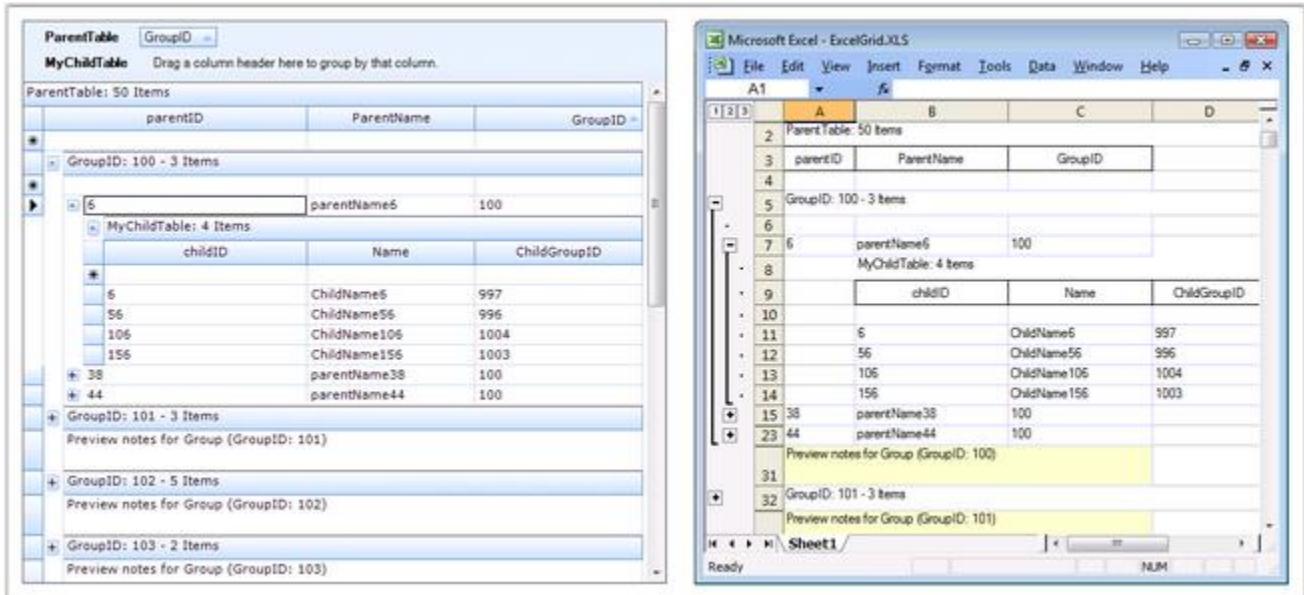


Figure 260: Serialization

### 4.3.3 Major Control Classes Overview

#### GridGroupingControl Class

The **GridGroupingControl** class is derived from the **Control** class and implements several interfaces that add Grouping support to this class. Provides support for displaying ADO.NET data and other data sources in a grid. Data will be loaded from the given datasource and changes will be written back to the datasource.

It is the proper choice if you need grouping support, multi-column sort support or true nested-table hierarchical support in a grid. It can be bound to any **IList** datasource. You can easily add expression columns, filter columns and summary rows to this grid. It is fully designable using Visual Studio and is customizable from code.

The **GridTableControl** is the main element in the Grid Grouping control. The Grid Table control displays the rows from the **Syncfusion.Grouping.Table.DisplayElements** collection of the Grid Grouping control. Table using schema information stored in the **TableDescriptor**.

The **TableDescriptor** gives access to the table schema information of the root table in the datasource. The **TableDescriptor** object is instantiated by the **GridEngine** class and initialized with default schema information from the list assigned to the **DataSource**.

There is only one GridEngine object for a Grid Grouping control. The GridTableDescriptor and GridTable objects on the other side can be more than one when hierarchies are displayed. For each hierarchy level, a GridTableDescriptor and GridTable are initialized. For example, if you have an ADO.NET DataSet with three tables: "Products", "Orders", and "OrderDetails", there will be three GridTableDescriptors and GridTables.

**Relations** between tables are defined with a GridTableDescriptor.Relations collection of a TableDescriptor. Each TableDescriptor can have one or multiple **RelationDescriptor** objects. A RelationDescriptor defines the foreign key columns in the parent table, a child with information about the related child table and the primary key columns in the child table.

The **GridTable** object is instantiated by the **GridEngine** class. The Table object manages the records from the engine's DataSource and provides access to records and grouped elements through several collection classes.

The most important collection used by the Grid Table control is the **DisplayElements** collection. This collection provides the Grid Table control with information on which element to display at a row. It returns elements such as CaptionSection, RecordRow, SummaryRow, and others. Based on the elements returned by this collection, the Grid Table control will display a record, a summary or a group caption bar. There are several collections returned such as Records which contains all records in the table. FilterRecords contains all visible records.

Another related collection is the **NestedDisplayElements** collection. Similar to DisplayElements, this collection also returns all records, groups and sections that are expanded and meet filter criteria. The only difference between these two collections is that the NestedDisplayElements collection steps into nested tables too where the DisplayElements collection will not.

### **The Grouping Engine**

**Engine** is the main object of the grid grouping control. It contains the **TableDescriptor** with schema information such as fields, relations and the **Table** with runtime representation of the data source with groups, records, data and display elements. The engine lets you set the main datasource for the whole engine. The TableDescriptor will pick up the ItemProperties (schema information) from the datasource and the table will be initialized at runtime with records from the list.

The **GridEngineBase** class adds design-time support for the engine class. It can be dropped as a component into the component tray of the designer. It can be initialized with a BindingContext so that the CurrencyManager can be kept synchronized.

The **GridEngine** class adds the plumbing for displaying the data in a Grid Grouping control. You can specify the datasource using the **DataSource** and **DataMember** properties through the designer. It is instantiated with the virtual **GridGroupingControl.CreateEngine** method. If you want to customize the engine object, you should subclass this class and should override the **CreateEngine** method.

The **GridEngine** object is the main grouping engine object. It is derived from the **Syncfusion.Grouping.Engine** base class and adds Windows Forms specific functionality such as support for a Forms **BindingContext** and **CurrencyManager**. **GridEngine** also has special overrides of the virtual **Engine.CreateTableDescriptor** and **Engine.CreateTable** methods so that the grid-specific derived **GridTable** class and **GridTableDescriptor** class are instantiated.

#### 4.3.4 Concepts and Features

This section provides you a basic insight into the grid grouping control architecture packed with detailed information on the features of the Essential Grid. It will also give you an overview of the major control classes.

Here you will learn about the following concepts which will help you to get familiar with Grid Grouping control.

##### 4.3.4.1 Performance

Grid Grouping control has an **extremely high performance standard**. It can handle very high frequency updates and refresh scenarios. It also offers complete support for Virtual Mode wherein the data will be loaded only on demand. By simply setting few properties, you can have the grid, work with large amounts of data without a performance hit.

All the properties that affects grid performance are wrapped into a category named **Optimization**. Here is an image of the property grid listing various optimization properties.

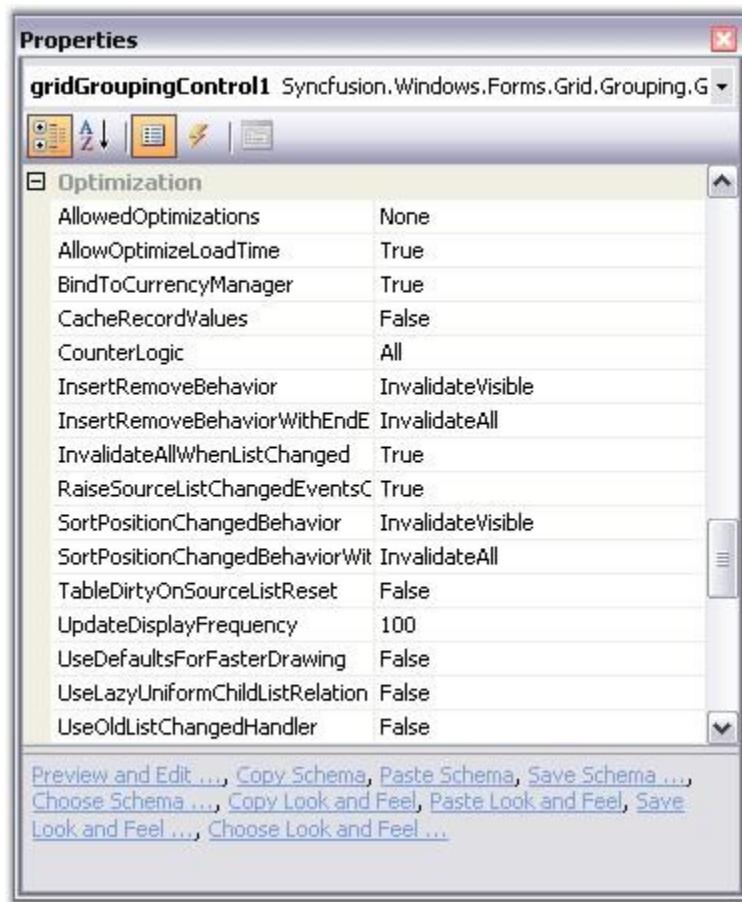


Figure 261: Optimization options in the Grid Grouping Control

### Optimization Properties - A Glance

Below is a brief overview on the above properties. We will discuss these properties further in detail with suitable examples in the forth coming chapters.

- **AllowedOptimizations**

Specifies the optimizations, the engine is allowed to use, when applicable. These optimizations can be used in combination with EngineCounter setting. EngineOptimizations enum defines the values for this property which will be discussed in the next chapter.

- **AllowOptimizeLoadTime**

This property might help in reducing the flickering issue at startup. When enabled, the grid will be rendered once into an offline bitmap before the form is shown for the first

time. This offline rendering of the grid ensures that all the required data is loaded into the memory and all grid data are initialized. Default value is true.

- **BindToCurrencyManager**

When you assign a DataTable to the grid grouping control, it will get access to the Default View of that Data Table through the Currency Manager that would listen to the updates to the underlying data table. The benefit of using CurrencyManager is that all the form elements would be kept in synchronization.

Using Currency Manager may cause performance issues in certain scenarios. In such cases you can bypass this Currency Manager and access the list directly without ever involving the CurrencyManager by setting this property to false (which is true by default). This will in turn detach the grid from the Currency Manager and hence the Grid Engine does not register the list with Windows Forms Currency Manager and it will solely relies on listening to ListChanged events.

- **CacheRecordValues**

If you have custom collections, you can choose to have the engine to cache record values with this property. When set to true, the engine will cache copies of the old values from a record in the record object. You can get these values with the Record.GetOldValue method. With custom collections, the engine can also determine exactly which values in a record were changed when the engine receives the ListChanged event and previous values were cached.

- **CounterLogic**

It specifies the CounterLogic to be used within the engine. GroupingEngine maintains the counters for the VisibleColumns, FilteredRecords, YAmount, HiddenRecords, and the like. These counters occupies a countable portion of the grid tree in memory. On every list change, all these counters need to be refreshed too along with the data records.

Invalidating all the counters is not required at all times. For instance, if you have a larger data source and you don't want support for groups and filtered records, then there is no need to maintain the counters such as FilteredRecords and the like. Keeping all the counters in memory will greatly increase the memory consumption which is not necessary in this case and this will give a big degradation on grid performance.

To handle such scenarios, Grid Grouping control provides options to skip allocating these counters. By setting this property, you can reduce the memory footprint by selectively disabling the counters which are not required in your application. EngineCounters enum defines the values for this property which will be discussed in the next chapter.

- **InsertRemoveBehavior / InsertRemoveBehaviorWithEndEdit**

These properties determine how the grid should react when a record is inserted or deleted. One or multiple records need to be shifted up and down. By default, the whole

display is invalidated and all the rows are repainted though only one record needs to be redrawn.

By setting these properties to **ListChangedInsertRemoveBehavior.ScrollWithImmediateUpdate**, you can instruct the engine not to repaint the whole screen. The engine will now determine the area affected by this change and use the ScrollWindow API to shift records up and down and only repaint the one record that was really changed. This will have a big impact if you have a larger grid and repainting the whole display is expensive.

- **InvalidateAllWhenListChanged**

It lets you specify whether the grid should simply call **Invalidate** when a ListChanged event is handled or if it should determine the area that is affected by the change and call **InvalidateRange**.

On first sight, you might think its better to determine the area that is affected by a change and call **InvalidateRange**. But when calling **InvalidateRange**, the grid needs to know the exact position of the record in the table before it can mark that area dirty. In order to determine the record position (and y-position of the row in the display), counters need to be evaluated. This operation can cost more time than simply calling **Invalidate** in high-frequency update scenarios. The group caption bar also needs to be updated when a record changes.

**InsertRemoveBehavior**, **SortPositionChangedBehavior** properties and **UpdateDisplayFrequency** can speed up the things a lot when **InvalidateAllWhenListChanged** is set to false.

- **RaiseSourceListChangedEventsOnEngineOnly**

When the engine handles the ListChanged event it will itself raise numerous events. When this property is set to true, the events will only be raised on the Engine object. If set to false then events will also be raised on inner objects (will bubble up on nested tables which causes some performance overhead). It will only have effect if **UseOldListChangedHandler** is set to false.

- **SortPositionChangedBehavior / SortPositionChangedBehaviorWithEndEdit**

These properties determine how to update the display with a change in the sort position of a record. By default, the whole display is invalidated and all the rows are repainted though only one record needs to be redrawn. This will degrade the performance and will not be efficient if you have a larger grid and repainting the whole display is expensive.

By setting these properties to **ListChangedInsertRemoveBehavior.ScrollWithImmediateUpdate**, you can instruct the engine not to repaint the whole screen. This will only repaint the one record that was really changed.

- **UpdateDisplayFrequency**

This property lets you specify the number of milliseconds to wait between display updates when new ListChanged event handler logic is used. This property doesn't have any effect if UseOldListChangedHandler = true. Special values are 0 - only manually update display by calling grid.Update() and 1 - update display immediately after each change.

- **UseDefaultsForFasterDrawing**

By setting this to true, you can quickly switch to faster GDI Draw Text, Solid Borders and more efficient calculation of the optimal width of a column. Initializes recommended settings to improve handling of ListChanged events and scrolling through grid. Affected settings are: TableOptions.ColumnsMaxLengthStrategy, TableOptions.GridLineBorder, TableOptions.DrawTextWithGdiInterop, TableOptions.VerticalPixelScroll, Appearance.AnyRecordFieldCell.WrapText and Appearance.AnyRecordFieldCell.Trimming.

- **UseOldListChangedHandler**

With version 4.4 the engine changed the way how the ListChanged event is handled internally to fix short-comings with performance of the code that was in place earlier. This property lets you switch back the behavior of the engine to the old mechanism if you notice compatibility issues. The default value is false.

- **BlinkTime**

Grid Grouping control has in-built support for highlighting cells for a short period of time after a change was detected to a cell. The BlinkTime property lets you specify the amount of time in milliseconds how long the values should be highlighted. You could also enable or disable this feature for individual columns by toggling GridColumnDescriptor.AllowBlink property.

## See Also

### [4.3.4.1.1 Memory Performance - Engine Optimizations](#)

This section discusses the various optimizations that GroupingEngine provides.

**EngineOptimizations** will greatly help to improve **Memory Performance**. Triggering these optimizations selectively will help a lot in reducing the memory footprint.

Engine optimizations can be enabled by setting **AllowOptimizations** to some value other than None. To optimize the memory usage, **CounterLogic** property needs to be assigned with a proper value.

### **AllowOptimizations**

The following is the list of optimizations the grid offers which are defined by EngineOptimizations enumeration. By default they are turned off, but you can tell the engine which optimizations should be applied when the specified criteria for the optimizations met.

- **EngineOptimizations Enumeration**

Specifies the values for AllowedOptimizations property.

- **None**

All optimizations are disabled.

- **DisableCounters**

When the engine detects that a table does not have RecordFilters, GroupedColumns or nested relations, counter logic will be disabled for the RecordsDetails collection since all counters are in sync with actual records (e.g. all records in data source are shown in TopLevelGroup). With this optimization the engine does still have full support for sorting.

- **VirtualMode**

When all criteria are met for the optimization and in addition to that no SortedColumns are set, the RecordsDetails collection does not have to be initialized at all. Instead, it can create records elements on demand and discard them using regular garbage collection when no references to a Record exist any more (e.g. once you scroll them out of view). This approach reduces memory footprint to absolute minimum. You should be able to load and display millions of records in a table. The PrimaryKey collection is still supported, but it will be initialized only on demand if you do access the Table.PrimaryKeyRecords collection. In such case all records will be enumerated.

- **PassThroughSort**

When all criteria are met for the optimization and SortedColumns are set, the engine will normally have to loop through records and sort them. When you specify the engine will check if the data source is an IBindingList and if IBindingList.SupportsSort returns true. In such case the data source will be sorted using its IBindingList.Sort routine and the engine will access records using VirtualMode. Using the IBindingList is usually a bit faster than the engines own sorting routines, but the disadvantage is that you will loose CurrentRecord and SelectedRecords information. Also, inserting and removing records will be slower (especially if the underlying data source is a DataView).

PassThroughSort will be ignored if criteria are met for the optimization are not met. If you want to force a Pass-through sort mechanism in such case, it can be done by implementing the IGroupingList interface. This allows performing the sort on the dataview directly instead of letting the grouping engine perform the sorting. Normally, it is recommended to use the engines own Sort mechanism and only rely on PassThroughSort for Virtual mode scenarios.

- **RecordsAsDisplayElements**

When the engine detects that records do not have nested child tables, no record preview rows are being used and each record only has one row (no ColumnSets are used), records do not have to be split into RecordParts. Instead when querying the DisplayElements collection for a specific row, the engine can simply return a Record element instead of a RecordRow element. The same applies to CaptionSection, ColumnHeaderSection and FilterBarSection. Instead of returning a CaptionRow, ColumnHeaderRow or FilterBarRow element the DisplayElements collection returns the section element. If you use this optimization you need to be careful in your own code and be aware that when you query the DisplayElements collection instead of a RecordRow element a Record element can be returned. Same issue is also with ColumnHeader, FilterBase and Caption.

- **All**

Enables the optimizations - DisableCounters, VirtualMode and RecordsAsDisplayElements.

Based on the schema that you specify the engine will determine if certain optimizations can be applied. If you do have a flat table and do not sort the records VirtualMode will be applied and the records don't have to be touched at all (only when drawing). If you sort records then TreeTables will be built so that the grid can sort the records but the logic for filtering and grouping is turned off (DisableCounters optimization). In case of pass-through sorting, the table is sorted by using the DataView.Sort routine and records will be accessed with VirtualMode. If you group records or if you have nested tables then full grouping logic will be needed.

- **CounterLogic**

In addition to being able to specify VirtualMode or WithoutCounter mode, you can also specify which counters you need. Most of the times you only want to count visible elements and filtered records and you can leave out custom counters, hidden element counter and others. That can save you a few bytes per record (40-80 bytes). The engine will also determine whether records actually need to be broken into pieces or if a record can be returned as leave elements (RecordsAsDisplayElements option). This again saves a few bytes per record.

- **EngineCounters Enumeration**

Specifies the values for CounterLogic.

- **All**

All counters are supported: visible elements, filtered records, YAmount, hidden elements, hidden records, CustomCount and VisibleCustomCount. Highest memory footprint.

- **FilteredRecords**

Counts only visible elements and filtered records. Smallest memory footprint.

- **YAmount**

Counts visible elements, filtered records and YAmount. Medium memory footprint.



**Note:** Allowing certain optimizations does not mean that the optimization is necessarily used. Optimizations will only be used when applicable. Take for example the optimization. If you allow this optimization the engine will check schema settings when loading the table. If there are no SortedColumns, RecordFilters, GroupedColumns and no nested relations for a table, then virtual mode can be used and no records need to be loaded into memory. If the user later sorts by one column, the virtual mode cannot be used any more. Records will need to be iterated through and sorted and tree structures will be built that allow quick access to records and IndexOf operations. When initializing the table the engine will check if criteria for DisableCounters optimization are met.

### Example

This example illustrates the Virtual Mode and WithoutCounter optimizations having a VirtualList as the data source. The VirtualList is just a [CustomCollection](#) that implements [IList](#) and [ITypedList](#) interfaces. The list is populated with 100K items and the same is bound to the grid grouping control. The example also displays a Log Window where you could track the different optimizations applied at different instances. It also displays the time elapsed for populating the grid grouping control.

All the optimizations are enabled by setting AllowedOptimizations to All. As said earlier, the optimizations specified will not be applied at all times. They will only be used when applicable, that is, when the criteria for those optimizations are met. This example best illustrates this process. On every property change, the log window displays the list of optimizations applied to the grid at that instance. When you run the sample, you could be able to track the optimizations applied in different engine states like with/without grouping, with/without sorting, etc.



**Note:** For more details, refer the following Browser Sample:

<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Performance\Engine Optimization Demo

### Implementation

Follow the steps below to experiment different engine optimizations.

1. Create a class (VirtualItem) that represents the record structure. It's data members form the record fields.

[C#]

```
public class VirtualItem
{
    int index;
```

```
string name;
double someValue;
double otherValue;

public int Index
{
    get
    {
        return index;
    }
    set
    {
        index = value;
    }
}

public string Name
{
    get
    {
        return name;
    }
    set
    {
        name = value;
    }
}

public double SomeValue
{
    get
    {
        return someValue;
    }
    set
    {
        someValue = value;
    }
}

public double OtherValue
{
    get
    {
        return otherValue;
    }
    set
    {
        otherValue = value;
    }
}
```

```
        }
    }
}
```

**[VB]**

```
Public Class VirtualItem
    Private index_Renamed As Integer
    Private name_Renamed As String
    Private someValue_Renamed As Double
    Private otherValue_Renamed As Double

    Public Property Index() As Integer
        Get
            Return index_Renamed
        End Get
        Set
            index_Renamed = Value
        End Set
    End Property
    Public Property Name() As String
        Get
            Return name_Renamed
        End Get
        Set
            name_Renamed = Value
        End Set
    End Property
    Public Property SomeValue() As Double
        Get
            Return someValue_Renamed
        End Get
        Set
            someValue_Renamed = Value
        End Set
    End Property
    Public Property OtherValue() As Double
        Get
            Return otherValue_Renamed
        End Get
        Set
            otherValue_Renamed = Value
        End Set
    End Property
End Class
```

2. Create another class(VirtualList) by implementing [IList](#) and [ITypedList](#) interfaces. This class represents your collection that serves as the data source for the grid grouping control. Refer [CustomCollections](#) to know about how to implement these interfaces.

[C#]

```
public class VirtualList : IList, ITypedList
{
    int virtualCount;
    public VirtualList(int count)
    {
        virtualCount = count;
    }

    // IList Members.
    public bool IsReadOnly
    {
        get
        {
            return true;
        }
    }

    public object this[int index]
    {
        get
        {
            VirtualItem item = new VirtualItem();
            item.Index = index;
            item.Name = "Name" + index.ToString("000000000");
            item.SomeValue = index * 0.873332f;
            item.OtherValue = (293023033 - index) / 8;

            return item;
        }
        set{ }
    }

    // Other IList members.
    .....
    .....

    // ICollection Members.
    .....
    .....
```

```
// IEnumarator Members.  
.....  
.....  
  
// ITypedList Members.  
public PropertyDescriptorCollection  
GetItemProperties(PropertyDescriptor[] listAccessors)  
{  
    PropertyDescriptorCollection pds =  
    TypeDescriptor.GetProperties(typeof(VirtualItem));  
    string[] atts = new string[]{"Index", "Name", "SomeValue",  
    "OtherValue"};  
    return pds.Sort(atts);  
}  
  
public string GetListName(PropertyDescriptor[] listAccessors)  
{  
    return "VirtualList";  
}  
}
```

**[VB]**

```
Public Class VirtualList : Implements IList, ITypedList  
    Private virtualCount As Integer  
  
    Public Sub New(ByVal count_Renamed As Integer)  
        virtualCount = count_Renamed  
    End Sub  
  
    ' IList Members.  
    Public ReadOnly Property IsReadOnly() As Boolean Implements  
    IList.IsReadOnly  
        Get  
            Return True  
        End Get  
    End Property  
  
    Public Default Property Item(ByVal index As Integer) As Object  
        Get  
            Dim item As VirtualItem = New VirtualItem()  
            item.Index = index  
            item.Name = "Name" & index.ToString("00000000")  
            item.SomeValue = index*0.873332f  
            item.OtherValue = (293023033-index)/8  
            Return item  
        End Get  
    End Property
```

```
Set
End Set
End Property

' Other IList Members.
.....
.....

' ICollection Members.
.....
.....

' IEnumerable Members.
.....
.....

' ITypedList Members.
Public Function GetItemProperties(ByVal listAccessors As
PropertyDescriptor()) As PropertyDescriptorCollection Implements
ITypedList.GetItemProperties
    Dim pds As PropertyDescriptorCollection =
TypeDescriptor.GetProperties(GetType(VirtualItem))
    Dim atts As String() = New String() {"Index", "Name",
"SomeValue", "OtherValue"}
    Return pds.Sort(atts)
End Function

Public Function GetListName(ByVal listAccessors As
PropertyDescriptor()) As String Implements ITypedList.GetListName
    Return "VirtualList"
End Function
End Class
```

3. Add a button and listbox to the main form. Clicking the button will create a grid grouping control and load it with the Virtual List. ListBox serves as the Log Window where in you will display the log messages like time elapsed for loading the grid, list of optimizations applied, and so on. Your form will look like the one below at design time.

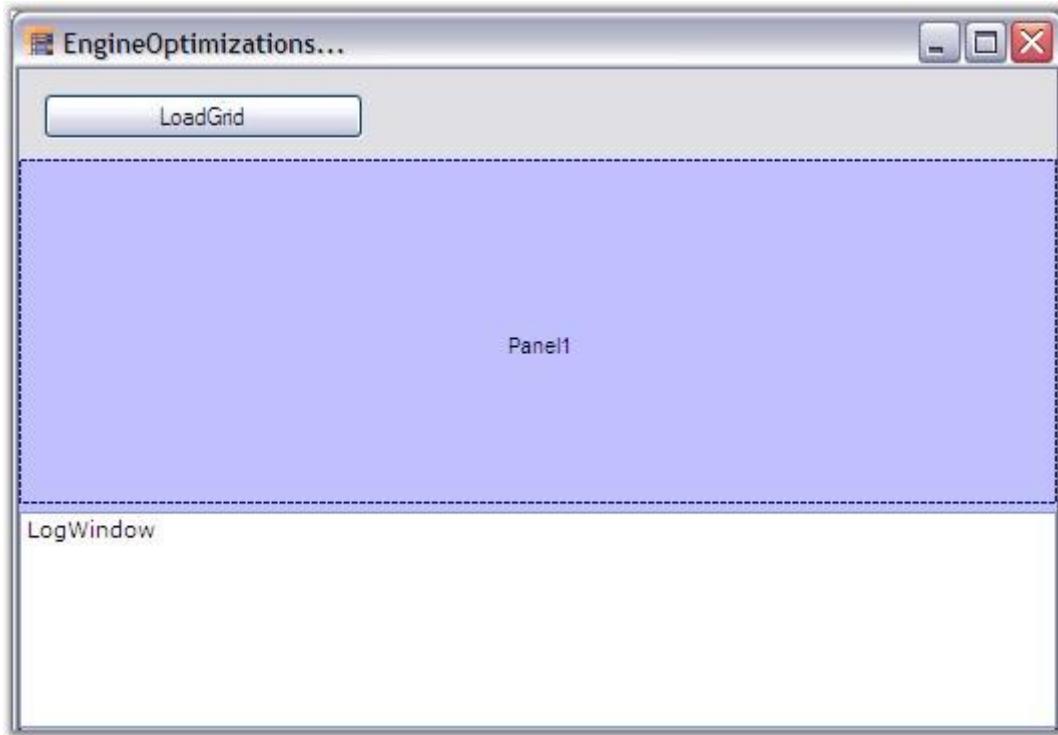


Figure 262: Loading Grid Grouping Control

4. Set up a new engine and specify the optimizations settings required.

[C#]

```
GridEngine schema = new GridEngine();
schema.InvalidateAllWhenListChanged = false;
schema.AllowedOptimizations = EngineOptimizations.All;
schema.CounterLogic = EngineCounters.YAmount;

// Also dependant on CounterLogic = EngineCounters.YAmount.
schema.TableOptions.VerticalPixelScroll = true;
schema.TableOptions.ColumnsMaxLengthStrategy =
GridColumnsMaxLengthStrategy.FirstNRecords;
schema.TableOptions.ColumnsMaxLengthFirstNRecords = 100;
schema.TableOptions.AllowSortColumns = true;
schema.TableDescriptor.AllowEdit = false;
schema.DataSource = new VirtualList(100000);
schema.Reset();
schema.TableDescriptor.Columns["Index"].MaxLength = 10;
```

[VB]

```
Dim schema As GridEngine = New GridEngine()
schema.InvalidateAllWhenListChanged = False
schema.AllowedOptimizations = EngineOptimizations.All
schema.CounterLogic = EngineCounters.YAmount

' Also dependant on CounterLogic = EngineCounters.YAmount.
schema.TableOptions.VerticalPixelScroll = True
schema.TableOptions.ColumnsMaxLengthStrategy =
GridColumnsMaxLengthStrategy.FirstNRecords
schema.TableOptions.ColumnsMaxLengthFirstNRecords = 100
schema.TableOptions.AllowSortColumns = True
schema.TableDescriptor.AllowEdit = False
schema.DataSource = New VirtualList(100000)
schema.Reset()
schema.TableDescriptor.Columns("Index").MaxLength = 10
```

5. Define a method LogMemoryUsage that calculates the amount of memory consumed and displays various optimizations applied to the grouping engine.

[C#]

```
void LogMemoryUsage()
{
    // Force garbage collection and get used memory size.
    GC.Collect();
    System.Threading.Thread.Sleep(10);
    GC.Collect();
    System.Threading.Thread.Sleep(100);
    GC.Collect();
    LogWindow.Items.Add(string.Format("Optimizations for {0}: ",
this.gridGroupingControl1.TableDescriptor.Name));
    LogWindow.Items.Add(string.Format("VirtualMode: {0}, ",
this.gridGroupingControl1.Table.VirtualMode));
    LogWindow.Items.Add(string.Format("WithoutCounter: {0}, ",
this.gridGroupingControl1.Table.WithoutCounter));
    LogWindow.Items.Add(string.Format("RecordsAsDisplayElements: {0},",
", gridGroupingControl1.Table.RecordsAsDisplayElements));

    Process myProcess = Process.GetCurrentProcess();
    double workingSetSizeinKiloBytes = myProcess.WorkingSet64 / 1000;
    string s = "Process's physical memory usage: " +
workingSetSizeinKiloBytes.ToString() + " kb";
    LogWindow.Items.Add(s);
    LogWindow.Items.Add("");
}
```

**[VB.NET]**

```
Private Sub LogMemoryUsage()

    ' Force garbage collection and get used memory size.
    GC.Collect()
    System.Threading.Thread.Sleep(10)
    GC.Collect()
    System.Threading.Thread.Sleep(100)
    GC.Collect()
    LogWindow.Items.Add(String.Format("Optimizations for {0}: ", 
        Me.gridGroupingControl1.TableDescriptor.Name))
    LogWindow.Items.Add(String.Format("VirtualMode: {0}, ", 
        Me.gridGroupingControl1.Table.VirtualMode))
    LogWindow.Items.Add(String.Format("WithoutCounter: {0}, ", 
        Me.gridGroupingControl1.Table.WithoutCounter))
    LogWindow.Items.Add(String.Format("RecordsAsDisplayElements: {0}, ", 
        gridGroupingControl1.Table.RecordsAsDisplayElements))

    Dim myProcess As Process = Process.GetCurrentProcess()
    Dim workingSetSizeinKiloBytes As Double = myProcess.WorkingSet64 / 
        1000
    Dim s As String = "Process's physical memory usage: " &
    workingSetSizeinKiloBytes.ToString() & " kb"
    LogWindow.Items.Add(s)
    LogWindow.Items.Add("")

End Sub
```

6. Handle the ButtonClick event in order to populate the grid when the button is clicked. It also calls LogMemoryUsage method to display the initial optimization settings for the grid - the optimizations for an ungrouped and unsorted grid.

**[C#]**

```
this.button1.Click += new System.EventHandler(this.button1_Click);

private void button1_Click(object sender, EventArgs e)
{
    this.LogWindow.Items.Add("");
    this.LogWindow.Items.Add("Flat, Virtual List with Sorting and
Grouping Enabled.");
    int time = Environment.TickCount;
    Cursor.Current = Cursors.WaitCursor;

    // Load a Grid Grouping control with a new engine.
    gridGroupingControl1 = new GridGroupingControl();
    gridGroupingControl1.BackColor =
```

```
System.Drawing.SystemColors.Window;
    gridGroupingControl1.Dock = System.Windows.Forms.DockStyle.Fill;
    gridGroupingControl1.Name = "gridGroupingControl1";
    gridGroupingControl1.TabIndex = 0;
    gridGroupingControl1.VersionInfo = "3.2.0.0";
    gridGroupingControl1.IntelliMousePanning = true;

this.splitContainer1.Panel1.Controls.Add(this.gridGroupingControl1);
    gridGroupingControl1.Engine = schema;
    gridGroupingControl1.DataSource = new
TestEngineOptimizations.VirtualList(100000);
    gridGroupingControl1.ShowGroupDropArea = true;
    this.Refresh();

    Cursor.Current = Cursors.Arrow;
    this.LogWindow.Items.Add(string.Format("Elapsed Time: {0}",
EnvironmentTickCount - time));
    gridGroupingControl1.Appearance.AnyCell.Font.Facename = "Verdana";
    gridGroupingControl1.Appearance.AnyCell.TextColor =
Color.MidnightBlue;
    gridGroupingControl1.TableOptions.GridVisualStyles =
Syncfusion.Windows.Forms.GridVisualStyles.Office2007Blue;
    gridGroupingControl1.TableOptions.GridLineBorder = new
GridBorder(GridBorderStyle.Solid, Color.FromArgb(227, 239, 255),
GridBorderWeight.Thin);

    // Initial Log Display.
    LogMemoryUsage();
}
```

**[VB.NET]**

```
Private Sub button1_Click(ByVal sender As Object, ByVal e As EventArgs)
Handles button1.Click
    Me.LogWindow.Items.Add("")
    Me.LogWindow.Items.Add("Flat, Virtual List with Sorting and Grouping
Enabled.")
    Dim time As Integer = Environment.TickCount
    Windows.Forms.Cursor.Current = Cursors.WaitCursor

    ' Load a Grid Grouping control with a new engine.
    gridGroupingControl1 = New GridGroupingControl()
    gridGroupingControl1.BackColor = System.Drawing.SystemColors.Window
    gridGroupingControl1.Dock = System.Windows.Forms.DockStyle.Fill
    gridGroupingControl1.Name = "gridGroupingControl1"
    gridGroupingControl1.TabIndex = 0
    gridGroupingControl1.VersionInfo = "3.2.0.0"
```

```
gridGroupingControll.IntelliMousePanning = True
Me.splitContainer1.Panel1.Controls.Add(Me.gridGroupingControll)
gridGroupingControll.Engine = schema
gridGroupingControll.DataSource = New
TestEngineOptimizations.VirtualList(100000)
gridGroupingControll.ShowGroupDropArea = True
Me.Refresh()

Cursor.Current = Cursors.Arrow
Me.LogWindow.Items.Add(String.Format("Elapsed Time: {0}",
EnvironmentTickCount - time))
gridGroupingControll.Appearance.AnyCell.Font.Facename = "Verdana"
gridGroupingControll.Appearance.AnyCell.TextColor =
Color.MidnightBlue
gridGroupingControll.TableOptions.GridVisualStyles =
GridVisualStyles.Office2007Blue
gridGroupingControll.TableOptions.GridLineBorder = New
GridBorder(GridBorderStyle.Solid, Color.FromArgb(227, 239, 255),
GridBorderWeight.Thin)

' Initial Log Display.
LogMemoryUsage()
End Sub
```

7. Handle PropertyChanging event to display the log for every property that is being changed in the grid. This will be raised when you group or sort the grid grouping control and hence you could track the results of these operations (especially the current optimizations) here.

[C#]

```
gridGroupingControll.PropertyChanging += new
DescriptorPropertyChangedEventHandler(gridGroupingControll_PropertyChang
ing);

Timer t = null;
void gridGroupingControll_PropertyChanging(object sender,
DescriptorPropertyChangedEventArgs e)
{
    LogWindow.Items.Add(e.ToString());
    if (t != null)
    {
        t.Tick -= new EventHandler(t_Tick);
        t.Dispose();
    }
    t = new Timer();
    t.Interval = 100;
    t.Tick += new EventHandler(t_Tick);
```

```
t.Start();  
}  
  
private void t_Tick(object sender, EventArgs e)  
{  
    Timer t = (Timer)sender;  
    t.Tick -= new EventHandler(t_Tick);  
    t.Dispose();  
    this.LogMemoryUsage();  
}
```

**[VB.NET]**

```
AddHandler gridGroupingControl1.PropertyChanging, AddressOf  
gridGroupingControl1_PropertyChanging  
  
Private t As Timer = Nothing  
Private Sub gridGroupingControl1_PropertyChanging(ByVal sender As  
Object, ByVal e As DescriptorPropertyChangedEventArgs)  
    LogWindow.Items.Add(e.ToString())  
    If Not t Is Nothing Then  
        RemoveHandler t.Tick, AddressOf t_Tick  
        t.Dispose()  
    End If  
    t = New Timer()  
    t.Interval = 100  
    AddHandler t.Tick, AddressOf t_Tick  
    t.Start()  
End Sub  
  
Private Sub t_Tick(ByVal sender As Object, ByVal e As EventArgs)  
    Dim t As Timer = CType(sender, Timer)  
    RemoveHandler t.Tick, AddressOf t_Tick  
    t.Dispose()  
    Me.LogMemoryUsage()  
End Sub
```

8. Here are the screen shots that show the optimizations applied at different engine states.

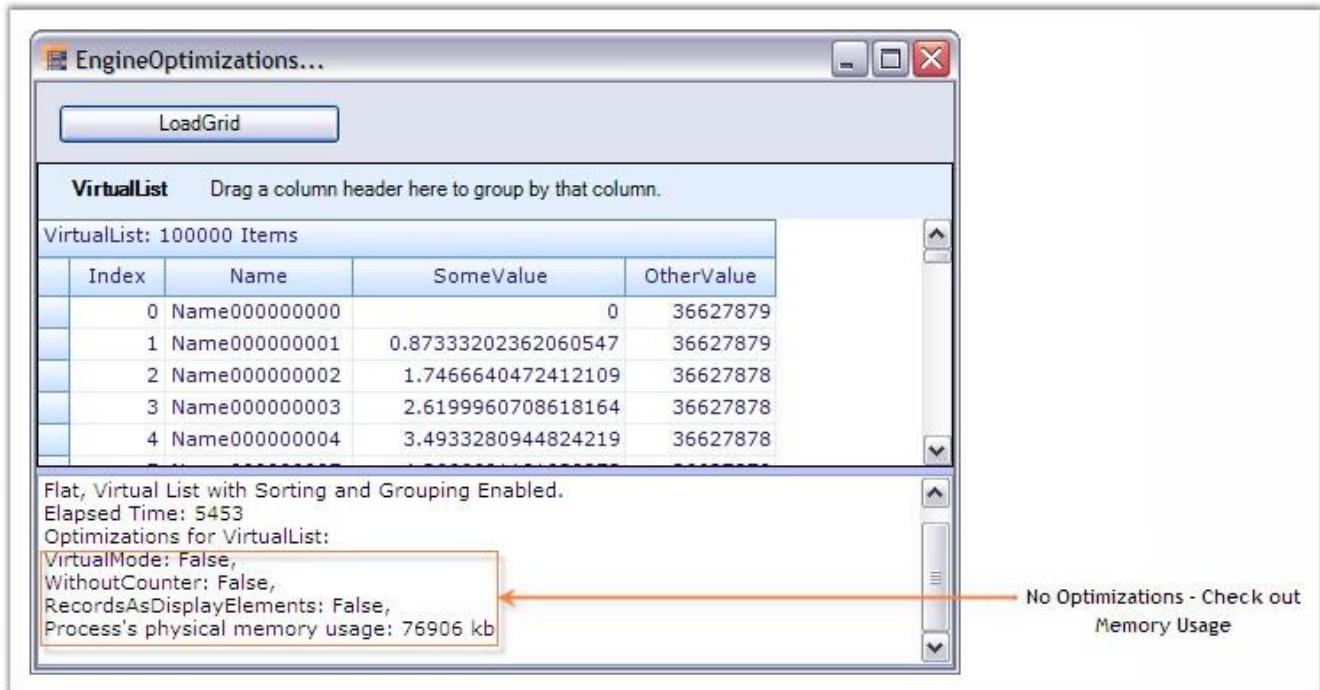


Figure 263: Grid Grouping control with No Optimizations Enabled

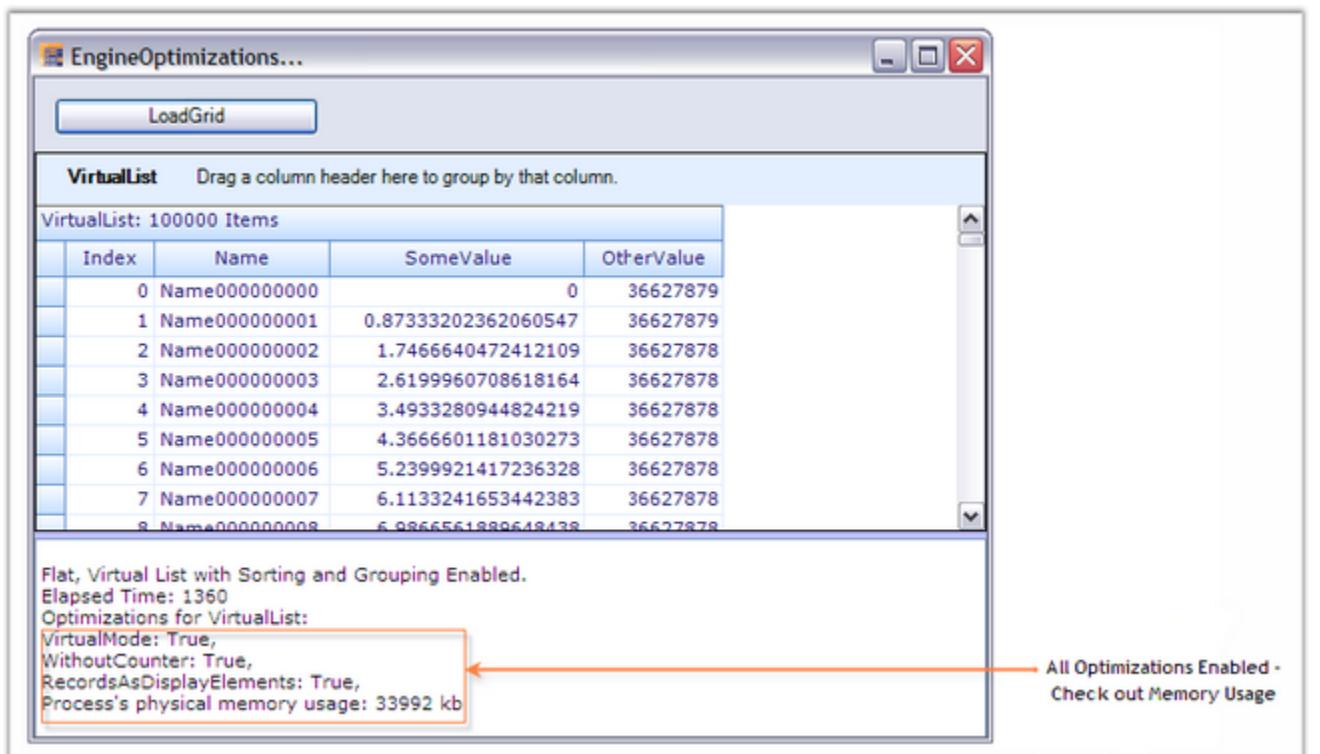
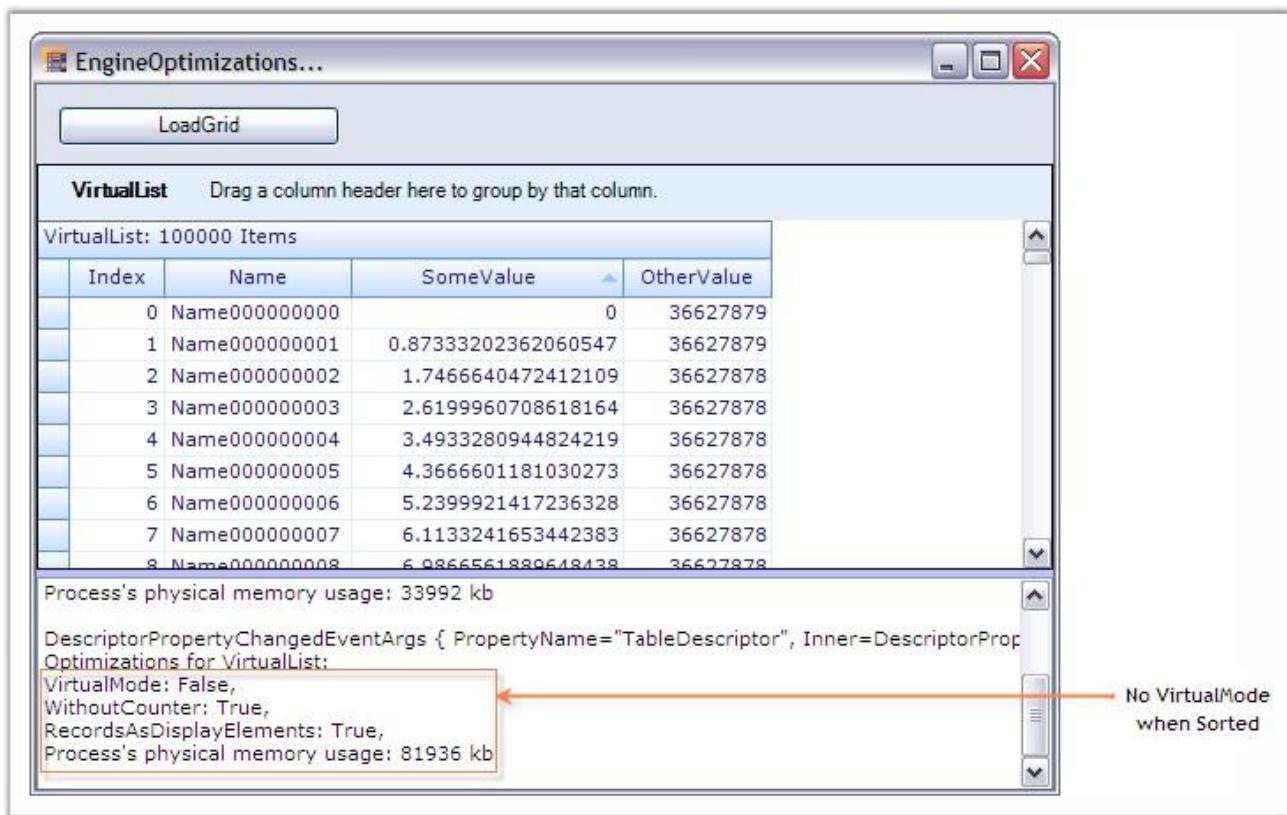


Figure 264: Grid Grouping control at StartUp - LogWindow displays Initial Optimizations Enabled (Optimizations - All)



*Figure 265: Log displays optimizations after Sorting the Grid  
(Virtual Mode disabled)*

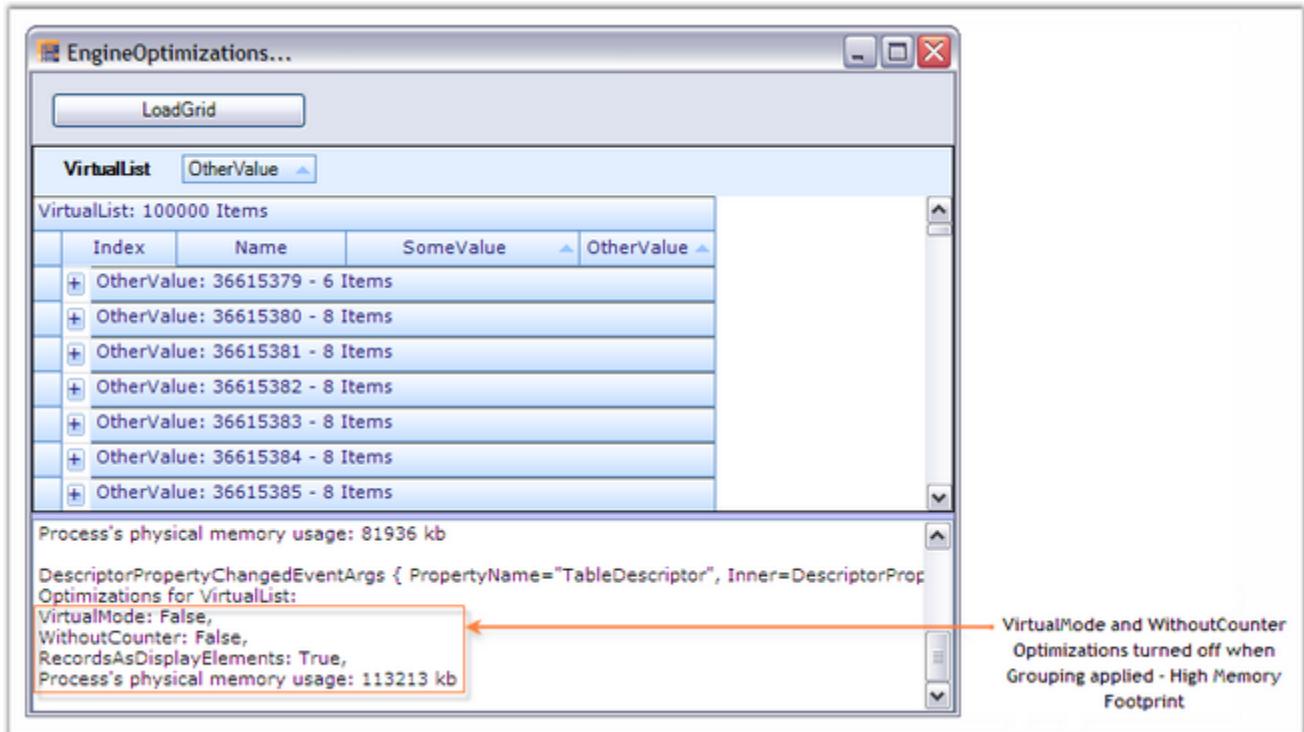


Figure 266: LogWindow displays Optimizations for Grid Grouping control with Sorting Applied  
(Both Virtual Mode and WithoutCounter optimizations disabled)

#### 4.3.4.1.2 ListChanged Performance

When a ListChange is detected, the grouping engine has to update the grid records accordingly. Every record change may affect its sort position, group dependency and the summaries. The engine should take care of all these things and should also invalidate the counters that are being affected with respect to the ListChange. The easiest way to accomplish this would be invalidating the whole display and repainting all the rows. But this will have a big impact on the grid performance in worst cases. For example, in case only one record is really changed and this change has not affect sort order and summaries, it requires to repaint only one record. Instead the engine will repaint the whole display.

GridEngine provides options to handle this type of scenarios using which it will track which expression fields and summary columns depend on changes to a field, which fields affect group dependency or sort position. Based on these findings, it will choose the most efficient way to update the engine's internal object to keep up with the ListChanged events.

#### Example

The example features this optimization that handles the case when changes to the record only affect single cells in the grid. The sample updates two columns (and summaries) in one thousand records in a timer event every 50 milliseconds while at the same time keeping CPU usage low. In this sample the engine detects that changes to the Freight or Employee field do not have any impact on counters in the engines object. It checks if the record is visible in the current view and if that is the case saves the record and field information for painting. The painting of the cell is delayed until **gridGroupingControl.Update** is called or until the time specified with **GridGroupingControl.UpdateDisplayFrequency** elapsed.

In that sample example you can also check out throttling the display updates. The **UpdateDisplayFrequency** is initially set to 0 which means that the `timer_tick` method in the form calls `grid.Update` to force pending updates but you can also specify any amount of milliseconds in the Property Grid for this property and watch the grid react slower or faster to changes and update the display while the `ListChanged` events still come in at the same pace.

If you click on the Freight column to sort records by values of this field then the strategy how the Grid Grouping control updates the internal structure and the display has to change. Now, every change to the Freight column can possibly affect the sort position of a record. The moment you click on the column header and sort by that field the engine will have this field singled out for more detailed inspection when a `ListChanged` event is received.

If a change is detected to the Freight field in a `DataRow` the engine will now check the new value against the value of the previous record and next record. If the value is greater or equal than the previous records value and smaller or equal than the next records value no further action is required other than repainting the cell. But if the new value does not fit in between these records the sort position of the records needs to be recalculated. The binary tree structures within the engine allow quick removal and reinsertion of the record at the correct sort position and the engine will raise **SourceListRecordChanging** and **SourceListRecordChanged** events that indicate that the records sort position is about to change and has changed.

Updating the display with changes in sort position of a record is much more demanding than simply repainting a cell. One or multiple records need to be shifted up or down. The easiest way is to invalidate the whole display and repaint all rows. This is how the Grid Grouping control handles this case by default. You can change this default behavior by specifying how the grid should update the display with the **InsertRemoveBehavior** and **SortPositionChangeBehavior** properties. By setting these properties to

**ListChangedInsertRemoveBehavior.ScrollWithImmediateUpdate** you can instruct the engine not to repaint the whole screen. Instead the engine will now determine the area affected by a sort position change and use the `ScrollWindow` API to shift records up and down and only repaint the one record that was really changed. This can have big impact if you have a large grid and repainting the whole display is expensive. This can increase the speed by the number of rows that are visible. Only one row needs to be repainted instead of repainting all rows.

The example also shows more optimized calculation of summaries. By default the engine uses binary trees and caches values in them. When changes to a field in a record are detected that affects a summary then the nodes in the binary tree are marked dirty. This is a  $2xO(\log n)$  operation that is needed to mark nodes dirty and later again recalculate the summaries. The ManualTotalSummary sample demonstrates this by using a different approach. If you do not care about more complicate summaries such as minimum, maximum, distinct count or median and if you know the delta of each value change, then you can keep a total value cached in the parent group and manually apply that delta to this parent groups summary value. Now you have a linear **O(1)** operation instead of the more costly binary tree updates. With that change, the sample can now deal nicely with 1000 updates in 100 ms and 2 summary columns being changed with each update.

#### **GridListChangedInsertRemoveBehavior** Enum

Defines the values for the properties InsertRemoveBehavior and SortPositionChangedBehavior.

##### **InvalidateVisible**

It will keep engine in synchronization with ListChanged notifications and then invalidate rows on screen, below affected row.

##### **InvalidateAll**

It will simply set TableDirty = true and the engine won't try to keep anything in synchronization at that time.

##### **ScrollWithImmediateUpdate**

It will keep engine in synchronization and use ScrollWindow, to scroll, window contents or adjust top row index if changes occurred before current visible row.



**Note:** For Complete Code for this example, refer the following Browser sample:

<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Performance\Manual Total Summary Demo

#### **Implementation**

- The implementation uses a custom summary class named ManualTotalSummary. This is a manual summary class which can be updated immediately using the difference between old and new value in a ListChanged event. The Total property calculates the summaries for groups and table manually by looping through each group and record and summing up the values of the changed field. It provides faster updates on summaries by applying a delta between the old and new value when a record is changed.

[C#]

```
public class ManualTotalSummary
{
    double total;
    bool dirty = true;
    Group group;
    int fieldIndex = -1;

    public ManualTotalSummary(Group g, string field)
        : this(g, g.ParentTableDescriptor.Fields[field])
    {
    }

    public ManualTotalSummary(Group g, FieldDescriptor field)
    {
        this.Field = field;
        this.Group = g;
    }
    .....
    .....

    public double Total
    {
        get
        {
            if (dirty)
            {
                CalculateTotal();
                this.dirty = false;
            }
            return this.total;
        }
        set
        {
            this.total = value;
        }
    }

    void CalculateTotal()
    {
        total = 0;
        if (group.Details is RecordsDetails)
        {
            foreach (Record r in group.Records)
```

```
{  
    object obj = r.GetValue(field);  
    if (obj != null && !(obj is DBNull))  
    {  
        double d = Convert.ToDouble(obj);  
        total += d;  
    }  
}  
}  
  
else  
{  
    foreach (Group g in group.Groups)  
    {  
        IManualTotalSummaryArraySource tsa = g as  
IManualTotalSummaryArraySource;  
        ManualTotalSummary mt =  
tsa.GetManualTotalSummaryArray() [this.FieldIndex];  
        if (mt == null)  
            mt = new ManualTotalSummary(g, field);  
        double d = mt.Total;  
        total += d;  
    }  
}  
}  
  
public void ApplyDelta(Element r, bool isObsoleteRecord, bool  
isAddedRecord, ChangedFieldInfo ci)  
{  
    if (Dirty)  
        return;  
  
    ManualTotalSummary mt = this;  
  
    if (isObsoleteRecord)  
    {  
        if (ci.OldValue != null && !(ci.OldValue is DBNull))  
            mt.Total -= Convert.ToDouble(ci.OldValue);  
    }  
    else if (isAddedRecord)  
    {  
        if (ci.NewValue != null && !(ci.NewValue is DBNull))  
            mt.Total += Convert.ToDouble(ci.NewValue);  
    }  
    else  
        mt.Total += ci.Delta;  
}
```

```
}
```

**[VB.NET]**

```
Public Class ManualTotalSummary

    Private total_Renamed As Double
    Private dirty_Renamed As Boolean = True
    Private group_Renamed As Group
    Private fieldIndex_Renamed As Integer = -1

    Public Sub New(ByVal g As Group, ByVal field_Renamed As String)
        Me.New(g, g.ParentTableDescriptor.Fields(field_Renamed))
    End Sub

    Public Sub New(ByVal g As Group, ByVal field_Renamed As FieldDescriptor)
        Me.Field = field_Renamed
        Me.Group = g
    End Sub
    .....
    .....

    Public Property Total() As Double
        Get
            If dirty_Renamed Then
                CalculateTotal()
                Me.dirty_Renamed = False
            End If
            Return Me.total_Renamed
        End Get
        Set
            Me.total_Renamed = Value
        End Set
    End Property

    Private Sub CalculateTotal()
        total_Renamed = 0

        If TypeOf group_Renamed.Details Is RecordsDetails Then
            For Each r As Record In group_Renamed.Records
                Dim obj As Object = r.GetValue(field_Renamed)
                If Not obj Is Nothing AndAlso Not(TypeOf obj Is DBNull)
                Then
                    Dim d As Double = Convert.ToDouble(obj)
                    total_Renamed += d
                End If
            Next
        End If
    End Sub
End Class
```

```

        End If
    Next r
Else
    For Each g As Group In group_Renamed.Groups
        Dim tsa As IManualTotalSummaryArraySource =
        CType(IIf(TypeOf(g) Is IManualTotalSummaryArraySource, g,
        Nothing), IManualTotalSummaryArraySource)
        Dim mt As ManualTotalSummary =
        tsa.GetManualTotalSummaryArray()(Me.FieldIndex)
        If mt Is Nothing Then
            mt = New ManualTotalSummary(g, field_Renamed)
        End If
        Dim d As Double = mt.Total
        total_Renamed += d
    Next g
End If
End Sub

Public Sub ApplyDelta(ByVal r As Element, ByVal isObsoleteRecord As
Boolean, ByVal isAddedRecord As Boolean, ByVal ci As
ChangedFieldInfo)
    If Dirty Then
        Return
    End If
    Dim mt As ManualTotalSummary = Me
    If isObsoleteRecord Then
        If Not ci.OldValue Is Nothing AndAlso Not(TypeOf(ci.OldValue)
Is DBNull) Then
            mt.Total -= Convert.ToDouble(ci.OldValue)
        End If
    Else If isAddedRecord Then
        If Not ci.NewValue Is Nothing AndAlso Not(TypeOf(ci.NewValue)
Is DBNull) Then
            mt.Total += Convert.ToDouble(ci.NewValue)
        End If
    Else
        mt.Total += ci.Delta
    End If
End Sub
End Class

```

- **ManualTotalSummary** class makes use of **ManualTotalSummaryTable** class which derives **GridTable** to calculate the new total. The **ManualTotalSummaryTable** class overrides **OnRecordChanged** event in order to track the record changes and keeps track of the old and new values of the **ChangedField**. For each entry in **ManualTotalSummaryTable.TotalSummaries**, a **ManualTotalSummary** will be created.

[C#]

```
public class ManualTotalSummaryTable : GridTable
{
    public ManualTotalSummaryTable(TableDescriptor tableDescriptor,
Table parentRelationTable)
        : base((GridTableDescriptor)tableDescriptor,
(GridTable)parentRelationTable)
    {
    }

    ArrayList totalSummaries = new ArrayList();
    public ArrayList TotalSummaries
    {
        get
        {
            return this.totalSummaries;
        }
        set
        {
            this.totalSummaries = value;
        }
    }
    .....
    .....

    protected override void OnRecordChanged(Element r, bool
isObsoleteRecord, bool isAddedRecord)
    {
        TableDescriptor td = TableDescriptor;
        Group g = r.ParentGroup;
        while (g is IManualTotalSummaryArraySource)
        {
            OnGroupSummaryInvalidated(new GroupEventArgs(g));

            IManualTotalSummaryArraySource tsa = g as
IManualTotalSummaryArraySource;
            foreach (ChangedFieldInfo ci in this.ChangedFieldsArray)
            {
                ManualTotalSummary mt =
tsa.GetManualTotalSummaryArray()[ci.FieldIndex];
                if (mt != null)
                    mt.ApplyDelta(r, isObsoleteRecord, isAddedRecord, ci);
            }
            g = g.ParentGroup;
        }
    }
}
```

```
}
```

**[VB.NET]**

```
Public Class ManualTotalSummaryTable : Inherits GridTable

    Public Sub New(ByVal tableDescriptor As TableDescriptor, ByVal parentRelationTable As Table)
        MyBase.New(CType(tableDescriptor, GridTableDescriptor),
                  CType(parentRelationTable, GridTable))
    End Sub

    Private totalSummaries_Renamed As ArrayList = New ArrayList()
    Public Property TotalSummaries() As ArrayList
        Get
            Return Me.totalSummaries_Renamed
        End Get
        Set
            Me.totalSummaries_Renamed = Value
        End Set
    End Property
    .....
    .....

    Protected Overrides Sub OnRecordChanged(ByVal r As Element, ByVal isObsoleteRecord As Boolean, ByVal isAddedRecord As Boolean)
        Dim td As TableDescriptor = TableDescriptor
        Dim g As Group = r.ParentGroup
        Do While TypeOf g Is IManualTotalSummaryArraySource
            OnGroupSummaryInvalidated(New GroupEventArgs(g))

            Dim tsa As IManualTotalSummaryArraySource = CType(IIf(TypeOf g
                Is IManualTotalSummaryArraySource, g, Nothing),
                IManualTotalSummaryArraySource)
            For Each ci As ChangedFieldInfo In Me.ChangedFieldsArray
                Dim mt As ManualTotalSummary =
                    tsa.GetManualTotalSummaryArray()(ci.FieldIndex)
                If Not mt Is Nothing Then
                    mt.ApplyDelta(r, isObsoleteRecord, isAddedRecord, ci)
                End If
            Next ci
            g = g.ParentGroup
        Loop
    End Sub
End Class
```

- A Grid Grouping control is setup with options to display the summary cells in caption. Also enable the optimizations required. Use **InvalidateAll** option for InsertRemoveBehavior and SortPositionChangedBehavior properties when many records change sort position for a short time. Use **ScrollWithImmediateUpdate** if ScrollWindow should be called to minimize painting when sort position of limited number of records is changed. Grid Grouping control is detached from the Currency Manager and access the list directly to solely rely on ListChanged events.

[C#]

```
// Optimization code.  
// 0 if Manual updates only from timer_tick.  
this.gridGroupingControl1.UpdateDisplayFrequency = 0;  
this.gridGroupingControl1.UseDefaultsForFasterDrawing = true;  
this.gridGroupingControl1.CounterLogic = EngineCounters.YAmount;  
this.gridGroupingControl1.AllowedOptimizations =  
EngineOptimizations.DisableCounters |  
EngineOptimizations.RecordsAsDisplayElements;  
this.gridGroupingControl1.CacheRecordValues = false;  
  
this.gridGroupingControl1.InsertRemoveBehavior =  
GridListChangedInsertRemoveBehavior.ScrollWithImmediateUpdate;  
this.gridGroupingControl1.SortPositionChangedBehavior =  
GridListChangedInsertRemoveBehavior.ScrollWithImmediateUpdate;  
  
this.gridGroupingControl1.BindToCurrencyManager = false;  
  
// Enable Caption Summaries.  
gridGroupingControl1.TableDescriptor.ChildGroupOptions.ShowCaptionSummaryCells = true;  
gridGroupingControl1.TableDescriptor.ChildGroupOptions.ShowSummaries = true;  
gridGroupingControl1.TableDescriptor.ChildGroupOptions.CaptionSummaryRow = "Caption";
```

[VB .NET]

```
' Optimization code.  
' 0 if Manual updates only from timer_tick.  
Me.gridGroupingControl1.UpdateDisplayFrequency = 0  
Me.gridGroupingControl1.UseDefaultsForFasterDrawing = True  
Me.gridGroupingControl1.CounterLogic = EngineCounters.YAmount  
Me.gridGroupingControl1.AllowedOptimizations =  
EngineOptimizations.DisableCounters Or  
EngineOptimizations.RecordsAsDisplayElements  
Me.gridGroupingControl1.CacheRecordValues = False  
  
Me.gridGroupingControl1.InsertRemoveBehavior =
```

```
GridListChangedInsertRemoveBehavior.ScrollWithImmediateUpdate  
Me.gridGroupingControll.SortPositionChangedBehavior =  
GridListChangedInsertRemoveBehavior.ScrollWithImmediateUpdate  
  
Me.gridGroupingControll.BindToCurrencyManager = False  
  
' Enable Caption Summaries.  
gridGroupingControll.TableDescriptor.ChildGroupOptions.ShowCaptionSummaryCells = True  
gridGroupingControll.TableDescriptor.ChildGroupOptions.ShowSummaries = True  
gridGroupingControll.TableDescriptor.ChildGroupOptions.CaptionSummaryRow = "Caption"
```

- Setup **ManualTotalSummary** for the columns **Freight** and **EmployeeID**. The **ManualTotalSummary.Total** value will be retrieved and displayed in a summary or caption cell in the **QueryCellStyleInfo** event handler. It tracks the changes in the sort positions of the columns Freight and EmployeeID by handling **PropertyChanged** event.

[C#]

```
ManualTotalSummaryTable tb =  
(ManualTotalSummaryTable)this.gridGroupingControll.Table;  
tb.TotalSummaries.Add("Freight");  
tb.TotalSummaries.Add("EmployeeID");  
tb.TableDirty = true;  
  
this.gridGroupingControll.QueryCellStyleInfo += new  
GridTableCellStyleInfoEventHandler(gridGroupingControll_QueryCellStyleI  
nfo);  
  
private void gridGroupingControll_QueryCellStyleInfo(object sender,  
GridTableCellStyleInfoEventArgs e)  
{  
    Element el = e.TableCellIdentity.DisplayElement;  
    ManualTotalSummaryTable table = el.ParentTable as  
ManualTotalSummaryTable;  
  
    if (table == null)  
        return;  
  
    if (Element.IsCaption(el))  
    {  
        if (e.Style.TableCellIdentity.ColumnIndex > 3)  
        {  
            // You need to provide manually, the code to look up the  
            // summaries you want to display here.  
            // e.TableCellIdentity.Column and
```

```

e.TableCellIdentity.SummaryColumn will be null.
    // You can get the column as follows.
    GridColumnDescriptor column =
gridGroupingControl1.TableModel.GetHeaderColumnDescriptorAt(e.TableCell
Identity.ColIndex);

    if (column != null &&
table.TotalSummaries.IndexOf(column.MappingName) != -1)
    {
        int index =
column.TableDescriptor.Fields.IndexOf(column.FieldDescriptor);
        IManualTotalSummaryArraySource tsa = (el is Group ? el
: el.ParentGroup) as IManualTotalSummaryArraySource;
        ManualTotalSummary tm =
tsa.GetManualTotalSummaryArray()[index];
        e.Style.CellValue = tm.Total;
        e.Style.CellValueType = typeof(double);
        e.Style.Format = "0.00";
    }
    // By using that column you could try and identify the
summary that should be displayed in this cell.
}
}
else if (el is GridSummaryRow)
{
    // You can get the column as follows.
    GridColumnDescriptor column =
this.gridGroupingControl1.TableModel.GetHeaderColumnDescriptorAt(e.Tabl
eCellIdentity.ColIndex);
    if (column != null &&
table.TotalSummaries.IndexOf(column.MappingName) != -1)
    {
        int index =
column.TableDescriptor.Fields.IndexOf(column.FieldDescriptor);
        IManualTotalSummaryArraySource tsa = (el is Group ? el
: el.ParentGroup) as IManualTotalSummaryArraySource;
        ManualTotalSummary tm =
tsa.GetManualTotalSummaryArray()[index];
        e.Style.CellValue = tm.Total;
        e.Style.CellValueType = typeof(double);
        e.Style.Format = "0.00";
    }
    // By using that column you could try and identify the summary
that should be displayed in this cell.
}
}

```

[VB.NET]

```
Dim tb As ManualTotalSummaryTable =
DirectCast(Me.gridGroupingControl1.Table, ManualTotalSummaryTable)
tb.TotalSummaries.Add("Freight")
tb.TotalSummaries.Add("EmployeeID")
tb.TableDirty = True

AddHandler Me.gridGroupingControl1.QueryCellStyleInfo, AddressOf
gridGroupingControl1_QueryCellStyleInfo

Private Sub gridGroupingControl1_QueryCellStyleInfo(ByVal sender As
Object, ByVal e As GridTableCellStyleEventArgs)
    Dim el As Element = e.TableCellIdentity.DisplayElement
    Dim table As ManualTotalSummaryTable = TryCast(el.ParentTable,
ManualTotalSummaryTable)

    If table Is Nothing Then
        Return
    End If

    If Element.IsCaption(el) Then
        If e.Style.TableCellIdentity.ColumnIndex > 3 Then

            ' You need to provide manually, the code to look up the
            summaries you want to display here.
            ' e.TableCellIdentity.Column and
            e.TableCellIdentity.SummaryColumn will be null.
            ' You can get the column as follows.
            Dim column As GridColumnDescriptor =

gridGroupingControl1.TableModel.GetHeaderColumnDescriptorAt(e.TableCell
Identity.ColumnIndex)
            If column IsNot Nothing AndAlso
table.TotalSummaries.IndexOf(column.MappingName) <> -1 Then
                Dim index As Integer =
column.TableDescriptor.Fields.IndexOf(column.FieldDescriptor)
                Dim tsa As IManualTotalSummaryArraySource =
TryCast((IIf(TypeOf el Is Group, el, el.ParentGroup)),
IManualTotalSummaryArraySource)
                Dim tm As ManualTotalSummary =
tsa.GetManualTotalSummaryArray()(index)
                e.Style.CellValue = tm.Total
                e.Style.CellValueType = GetType(Double)
                e.Style.Format = "0.00"

            ' By using that column you could try and identify the
            summary that should be displayed in this cell.
        End If
    End If
End Sub
```

```

        End If
    ElseIf TypeOf e1 Is GridSummaryRow Then

        ' You can get the column as follows.
        Dim column As GridColumnDescriptor =
Me.gridGroupingControll.TableModel.GetHeaderColumnDescriptorAt(e.TableC
ellIdentity.ColumnIndex)
        If column IsNot Nothing AndAlso
table.TotalSummaries.IndexOf(column.MappingName) <> -1 Then
            Dim index As Integer =
column.TableDescriptor.Fields.IndexOf(column.FieldDescriptor)
            Dim tsa As IManualTotalSummaryArraySource =
TryCast((IIf(TypeOf el Is Group, el, el.ParentGroup)),
IManualTotalSummaryArraySource)
            Dim tm As ManualTotalSummary =
tsa.GetManualTotalSummaryArray()(index)
            e.Style.CellValue = tm.Total
            e.Style.CellValueType = GetType(Double)
            e.Style.Format = "0.00"

            By using that column you could try and identify the summary
that should be displayed in this cell.
        End If
    End If
End Sub

```

- Enable highlighting of the cells changed for all the columns.

**[C#]**

```

for (int c = 0; c < gridGroupingControll.TableDescriptor.Columns.Count;
c++)
this.gridGroupingControll.TableDescriptor.Columns[c].AllowBlink = true;
this.gridGroupingControll.BlinkTime = 100;

```

**[VB.NET]**

```

Do While c < gridGroupingControll.TableDescriptor.Columns.Count
Me.gridGroupingControll.TableDescriptor.Columns(c).AllowBlink = True
c += 1
Loop
Me.gridGroupingControll.BlinkTime = 100

```

- To optimize the performance, the grid is updated manually (UpdateDisplayFrequency = 0) at regular intervals. A timer is used to keep track of the duration of the time periods. The code to track the changes in Freight and EmployeeID columns and to update the grid rows is written inside the timer\_tick event handler where the update is done manually by

making a call to **gridGroupingControl.Update** method. Timer Interval is set to 100 which means that there would be a update for every 100 ms. This implementation pushes in the pending updates every 100 ms and updates 1000 records each time.

[C#]

```
void timer_tick(object sender, EventArgs e)
{
    GridTableDescriptor td = this.gridGroupingControl1.TableDescriptor;
    ManualTotalSummaryTable tb =
    (ManualTotalSummaryTable)this.gridGroupingControl1.Table);
    int i = 0;
    using (MeasureTime.Measure("Form1.timer_tick"))
    {
        int count = 1000;
        if (this.gridGroupingControl1.SortPositionChangedBehavior ==
GridListChangedInsertRemoveBehavior.ScrollWithImmediateUpdate)
        {
            if (sortedByFreight || gridGroupingControl1.TestDeleteRecords
|| gridGroupingControl1.TestInsertRecords ||
gridGroupingControl1.TestChangeGroup)

                // When sort position is changed, this is much more demanding,
let's do less records then.
                count = 200;

            if (sortedByEmployeeID)

                // Each update will cause records being shifted around, so
let's do even less records. You can also check out InvalidateAll
option instead above.
                count = 50;
        }
        for (i = 0; i < count; i++)
        {
            ManualTotalSummaries.DataSet1.OrdersRow dr;

            // Insert Records.
            if (gridGroupingControl1.TestInsertRecords)
            {
                if (i % 10 == 0)
                {
                    dr = northwindDataSet1.Orders.NewOrdersRow();
                    dr.CustomerID = i.ToString() + (j++).ToString();
                    dr.EmployeeID = i;
                    dr.Freight = i / 10;
                    dr.ShipVia = 0;
                }
            }
        }
    }
}
```

```
        dr.Table.Rows.Add(dr);
        continue;
    }
}
if (northwindDataSet1.Orders.Rows.Count == 0)
{
    this.gridGroupingControll1.Update();
    return;
}
int newIndex = rnd.Next(northwindDataSet1.Orders.Rows.Count);
dr = northwindDataSet1.Orders[newIndex];

// Delete Records.
if (gridGroupingControll1.TestDeleteRecords)
{
    if (i % 12 == 0)
    {
        dr.Delete();
        continue;
    }
}
// Change Records.
// Freight
decimal freight = (decimal)dr.Freight +
Math.Round((decimal)rnd.Next(-100, 100) / 10000, 2);
int employeeID = (int)(rnd.NextDouble() * 1000);
dr.BeginEdit();
decimal oldFreight = dr.Freight;
dr.Freight = freight;
dr.EmployeeID = employeeID;
if (gridGroupingControll1.TestChangeGroup)
{
    // Change Group Category.
    if (i == 10)
    {
        tb.AddChangedField(new ChangedFieldInfo(td, "ShipVia",
dr.ShipVia, 0));
        dr.ShipVia = 0;
    }
}
// Fires ListChanged event.
dr.EndEdit();
}
// Optionally manually flush changes.
if (this.gridGroupingControll1.UpdateDisplayFrequency == 0)
    this.gridGroupingControll1.Update();
```

```
    }  
}
```

**[VB.NET]**

```
Private Sub timer_tick(ByVal sender As Object, ByVal e As EventArgs)  
    Dim td As GridTableDescriptor =  
        Me.gridGroupingControl1.TableDescriptor  
    Dim tb As ManualTotalSummaryTable =  
        (CType(Me.gridGroupingControl1.Table, ManualTotalSummaryTable))  
    Dim i As Integer = 0  
  
    MeasureTime.Measure("Form1.timer_tick")  
    Try  
        Dim count As Integer = 1000  
  
        If gridGroupingControl1.SortPositionChangedBehavior =  
            GridListChangedInsertRemoveBehavior.ScrollWithImmediateUpdate  
        Then  
            If sortedByFreight OrElse  
                gridGroupingControl1.TestDeleteRecords OrElse  
                gridGroupingControl1.TestInsertRecords OrElse  
                gridGroupingControl1.TestChangeGroup Then  
  
                ' When sort position is changed, this is much more  
                demanding, let's do less records then.  
                count = 200  
            End If  
  
            If sortedByEmployeeID Then  
  
                ' Each update will cause records being shifted around, so  
                let's do even less records. You can also check out  
                InvalidateAll option instead above.  
                count = 50  
            End If  
        End If  
  
        i = 0  
        Do While i < count  
            Dim dr As ManualTotalSummaries.DataSet1.OrdersRow  
  
            ' Insert Records.  
            If gridGroupingControl1.TestInsertRecords Then  
                If i Mod 10 = 0 Then  
                    dr = northwindDataSet1.Orders.NewOrdersRow()  
                    dr.CustomerID = i.ToString() & (j += 1).ToString()  
                End If  
            End If  
        Loop
```

```
        dr.EmployeeID = i
        dr.Freight = i / 10
        dr.ShipVia = 0
        dr.Table.Rows.Add(dr)
        GoTo Continue1
    End If
End If

If northwindDataSet1.Orders.Rows.Count = 0 Then
    Me.gridGroupingControl1.Update()
    Return
End If

Dim newIndex As Integer =
rnd.Next(northwindDataSet1.Orders.Rows.Count)
dr = northwindDataSet1.Orders(newIndex)

' Delete Records.
If gridGroupingControl1.TestDeleteRecords Then
    If i Mod 12 = 0 Then
        dr.Delete()
        GoTo Continue1
    End If
End If

' Change Records.
' Freight
Dim freight As Decimal = CDec(dr.Freight) +
Math.Round(CDec(rnd.Next(-100, 100)) / 10000, 2)
Dim employeeID As Integer = CInt(rnd.NextDouble() * 1000)
dr.BeginEdit()

Dim oldFreight As Decimal = dr.Freight
dr.Freight = freight
dr.EmployeeID = employeeID

If gridGroupingControl1.TestChangeGroup Then
    ' Change Group Category.
    If i = 10 Then
        tb.AddChangedField(New ChangedFieldInfo(td, "ShipVia",
dr.ShipVia, 0))
        dr.ShipVia = 0
    End If
End If

' Fires ListChanged event.
```

```
dr.EndEdit()
i += 1
Continue1:
Loop

' Optionally manually flush changes.
If Me.gridGroupingControl1.UpdateDisplayFrequency = 0 Then
    Me.gridGroupingControl1.Update()
End If
Finally
End Try
End Sub
```

- It should also take care of the **UnboundFields** whose values are usually dependent on changes to other fields. If unbound fields are used, you can tell the engine which are the fields the unbound field is dependent on, by setting the **ReferencedFields** property. When ReferencedFields is set and the engine detects changes to the unbound field, it will then automatically also mark the field as changed. This subsequently can affect sort order, group attachment, and so on.

**[C#]**

```
// Add Unbound field 'ShipVia_CompanyName'.
gridGroupingControl1.TableDescriptor.UnboundFields.Add("ShipVia_Company
Name");

// Inform the engine about dependency on ShipVia of this field.
gridGroupingControl1.TableDescriptor.UnboundFields["ShipVia_CompanyName
"].ReferencedFields = "ShipVia";
```

**[VB .NET]**

```
' Add Unbound field 'ShipVia_CompanyName'.
gridGroupingControl1.TableDescriptor.UnboundFields.Add("ShipVia_Company
Name")

' Inform the engine about dependency on ShipVia of this field.
gridGroupingControl1.TableDescriptor.UnboundFields("ShipVia CompanyName
").ReferencedFields = "ShipVia"
```

Here is a sample screen shot.

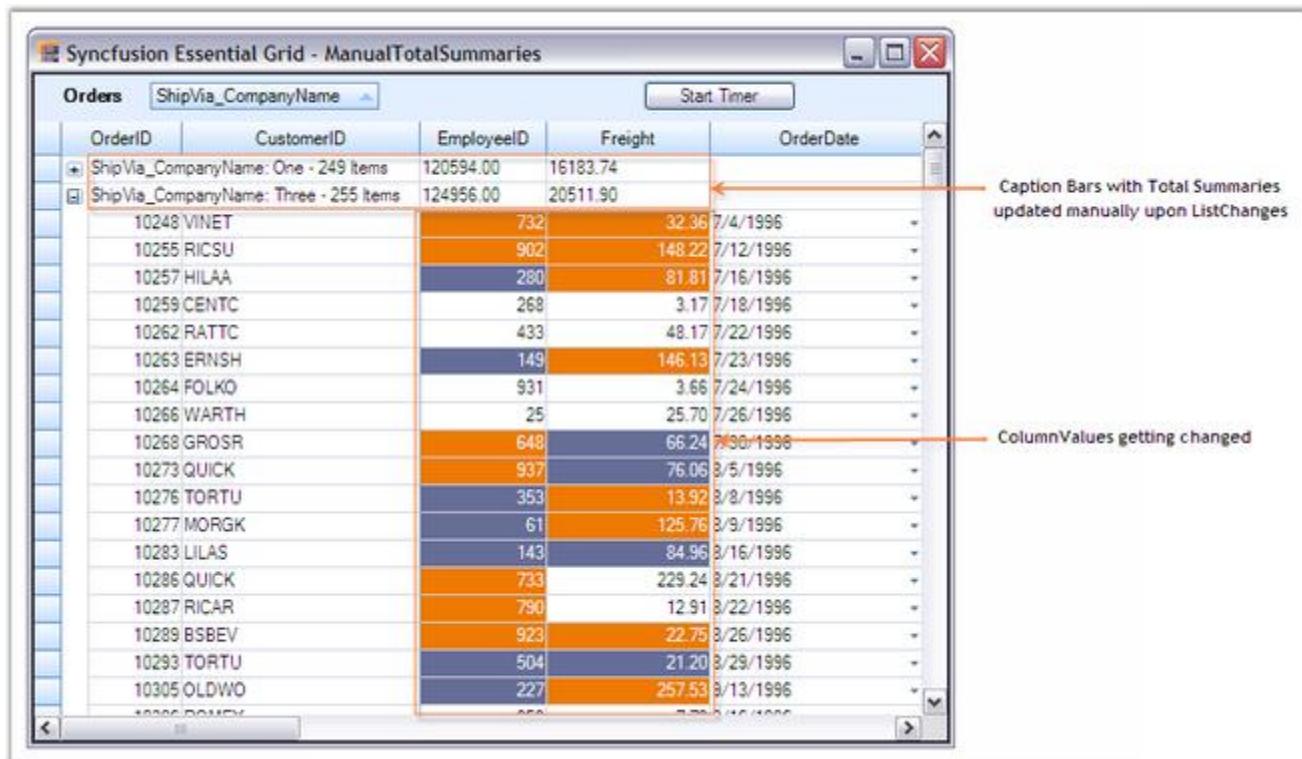


Figure 267: ListChange Performance

#### 4.3.4.1.3 High Frequency Updates

This section discusses an example that lets you make high frequency updates in efficient manner. It shows sort position changes, inserting and removing of records from a timer event. At start up, the grid grouping control is sorted by Column "1" and changes to fields in that column affects the sort position of a record. Also, the background colors of the cells in records are dependant on the value in column "1". This dependency is specified with the **ReferencedFields** property. The changes are highlighted for a short period of time after a change was detected.

##### ReferencedFields Property

**ReferencedFields** property, as the name implies, saves a list of field names referred by a given field. The engine will use these fields in the **ListChanged** event to determine which cells to update when a change was made in an underlying field.

It is very user interactive options and provides options to test the grid performance by enabling/disabling grouping, sorting, filtering in the midst of heavy updates. It also allows you to change the timer frequency that controls the throughput i.e., the number of updates per unit time. At run time, you can also vary the amount of time the changes are highlighted.



**Note:** For Complete Code for this example, refer the following Browser sample:

**<Install Location>\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Grouping.Windows\Samples\2.0\Performance\Manual Total Summary Demo**

### Implementation

This example demonstrates the frequent updates that occur in random cells across the grid grouping control, while keeping the CPU usage at a minimum level. A timer changes cells in short intervals, inserts and removes rows. When you run the sample you also need to open up the TaskManager in order to notice the CPU usage while the sample runs. You should be able to start up multiple instances without slowing down your machine.

1. Set up a Grid Grouping control and load it with some random data. Enable the optimizations as required.

[C#]

```
GridGroupingControl gridGroupingControl1 = new GridGroupingControl();
gridGroupingControl1.VerticalThumbTrack = true;
gridGroupingControl1.HorizontalThumbTrack = true;
gridGroupingControl1.TableOptions.VerticalPixelScroll = true;
gridGroupingControl1.DataSource = GetRandomDataTable();
this.gridGroupingControl1.ShowGroupDropArea = true;

// Use less memory for internal binary tree structures.
gridGroupingControl1.CounterLogic = EngineCounters.YAmount;
gridGroupingControl1.AllowedOptimizations =
EngineOptimizations.DisableCounters |
EngineOptimizations.RecordsAsDisplayElements;

// Use faster GDI drawing.
gridGroupingControl1.UseDefaultsForFasterDrawing = true;

// Skip Currency Manager.
gridGroupingControl1.BindToCurrencyManager = false;

// Immediate update after each ListChanged event.
gridGroupingControl1.UpdateDisplayFrequency = 1;
```

```
// Scroll Window will cause immediate update.  
gridGroupingControl1.InsertRemoveBehavior =  
GridListChangedInsertRemoveBehavior.ScrollWithImmediateUpdate;  
gridGroupingControl1.SortPositionChangedBehavior =  
GridListChangedInsertRemoveBehavior.ScrollWithImmediateUpdate;  
  
// InsertRemoveBehaior/SortPositionChangedBehavior takes effect only  
when InvalidateAll is set to false.  
gridGroupingControl1.InvalidateAllWhenListChanged = false;
```

**[VB.NET]**

```
Dim gridGroupingControl1 As New GridGroupingControl()  
gridGroupingControl1.VerticalThumbTrack = True  
gridGroupingControl1.HorizontalThumbTrack = True  
gridGroupingControl1.TableOptions.VerticalPixelScroll = True  
gridGroupingControl1.DataSource = GetRandomDataTable()  
Me.gridGroupingControl1.ShowGroupDropArea = True  
  
' Use less memory for internal binary tree structures.  
gridGroupingControl1.CounterLogic = EngineCounters.YAmount  
gridGroupingControl1.AllowedOptimizations =  
EngineOptimizations.DisableCounters Or  
EngineOptimizations.RecordsAsDisplayElements  
  
' Use faster GDI drawing.  
gridGroupingControl1.UseDefaultsForFasterDrawing = True  
  
' Skip Currency Manager.  
gridGroupingControl1.BindToCurrencyManager = False  
  
' Immediate update after each ListChanged event.  
gridGroupingControl1.UpdateDisplayFrequency = 1  
  
' Scroll Window will cause immediate update.  
gridGroupingControl1.InsertRemoveBehavior =  
GridListChangedInsertRemoveBehavior.ScrollWithImmediateUpdate  
gridGroupingControl1.SortPositionChangedBehavior =  
GridListChangedInsertRemoveBehavior.ScrollWithImmediateUpdate  
  
' InsertRemoveBehaior/SortPositionChangedBehavior takes effect only  
when InvalidateAll is set to false.  
gridGroupingControl1.InvalidateAllWhenListChanged = False
```

2. Set **AllowBlink** to true for all the columns in order to enable highlighting cells for a short period of time when a change is detected. **Engine.AddBaseStylesForBlinking** method is used to add base styles for the customization of the appearance of blink cells. Initialize the

base styles for various blink states. **PrepareViewStyleInfo** is handled to set the custom base style for a newly added record. A cell change is highlighted by checking its **BlinkState**. BlinkState indicates whether the cell's value is increased or reduced or if the record has been recently added. If its state is either **Increased** or **Reduced**, its back color and text colors are changed.

**[C#]**

```
// Allow Blinking for all the columns.  
// Highlight up and down ticks.  
gridGroupingControl1.BlinkTime = 700;  
for (int i = 1; i <= 20; i++)  
gridGroupingControl1.TableDescriptor.Columns[i.ToString()].AllowBlink =  
true;  
gridGroupingControl1.Engine.AddBaseStylesForBlinking();  
gridGroupingControl1.BaseStyles[GridEngine.BlinkIncreased].StyleInfo.Text  
Color = Color.White;  
gridGroupingControl1.BaseStyles[GridEngine.BlinkReduced].StyleInfo.Text  
Color = Color.White;  
  
gridGroupingControl1.Engine.BaseStyles.Add("CustomStyle");  
gridGroupingControl1.BaseStyles["CustomStyle"].StyleInfo.TextColor =  
Color.Black;  
gridGroupingControl1.BaseStyles["CustomStyle"].StyleInfo.BackColor =  
Color.White;  
  
// PrepareViewStyleInfo  
void gridGroupingControl1_TableControlPrepareViewStyleInfo(object  
sender, GridTableControlPrepareViewStyleInfoEventArgs e)  
{  
    GridTableCellStyleInfo style =  
(GridTableCellStyleInfo)e.Inner.Style;  
    BlinkState bs =  
gridGroupingControl1.GetBlinkState(style.TableCellIdentity);  
    if (bs != BlinkState.None)  
    {  
        if (bs == BlinkState.NewRecord)  
        {  
            e.Inner.Style.BaseStyle = "CustomStyle";  
        }  
    }  
}
```

**[VB.NET]**

```
' Allow Blinking for all the columns.  
' Highlight up and downticks.
```

```
gridGroupingControl1.BlinkTime = 700
For i As Integer = 1 To 20
    gridGroupingControl1.TableDescriptor.Columns(i.ToString()).AllowBlink =
        k = True
Next i
gridGroupingControl1.Engine.AddBaseStylesForBlinking()
gridGroupingControl1.BaseStyles(GridEngine.BlinkIncreased).StyleInfo.TextColor =
    Color.White
gridGroupingControl1.BaseStyles(GridEngine.BlinkReduced).StyleInfo.TextColor =
    Color.White

gridGroupingControl1.Engine.BaseStyles.Add("CustomStyle")
gridGroupingControl1.BaseStyles("CustomStyle").StyleInfo.TextColor =
    Color.Black
gridGroupingControl1.BaseStyles("CustomStyle").StyleInfo.BackColor =
    Color.White

' PrepareViewStyleInfo
Private Sub
    gridGroupingControl1_TableControlPrepareViewStyleInfo(ByVal sender
        As Object, ByVal e As GridTableControlPrepareViewStyleInfoEventArgs)
    Dim style As GridTableCellStyleInfo = CType(e.Inner.Style,
        GridTableCellStyleInfo)
    Dim bs As BlinkState =
        gridGroupingControl1.GetBlinkState(style.TableCellIdentity)
    If bs <> BlinkState.None Then
        If bs = BlinkState.NewRecord Then
            e.Inner.Style.BaseStyle = "CustomStyle"
        End If
    End If
End Sub
```

3. A timer event is handled to insert and remove the records. This results in frequent list changes at regular intervals.

[C#]

```
bool skipTimer = false;
private void timer_Tick(object sender, EventArgs e)
{
    if (skipTimer)
        return;
    timerCount++;
    try
    {
        for (int i = 0; i < m_numUpdatesPerTick; i++)
        {
```

```

// Application.DoEvents();
int recNum = rand.Next(table.Rows.Count - 1);
int rowNum = recNum + 1;
int col = rand.Next(16) + 1;
int colNum = col + 1;
DataRow drow = table.Rows[recNum];
if (!(drow[col] is DBNull))
    drow[col] = (int) (Convert.ToDouble(drow[col]) *
(rand.Next(50) / 100.0f + 0.8));
}

// Insert or remove a row.
if (insertRemoveCount == 0)
    return;
if (toggleInsertRemove > 0 && (timerCount %
insertRemoveModulus) == 0)
{
    icount = ++icount % (toggleInsertRemove * 2);
    shouldInsert = icount < toggleInsertRemove;
    if (shouldInsert)
    {
        for (int ri = 0; ri < insertRemoveCount; ri++)
        {
            int recNum = 5;
            int next = rand.Next(100);
            object[] row = new
object[] {"H"+ti.ToString("00000"),next+1,next+2,next+3,next+4,next+5,next+6,next+7,next+8,next+9,
next+10,next+11,next+12,next+13,next+14,next+15,next+16,next+17,next+18,next+19,next+20};
            ti++;
            DataRow drow = table.NewRow();
            drow.ItemArray = row;
            table.Rows.InsertAt(drow, recNum);
        }
    }
    else
    {
        for (int ri = 0; ri < insertRemoveCount; ri++)
        {
            int recNum = 5;
            int rowNum = recNum + 1;

            // Underlying data structure (this could be a data
table or whatever structure
            // you use behind a virtual grid).
            if (table.Rows.Count > 10)

```

```
        table.Rows.RemoveAt(recNum);
    }
}
}
}
finally
{
}
}
```

**[VB.NET]**

```
Private skipTimer As Boolean = False

Private Sub timer_Tick(ByVal sender As Object, ByVal e As EventArgs)
If skipTimer Then
    Return
End If

timerCount += 1

Try
    Dim i As Integer = 0
    Do While i < m_numUpdatesPerTick

        ' Application.DoEvents();
        Dim recNum As Integer = rand.Next(table.Rows.Count - 1)
        Dim rowNum As Integer = recNum + 1
        Dim col As Integer = rand.Next(16) + 1
        Dim colNum As Integer = col + 1
        Dim drow As DataRow = table.Rows(recNum)
        If Not(TypeOf drow(col) Is DBNull) Then
            drow(col) = CInt(Convert.ToDouble(drow(col)) *
                (rand.Next(50) / 100.0f + 0.8))
        End If
        i += 1
    Loop

    ' Insert or remove a row.
    If insertRemoveCount = 0 Then
        Return
    End If

    If toggleInsertRemove > 0 AndAlso (timerCount Mod
insertRemoveModulus) = 0 Then
        icount += 1
    End If
End Sub
```

```

icount = icount Mod (toggleInsertRemove * 2)
shouldInsert = icount < toggleInsertRemove

If shouldInsert Then
    Dim ri As Integer = 0

    Do While ri < insertRemoveCount
        Dim recNum As Integer = 5

        Dim next_Renamed As Integer = rand.[next](100)
        Dim row As Object() = New Object(){{"H"+ti.ToString("00000"),next_Renamed+1,next_Renamed+2,next_Renamed+3, next_Renamed+4,
        next_Renamed+5,next_Renamed+6,
        next_Renamed+7,next_Renamed+8,next_Renamed+9,next_Renamed+10,
        next_Renamed+11,next_Renamed+12,next_Renamed+13,next_Renamed+14,
        next_Renamed+15,next_Renamed+16,next_Renamed+17,
        next_Renamed+18,next_Renamed+19,next_Renamed+20}}
        ti += 1
        Dim drow As DataRow = table.NewRow()
        drow.ItemArray = row
        table.Rows.InsertAt(drow, recNum)
        ri += 1
    Loop
Else
    Dim ri As Integer = 0
    Do While ri < insertRemoveCount
        Dim recNum As Integer = 5
        Dim rowNum As Integer = recNum + 1

        ' Underlying data structure (this could be a data table
        ' or whatever structure
        ' you use behind a virtual grid).

        If table.Rows.Count > 10 Then
            table.Rows.RemoveAt(recNum)
        End If
        ri += 1
    Loop
End If
End If

Finally
End Try
End Sub

```

4. **QueryCellStyleInfo** is handled to enable coloring of the cells. The background colors of the cells in the records are dependent on the column "1" values. This dependency is specified using the **Referenced Fields** property. To make user friendly, you can use a CheckBox control to enable or disable this coloring. Hook this event if the checked state of the CheckBox is true; unhook otherwise.

**[C#]**

```

Color[] colors = new Color[] { Color.FromArgb( 0x85, 0xbf, 0x75 ),
Color.FromArgb( 0xb4, 0xe7, 0xf2 ),
Color.FromArgb( 0xff, 0xbf, 0x34 ), Color.FromArgb( 0x82, 0x2e, 0x1b ),
Color.FromArgb( 0x3a, 0x86, 0x7e ), };

void gridGroupingControl1_QueryCellStyleInfo(object sender,
GridTableViewCellEventArgs e)
{
    GridTableViewCellInfo style = (GridTableViewCellInfo)e.Style;
    if (e.TableCellIdentity.TableCellType ==
GridTableViewCellType.RecordFieldCell
    || e.TableCellIdentity.TableCellType ==
GridTableViewCellType.AlternateRecordFieldCell)
    {
        if
(e.TableCellIdentity.Column.FieldDescriptor.Field.PropertyType ==
typeof(string))
            return;
        // Get the value from column 1 and color all cells in record
based on this value.
        Record r =
e.Style.TableCellIdentity.DisplayElement.GetRecord();
        object value = r.GetValue("1");
        int v = Convert.ToInt32(value) % colors.Length;
        e.Style.BackColor = colors[v];
    }
}

// CheckBox event to enable or disable cell coloring.
private void checkBoxColor_CheckedChanged(object sender,
System.EventArgs e)
{
if (this.checkBoxColor.Checked)
{
    // Callback for dynamically coloring cells.
    gridGroupingControl1.QueryCellStyleInfo += new
GridTableViewCellInfoEventHandler(gridGroupingControl1_QueryCellStyleI
nfo);
}
}

```

```
// The color of these cells depends on value of cell 1. If engines
ListChanged handler
    // detects a change to column 1, it should also automatically
    // repaint the dependant columns.
    for (int i = 2; i <= 20; i++)

gridGroupingControl1.TableDescriptor.Fields[i.ToString()].ReferencedFile
lds = "1";
}
else
{
    gridGroupingControl1.QueryCellStyleInfo == new
GridTableCellStyleInfoEventHandler(gridGroupingControl1_QueryCellStyleI
nfo);
    gridGroupingControl1.TableDescriptor.Fields.LoadDefault();
}
this.gridGroupingControl1.Refresh();
}
```

**[VB.NET]**

```
Private colors As Color() = New Color() {Color.FromArgb(&H85, &HBF,
&H75), Color.FromArgb(&HB4, &HE7, &HF2), Color.FromArgb(&FFF, &HBF,
&H34), Color.FromArgb(&H82, &H2E, &H1B), Color.FromArgb(&H3A, &H86,
&H7E) }

Private Sub gridGroupingControl1_QueryCellStyleInfo(ByVal sender As
Object, ByVal e As GridTableViewCellEventArgs)
Dim style As GridTableViewCellStyleInfo = CType(e.Style,
GridTableViewCellStyleInfo)
If e.TableCellIdentity.TableCellType =
GridTableCellType.RecordFieldCell OrElse
e.TableCellIdentity.TableCellType =
GridTableCellType.AlternateRecordFieldCell Then
    If e.TableCellIdentity.Column.FieldDescriptor.FieldPropertyType Is
GetType(String) Then
        Return
    End If

    ' Get the value from column 1 and color all cells in record based on
    this value.
    Dim r As Record =
e.Style.TableCellIdentity.DisplayElement.GetRecord()
    Dim value As Object = r.GetValue("1")
    Dim v As Integer = Convert.ToInt32(value) Mod colors.Length
    e.Style.BackColor = colors(v)
End If
End Sub
```

```
Private Sub checkBoxColor_CheckedChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles checkBoxColor.CheckedChanged
If Me.checkBoxColor.Checked Then

    ' Callback for dynamically coloring cells.
    AddHandler gridGroupingControl1.QueryCellStyleInfo, AddressOf
    gridGroupingControl1_QueryCellStyleInfo

    ' The color of these cells depends on value of cell 1. If engines
    ListChanged handler
    ' detects a change to column 1, it should also automatically repaint
    the dependant columns.
    For i As Integer = 2 To 20
        gridGroupingControl1.TableDescriptor.Fields(i.ToString()).Referenced
        Fields = "1"
    Next i
Else
    RemoveHandler gridGroupingControl1.QueryCellStyleInfo, AddressOf
    gridGroupingControl1_QueryCellStyleInfo
    gridGroupingControl1.TableDescriptor.Fields.LoadDefault()
End If
Me.gridGroupingControl1.Refresh()
End Sub
```

5. Add three more CheckBoxes to include options to enable or disable Grouping, Sorting and Filtering at runtime. Handle their CheckedChanged events to add the code for adding and removing groups, sorted columns and record filters. For example, if the checked state of groupCheckBox is true, group the table against a column. When its checked state is turned to false, simply ungroup the table.

[C#]

```
// Sorting Option.
private void checkBoxSorting_CheckedChanged(object sender,
System.EventArgs e)
{
    if (this.checkBoxSorting.Checked)
    {
        gridGroupingControl1.TableDescriptor.SortedColumns.Clear();
        gridGroupingControl1.TableDescriptor.SortedColumns.Add("1");
        gridGroupingControl1.TableDescriptor.SortedColumns.Add("2");
    }
    else
    {
        gridGroupingControl1.TableDescriptor.SortedColumns.Clear();
    }
}
```

```
    this.gridGroupingControl1.Refresh();
}

// Grouping Option.
private void checkBoxGrouping_CheckedChanged(object sender,
System.EventArgs e)
{
    if (this.checkBoxGrouping.Checked)
    {
        gridGroupingControl1.TableDescriptor.GroupedColumns.Clear();
        gridGroupingControl1.TableDescriptor.GroupedColumns.Add("1");
        this.gridGroupingControl1.Table.ExpandAllGroups();
    }
    else
    {
        gridGroupingControl1.TableDescriptor.GroupedColumns.Clear();
    }
    this.gridGroupingControl1.Refresh();
}

// Filter Option.
private void checkBoxFilter_CheckedChanged(object sender,
System.EventArgs e)
{
    if (this.checkBoxFilter.Checked)
    {
        gridGroupingControl1.TableDescriptor.RecordFilters.Clear();

        // Gets the filter expression from a Text Box.

gridGroupingControl1.TableDescriptor.RecordFilters.Add(this.textBoxFilter.Text);
    }
    else
    {
        gridGroupingControl1.TableDescriptor.RecordFilters.Clear();
    }
    this.gridGroupingControl1.Refresh();
}
```

**[VB.NET.NET]**

```
' Sorting Option.
Private Sub checkBoxSorting_CheckedChanged(ByVal sender As Object,
 ByVal e As System.EventArgs) Handles checkBoxSorting.CheckedChanged
 If Me.checkBoxSorting.Checked Then
```

```
gridGroupingControl1.TableDescriptor.SortedColumns.Clear()
gridGroupingControl1.TableDescriptor.SortedColumns.Add("1")
gridGroupingControl1.TableDescriptor.SortedColumns.Add("2")
Else
    gridGroupingControl1.TableDescriptor.SortedColumns.Clear()
End If
Me.gridGroupingControl1.Refresh()
End Sub

' Grouping Option.
Private Sub checkBoxGrouping_CheckedChanged(ByVal sender As Object,
 ByVal e As System.EventArgs) Handles checkBoxGrouping.CheckedChanged
If Me.checkBoxGrouping.Checked Then
    gridGroupingControl1.TableDescriptor.GroupedColumns.Clear()
    gridGroupingControl1.TableDescriptor.GroupedColumns.Add("1")
    gridGroupingControl1.Table.ExpandAllGroups()
Else
    gridGroupingControl1.TableDescriptor.GroupedColumns.Clear()
End If
Me.gridGroupingControl1.Refresh()
End Sub

' Filtering Option.
Private Sub checkBoxFilter_CheckedChanged(ByVal sender As Object, ByVal
e As System.EventArgs) Handles checkBoxFilter.CheckedChanged
If Me.checkBoxFilter.Checked Then
    gridGroupingControl1.TableDescriptor.RecordFilters.Clear()
    gridGroupingControl1.TableDescriptor.RecordFilters.Add(Me.textBox
Filter.Text)
Else
    gridGroupingControl1.TableDescriptor.RecordFilters.Clear()
End If
Me.gridGroupingControl1.Refresh()
End Sub
```

6. Two TrackBar controls are used to change the frequencies of the Timer and BlinkTime. The frequencies that are set by the end user are integrated into the grid grouping control in their respective TrackBarScroll event handlers.

[C#]

```
// To change the Blink Time Frequency.
private void trackBarBlinkFrequency_Scroll(object sender,
System.EventArgs e)
{
    this.gridGroupingControl1.BlinkTime =
this.trackBarBlinkFrequency.Value*100;
```

```
if (this.gridGroupingControl1.BlinkTime == 0)
    this.labelBlinkTime.Text = String.Format("Disabled.");
else
    this.labelBlinkTime.Text = String.Format("{0} milliseconds.", 
gridGroupingControl1.BlinkTime);
    this.gridGroupingControl1.Refresh();
}

// To change the Timer Frequency.
private void trackBarTimer_Scroll(object sender, System.EventArgs e)
{
    if (this.trackBarTimer.Value == 0)
    {
        timer.Enabled = false;
        this.labelTimerInterval.Text = String.Format("Timer
disabled.");
    }
    else
    {
        timer.Interval =
1000/(this.trackBarTimer.Value*trackBarTimer.Value);
        timer.Enabled = true;
        this.labelTimerInterval.Text = String.Format("Every {0}
milliseconds.", timer.Interval);
    }
}
```

**[VB.NET]**

```
' To change the Blink Time Frequency
Private Sub trackBarBlinkFrequency_Scroll(ByVal sender As Object, ByVal
e As System.EventArgs) Handles trackBarBlinkFrequency.Scroll
    Me.gridGroupingControl1.BlinkTime = Me.trackBarBlinkFrequency.Value
    * 100
    If Me.gridGroupingControl1.BlinkTime = 0 Then
        Me.labelBlinkTime.Text = String.Format("Disabled.")
    Else
        Me.labelBlinkTime.Text = String.Format("{0} milliseconds.", 
gridGroupingControl1.BlinkTime)
    End If
    Me.gridGroupingControl1.Refresh()
End Sub

' To change the Timer Frequency.
Private Sub trackBarTimer_Scroll(ByVal sender As Object, ByVal e As
System.EventArgs) Handles trackBarTimer.Scroll
    If Me.trackBarTimer.Value = 0 Then
```

```

        timer.Enabled = False
Me.labelTimerInterval.Text = String.Format("Timer disabled.")
Else
    timer.Interval = 1000 / (Me.trackBarTimer.Value *
trackBarTimer.Value)
    timer.Enabled = True
    Me.labelTimerInterval.Text = String.Format("Every {0}
milliseconds.", timer.Interval)
End If
End Sub

```

7. Given below is a sample screen shot. While running the sample, apply grouping, sorting and filtering and then check for the CPU time usage in the TaskManager to detect the grid performance.

The screenshot shows a Windows application window titled "TraderGridTest...". Inside, there is a 16-column grid labeled "RandomData" with a header "Drag a column header here to group by that column.". The grid contains various numerical values. Below the grid is a configuration panel:

- Timer Frequency:** A slider set to "Every 27 milliseconds."
- Filter:** A checked checkbox "Enable Filter" with the condition "[2] > 88" and a "Apply" button.
- Grouping:** An unchecked checkbox "Enable Grouping".
- Sorting:** A checked checkbox "Enable Sorting".
- Coloring:** An unchecked checkbox "Enable Coloring".
- Blink Frequency:** A slider set to "1300 milliseconds."

*Figure 268: High Frequency Updates in Grid Grouping Control*

#### 4.3.4.1.4 Grouping Performance

This section focuses a sample that lets you check the performance of the grid grouping control by toggling various options that can affect the speed of the grid. The different options include Sort and Categorize the records, Calculating MaximumColumnWidth, CustomSorting and MultiThreading (in case if a multiprocessor system is available).



**Note:** For Code, refer the following Browser sample:

`<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Performance\Grouping Performance Demo`

The following is the list of the options used.

##### Sort and Categorize

This option will enable grouping and sorting by assigning a group and sort order.

##### UseDataGridViewSort

It uses the class GroupingSortList to wrap the DataView with IBindingList. It also implements IGroupingList interface. This allows performing the sort on the data view directly instead of relying on the grouping engine to perform sort.

##### CalculateMaximumColumnWidth

When enabled, the maximum number of characters found in record field cells is calculated for columns. This will be used in re sizing the columns to optimal width. Affects TableDescriptor.AllowCalculateMaxColumnWidth property.

##### MultiThreading

When set to true, this option will allow multithreading. It allows you to calculate the count in a separate thread when all records are categorized. Affects Table.AllowThreading property. Enable this only on true multiprocessor machines otherwise systems calculating counts in separate thread will slow categorization down.

## **ListChanging Options**

It also includes options to insert, remove and modify the records in the data source. All the changes will be immediately updated manually by making a call to grid.Update method.

## **UseScrollWindow**

When enabled, inserting and removing cells will be optimized by scrolling window contents and only invalidating new cells. If set to false, it results in repainting of the whole display. Affects TableControl.OptimizeInsertRemoveCells property.

## **ExpandAll/CollapseAll**

Using these options, you can track the time taken to expand / collapse all the groups and memory usage too.

After enabling the options required, click the LoadGrid button. This will then check for the options requested and apply those options before painting the grid. After loading, it also displays a log to print various performance measures like time taken to paint the grid, physical memory usage, etc. The log will continue to display these performance measure at every instant the grid options are changed.

Given below is a sample screen shot.

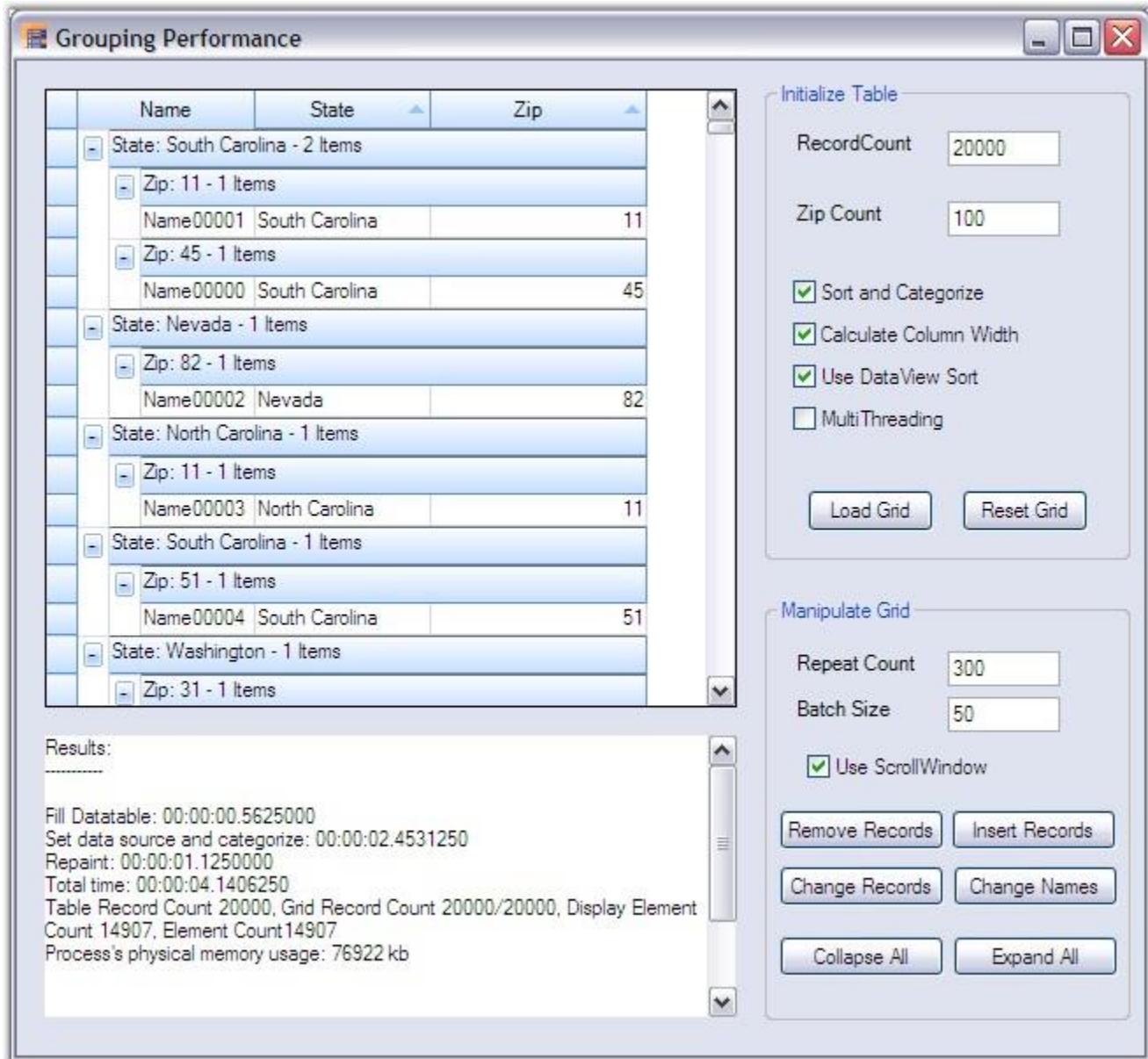


Figure 269: Checking the Grouping Performance in the Grid Grouping Control

#### 4.3.4.1.5 IList Grouping Performance

The **IList** binded to the **GridGroupingControl** has been implemented with an optimization process for grouping columns to improve the performance.

Grouping a column which has **Ilist** binded reduces the time taken to refresh the control after grouping. The grouping performance has been improved with huge data loaded

Set **OptimizeListGroupingPerformance** to **true** to enable grouping optimization over the **Ilist** data source.

The following code illustrates how to enable grouping optimization.

[C#]

```
private void Form1_Load(object sender, EventArgs e)
{
    gridGroupingControl1.OptimizedListGrouping = true;
}
```

[VB]

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    GridGroupingControl1.OptimizedListGrouping = true
End Sub
```

### **Enable Real Time Updates**

The **OptimizeListGroupingPerformance** method has to be called to enable real time updates with the data source from the **GridGroupingControl**

The following code illustrates how to enable real time updates.

[C#]

```
void gridGroupingEmployee_SourceListListChanged(object sender,
TableListChangedEventArgs e)
{
    this.gridGroupingEmployee.OptimizeListGroupingPerformance(sender, e);
}
```

[VB]

```
Private Sub gridGroupingEmployee_SourceListListChanged(ByVal
```

```
    sender As Object, ByVal e As TableListChangedEventArgs)
Me.gridGroupingEmployee.OptimizeIListGroupingPerformance(sender, e)
End Sub
```

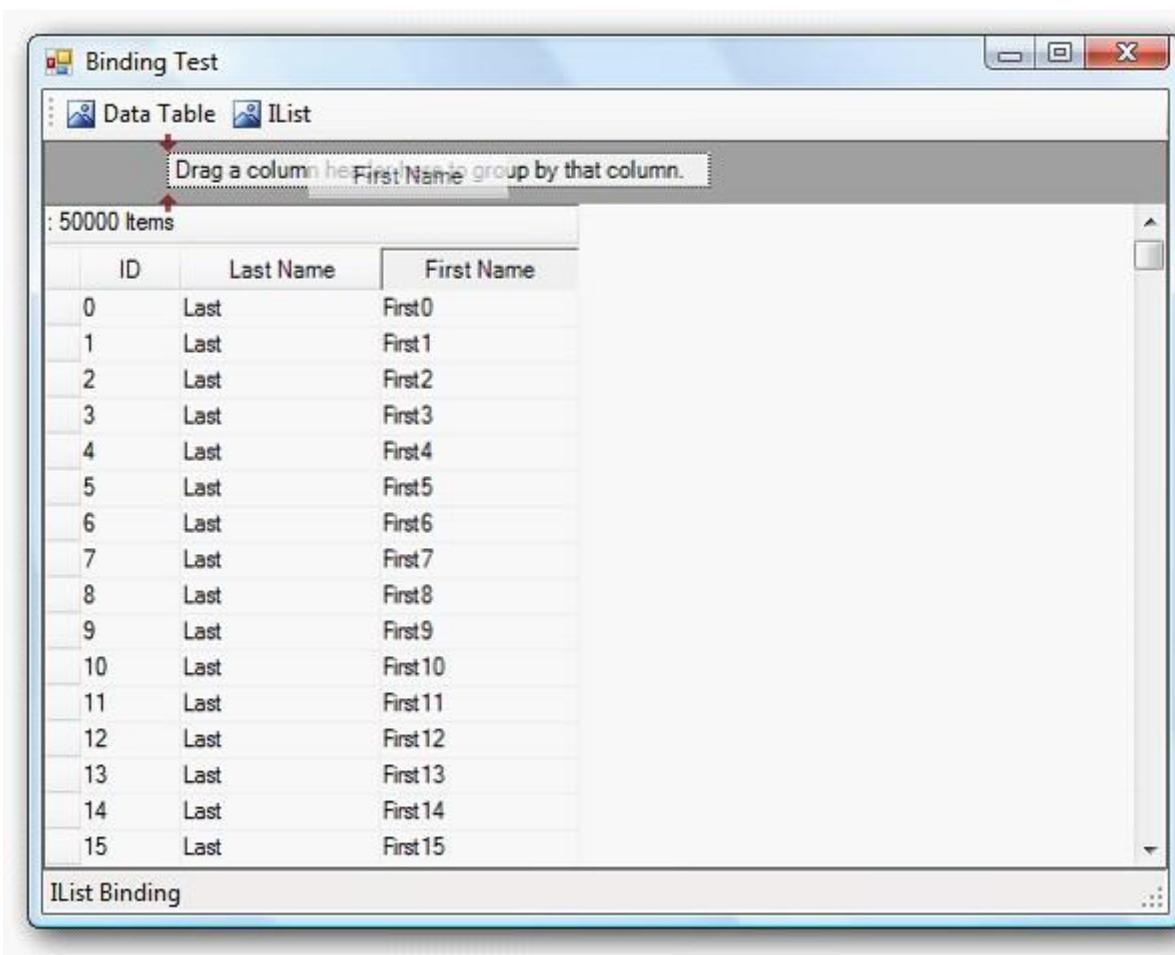
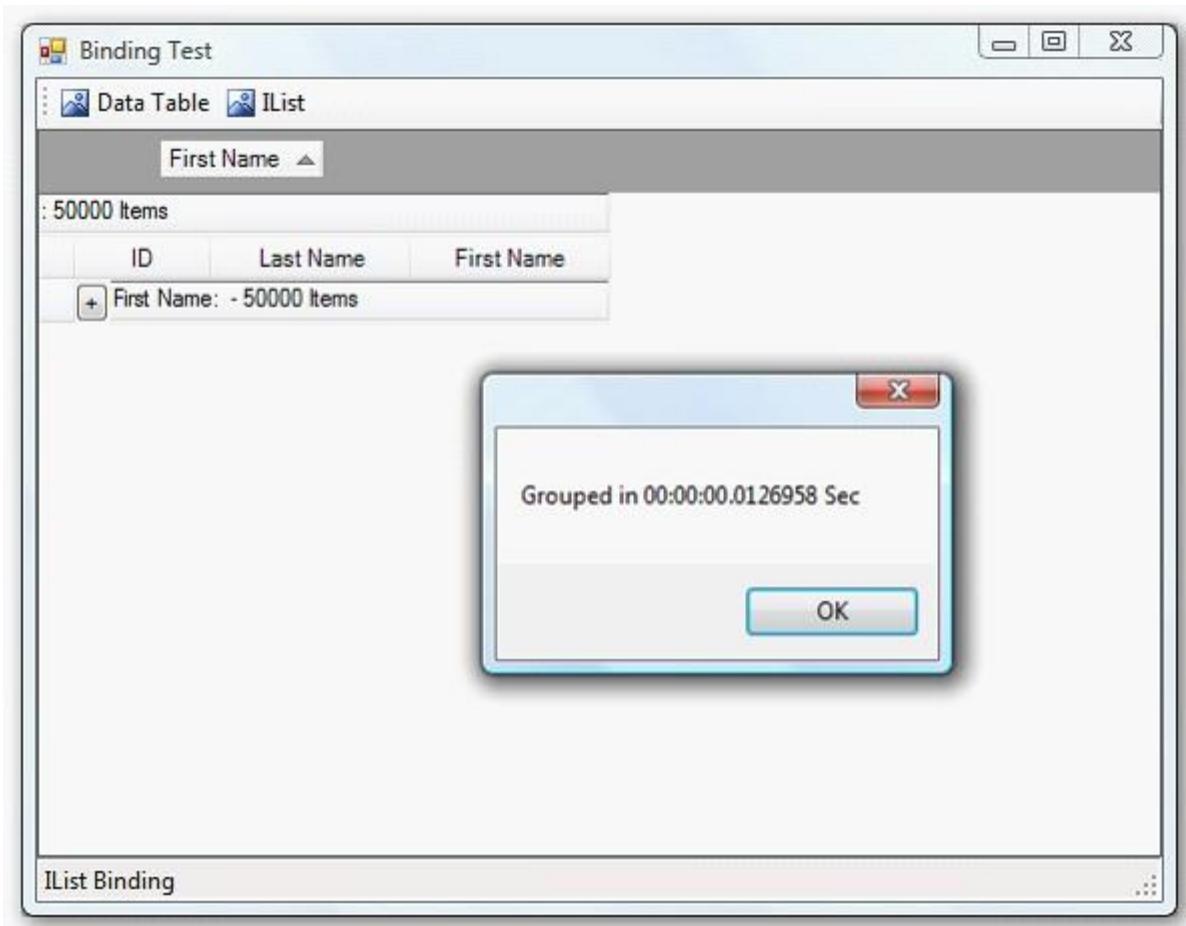


Figure 270: First Name Is Dragged to Add to Group.



*Figure 271: Time Taken to Group the Records*

#### 4.3.4.2 Data Binding

To display data, the grid grouping control must be bound to a data source. The grid grouping control supports variety of data sources such as  **DataTables**,  **DataSets** or any component that implements the interface  **IList**,  **IBindingList**,  **ITypedList** or  **IListSource**. The data source can have multiple nested tables which will be displayed hierarchically by the grid grouping control.

This section deals with the different types of data binding mechanisms, supported by the grid grouping control, which are listed below.

#### 4.3.4.2.1 Data Binding using ADO.NET

**ADO.NET** is an object-oriented set of libraries that allows you to interact with different types of data sources and different types of databases. It comes in different sets of libraries. These libraries are called **DataProviders** and they allow a common way to interact with specific data sources or protocols. The following table lists the data providers that are widely used.

Provider Name	Description
Ole Db Data Provider	Data Sources that expose an OleDb interface, i.e. Access or Excel.
SQL Data Provider	For interacting with Microsoft SQL Server.

#### ADO.NET Objects

The ADO.NET objects are used by the ADO data model to support database interaction. These objects must be created to supply a data-aware control like the grid with database data. The data-aware controls possess the two data binding properties, the **DataSource** and the **DataMember**. Any data source can be bound to the control by assigning it to the **DataSource** and the **DataMember** properties.

##### The Connection Object

It is used for connection to database and managing transactions against the database. The database location and access method will be specified through this connection object. The connection object should be a type of **OleDbConnection** in case of OLE DB data sources or should be a **SqlConnection** object for data sources provided by MS SQL Server.

##### The DataAdapter Object

The data adapter acts like a bridge between the dataset and the data source. It is used to retrieve the data from the database and populate the tables within a dataset. It uses the connection object to connect to the database in order to fill the dataset and update the changes back to the database. There are two adapter components supplied: the **OleDbDataAdapter** and the **SqlDataAdapter**. The former accesses data sources exposed using OLE DB and the later is designed to work with data sources provided by MS SQL Server version 7.0 or later.

##### The DataSet

The dataset acts like a memory resident cache to hold the data. It represents a complete set of data including the tables that organize the data and relationships between the tables. The dataset is designed to help manage data in memory and to support disconnected operations on data. It can be populated by calling the Fill method of the DataAdapter.

### **The Command Object**

Commands contains the information that is submitted to a database, and are represented by provider-specific classes such as SQLCommand. A command can be a stored procedure call, an UPDATE statement, or a statement that returns results.

### **The DataReader Object**

This is a suitable object when you want, to only get the stream of data for reading. The data returned from a data reader is a fast forward-only stream of data. This means that you can only pull the data from the stream in a sequential manner. This is good for speed, but if you need to manipulate data, then a DataSet is a better object to work with.

### **Data Binding Methods**

To bind the grid to a database, you can use any one of the following methods.

#### 1. Binding At Design Time

- [Binding to a database at Design time using VS2003](#)
- [Binding to a database at Design time using VS2005](#)

#### 2. Binding At Run Time

- [Binding to an MDB file at run time](#)
- [Binding to a manually created datasource](#)

#### **4.3.4.2.2 Binding to XML Data**

Grid Grouping Control can be bound to data from XML files. To do this, a **DataSet** object is required which provides the necessary methods that will let you read the Xml data into the dataset. After loading the data, you can bind the grouping grid to this dataset by setting the data binding properties, the **DataSource** and the **DataMember** to the dataset and the table name respectively. It is also possible to save the changes back to the Xml file.

The following table lists some important methods provided by the dataset that allows you to manipulate Xml data. In this, the XmlSchema represents the type of the data stored in the Xml file.

Method Name	Description
ReadXml	Reads the Xml Schema and the data into the dataset using the specified Xml file.
ReadXmlSchema	Reads the Xml Schema from the specified file into the dataset.
WriteXml	Writes the current data, and optionally the schema, for the dataset into the specified Xml file.
WriteXmlSchema	Writes the dataset structure as an Xml schema into the specified file.

The following code example illustrates the binding process.

**[C#]**

```
// Create a Data Set.
DataSet XmlData = new DataSet();

// Populate it with data from an XML file.
XmlData.ReadXml("C:\\Data\\Customers_Orders.xml");

// Bind the grid to the Data Set.
gridGroupingControl1.DataSource = XmlData;
gridGroupingControl1.DataMember = XmlData.Tables[0];
```

**[VB .NET]**

```
' Create a Data Set.
Dim XmlData As New DataSet()

' Populate it with data from an XML file
XmlData.ReadXml("C:\\Data\\Customers_Orders.xml")

' Bind the grid to the Data Set.
gridGroupingControl1.DataSource = XmlData
gridGroupingControl1.DataMember = XmlData.Tables(0)
```

#### **4.3.4.2.3 Binding to Custom Collections**

**Custom Collections** provides a way to store arbitrary objects in a structured fashion that can be bound to the grouping grid. All the data binding is based on a set of interfaces that define different capabilities of objects and collections within the context of accessing and navigating through data. These interfaces set up a two-way communication between the bound grid and the objects collection used by the same grid. Those collections may be custom business objects collection or may be the one provided by .NET Framework itself like DataView.

The data binding interfaces will allow you to create collections of custom objects where you want to present those collections of objects together, through the grid or you can navigate through the objects to view them through the same grid and interact with them. Some of these interfaces are **IList**, **ITypedList**, or **IBindingList** whose definitions are given below.

##### **The IList Interface**

Using this interface, you can create an ordered, indexed set of data items. The **IList** interface is one of the most important interfaces in data binding, because complex data-bound controls can only be bound to collections that implement **IList**. This interface lets you manage the collection's data by adding, removing, inserting and accessing items.

The data source implementing the **IList** interface must have at least one record in order to make the bound controls like grid, to create any rows. It will not be notified of any data changes and thus the changes must be updated manually.

##### **The ITypedList Interface**

**ITypedList** is suitable for complex data binding process where you want to control which columns are visible, with which description and how they should be treated, for example, should they be read only, even there is a set clause in the property definition. Using this interface, you can tell the bound grid exactly how the objects inside a bound collection have to show up in the control and which properties should show up and how they should be treated.

In this case, it is not necessary to have any records for the rows to be created. Like **IList**, the data source will not be notified when items are added or removed from the list.

##### **The IBindingList Interface**

The **IBindingList** interface is the most important data-binding interface that provides rich data binding support. Implementing this interface lets you control over changes to the list, sorting and searching the list. One important benefit is the support for providing change notifications to the collection subscribing to this interface.

The **IBindingList** interface overcomes the shortcomings of the other interfaces by declaring a **ListChanged** event. The data sources that referencing this interface will hook onto this event and so, it will be aware of any items that are added or removed from the list, which makes the bound grid to update itself automatically.

The chapters in this section will demonstrate how to create such collections by implementing the collection interfaces and how to bind the grouping grid to these collections.

#### 4.3.4.2.3.1 *IList*

This section demonstrates the implementation of a collection using **ArrayList** and shows how to bind this collection to a grouping grid. **ArrayList** is an implementation of **IList** that could be best defined as a hybrid of a normal array and a collection. It holds the items in the order they were added. The items can be retrieved in any order via their index. As elements are added, the capacity of the **ArrayList** increases automatically. It allows null references and allows duplicate elements. Objects implementing the **IList** interface should have at least one record for the data rows to be created. The data rows correspond to the data objects in the collection and the data columns represents the properties of the data.

### Implementation

Follow these steps to bind an array of custom objects to a grouping grid.

1. Create a class (**Data**) whose instances represent the records. Its properties represent the record fields.

[C#]

```
public class Data
{
    public Data() : this("", "", "")
    {
    }
    public Data(string cat_Id, string cat_Name, string desc)
    {
```

```
        this.cat_Id = cat_Id;
        this.cat_Name = cat_Name;
        this.desc = desc;
    }
    private string cat_Name;
    public string CategoryName
    {
        get
        {
            return this.cat_Name;
        }
        set
        {
            this.cat_Name = value;
        }
    }
    private string desc;
    public string Description
    {
        get
        {
            return this.desc;
        }
        set
        {
            this.desc = value;
        }
    }
    private string cat_Id;
    public string CategoryID
    {
        get
        {
            return this.cat_Id;
        }
        set
        {
            this.cat_Id = value;
        }
    }
}
```

**[VB.NET]**

Public Class Data

```
Public Sub New()
    Me.New("", "", "")
End Sub

Public Sub New(ByVal cat_Id As String, ByVal cat_Name As String,
ByVal desc As String)
    Me.cat_Id = cat_Id
    Me.cat_Name = cat_Name
    Me.desc = desc
End Sub

Private cat_Name As String
Public Property CategoryName() As String
    Get
        Return Me.cat_Name
    End Get
    Set(ByVal value As String)
        Me.cat_Name = Value
    End Set
End Property

Private desc As String
Public Property Description() As String
    Get
        Return Me.desc
    End Get
    Set(ByVal value As String)
        Me.desc = Value
    End Set
End Property

Private cat_Id As String
Public Property CategoryID() As String
    Get
        Return Me.cat_Id
    End Get
    Set(ByVal value As String)
        Me.cat_Id = Value
    End Set
End Property
End Class
```

2. Create an instance of **ArrayList** and add a list of **Data** type objects into it. This represents the collection.

[C#]

```
using System;
using System.Collections;

ArrayList al = new ArrayList();
al.Add(new Data("Condiments", "Charlotte Cooper", "Bigfoot Breweries"));
al.Add(new Data("Confections", "Regina Murphy", "Grandma Kelly's
Homestead"));
al.Add(new Data("Grains/Cereals", "Jean-Guy Lauzon", "Ma Maison"));
al.Add(new Data("Meat/Poultry", "Shelley Burke", "New Orleans Cajun
Delights"));
al.Add(new Data("Produce", "Mayumi Ohno", "Mayumi's"));
al.Add(new Data("Seafood", "Robb Merchant", "New England Seafood
Cannery"));
```

**[VB.NET]**

```
Imports System
Imports System.Collections

Dim al As ArrayList = New ArrayList()
al.Add(New Data("Condiments", "Charlotte Cooper", "Bigfoot Breweries"))
al.Add(New Data("Confections", "Regina Murphy", "Grandma Kelly's
Homestead"))
al.Add(New Data("Grains/Cereals", "Jean-Guy Lauzon", "Ma Maison"))
al.Add(New Data("Meat/Poultry", "Shelley Burke", "New Orleans Cajun
Delights"))
al.Add(New Data("Produce", "Mayumi Ohno", "Mayumi's"))
al.Add(New Data("Seafood", "Robb Merchant", "New England Seafood
Cannery"))
```

3. Assign this array list to the grouping grid's **DataSource**.

**[C#]**

```
this.gridGroupingControll1.DataSource = al;
```

**[VB.NET]**

```
Me.gridGroupingControll1.DataSource = al
```

4. Run the sample. Your grid will look similar to the one below.

ArrayList: 6 Items			
	CategoryID	CategoryName	Description
Condiments	Charlotte Cooper	Bigfoot Breweries	
Confections	Regina Murphy	Grandma Kelly's Homestead	
Grains/Cereals	Jean-Guy Lauzon	Ma Maison	
Meat/Poultry	Shelley Burke	New Orleans Cajun Delights	
Produce	Mayumi Ohno	Mayumi's	
Seafood	Robb Merchant	New England Seafood Cannery	

Figure 272: Binding an Array of Custom Objects to the Grouping Grid



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Custom Collections\Array List Demo

#### 4.3.4.2.3.2 IBindingList

This section demonstrates the implementation of **IBindingList** and how to bind this list to a Grid Grouping control. As **IBindingList** derives from **IList**, its implementation requires the implementation of all of the members of the **IList**, **ICollection** and **IEnumerable** interfaces.

#### Benefits

The benefits of using **IBindingList** includes the support for change notifications when the list is modified. It does have a **ListChanged** event which will be fired upon any data changes. If the collection supports changes, it should also support firing a **ListChanged** event when the collection changes. To indicate that, the collection should return true from the **SupportsChangeNotification** property. Hence when items are added or removed from the collection, the grouping grid will be notified of these changes and will update itself automatically.

#### Implementation Procedure

Follow these steps to create your own collection that implements **IBindingList** and to bind this collection to a grouping grid.

1. Create a class(Class1) whose instances will represent the records. Its properties represent the record fields.
2. Create another class(Class2) that implements the interfaces **IBindingList** and **INotifiedPropertyChanged**, in order to be notified of any property value change. This class will represent the list of records and will act as the data store for your grouping grid.
3. Implements the necessary events: **ListChanged** and **PropertyChanged**.
4. Raise ListChanged event once the list is modified.
5. Make the **SupportChangeNotifications** property to return true.
6. Create an instance of Class2 and add a list of records into it. Then assign this list to the **DataSource** of the grouping grid.



**Note:** For the complete code refer the sample:

**<Install Location>\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Grouping.Windows\Samples\2.0\Custom Collections\BindingList Demo**

#### 4.3.4.2.3.3 *ITypedList*

This section demonstrates the implementation of **ITypedList** which will be used for complex data binding. It is best to use this interface where you want a type to explicitly specify the properties it exposes instead of letting the TypeDescriptor to find out using reflection. The interface contains two methods: **GetItemProperties** and **GetListName**.

The **GetItemProperties** accepts no parameters and returns a **PropertyDescriptorCollection**. In this method, you can use **TypeDescriptor.GetProperties** to load the properties from your type. The **GetListName** returns the name of the list.

#### Implementation

Follow the steps below to create a collection (**Books Collection**) that implements **ITypedList** and bind this collection to the grouping grid.

1. Create a class **Book**, that represents the structure of the record.

[C#]

```
class Book
{
    public Book(string bname, string author)
    {
        this.bookName = bname;
    }
}
```

```
        this.author = author;
    }
    private string bookName;
    public string BookName
    {
        get
        {
            return this.bookName;
        }
        set
        {
            this.bookName = value;
        }
    }
    private string author;
    public string Author
    {
        get
        {
            return this.author;
        }
        set
        {
            this.author = value;
        }
    }
}
```

**[VB.NET]**

```
Public Class Book
    Public Sub New(ByVal bname As String, ByVal auth As String)
        Me.bName = bname
        Me.authr = auth
    End Sub
    Private bname As String
    Public Property BookName() As String
        Get
            Return Me.bname
        End Get
        Set
            Me.bname = Value
        End Set
    End Property
    Private authr As String
```

```
Public Property Author() As String
    Get
        Return Me.authr
    End Get
    Set
        Me.authr = Value
    End Set
End Property
End Class
```

2. Create another class Books that inherits the ArrayList class and implements ITypedList. A list of its instances will act as the data store for the grouping grid. Implement **GetItemProperties** and **GetListName** methods.

[C#]

```
public class Books : ArrayList, ITypedList
{
    public PropertyDescriptorCollection
GetItemProperties(PropertyDescriptor[] listAccessors)
    {
        return TypeDescriptor.GetProperties(typeof(Book));
    }

    public string GetListName(PropertyDescriptor[] listAccessors)
    {
        return "Book";
    }
}
```

[VB .NET]

```
Public Class Books : Inherits ArrayList : Implements ITypedList
    Public Function GetItemProperties(ByVal listAccessors As
PropertyDescriptor()) As PropertyDescriptorCollection Implements
ITypedList.GetItemProperties
        Return TypeDescriptor.GetProperties(GetType(Book))
    End Function

    Public Function GetListName(ByVal listAccessors As
PropertyDescriptor()) As String Implements ITypedList.GetListName
        Return "Book"
    End Function
End Class
```

3. Create an instance of Books and add a few records into it.

**[C#]**

```
Books MyBooks = new Books();
MyBooks.Add(new Book("Computer Networks", "Tanenbaum"));
MyBooks.Add(new Book("Data Structures", "Tremblay and Sorenson"));
MyBooks.Add(new Book("Database Management", "Alexis Leon"));
```

**[VB .NET]**

```
Dim MyBooks As Books = New Books()
MyBooks.Add(New Book("Computer Networks", "Tanenbaum"))
MyBooks.Add(New Book("Data Structures", "Tremblay and Sorenson"))
MyBooks.Add(New Book("Database Management", "Alexis Leon"))
```

4. Assign this list to the grouping grid's **DataSource**.

**[C#]**

```
this.gridGroupingControl1.DataSource = MyBooks;
```

**[VB .NET]**

```
Me.GridGroupingControl1.DataSource = MyProducts
```

5. Run the sample. Given below is a sample screenshot.

Book: 3 Items		
	Author	BookName
	Tanenbaum	Computer Networks
	Tremblay and Sorenson	Data Structures
	Alexis Leon	Database Management

*Figure 273: Binding a Collection that implements *ITypedList* to the Grouping Grid*

 **Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Custom Collections\Array List Demo

#### 4.3.4.2.3.4 Strongly Typed Collections

The Grid Grouping control can be bound to a Strongly Typed Collection. A **Strongly Typed Collection** is a collection that stores a Known Type. It can be viewed as an array of specific object. For example, suppose your application needs to store the information about products in a factory. You could create a Strongly Typed collection of Product objects.

#### Benefits

- Since the collection knows the type of the object, it does not need to type cast the items between the type you are really storing and a generic object type stored in a collection.
- If you are writing the code to manage the collection items, then you can perform any other operations on the items that are being written into and read from the collection.

In this lesson, you will learn about the following topics.

#### 4.3.4.2.3.4.1 Collection Base

This section demonstrates how to build a **Strongly Typed Collection** using **CollectionBase** class. A Strongly Typed collection can be created by inheriting from the **System.Collections.CollectionBase** class. The CollectionBase class implements **IList**, **IListSource**, and **IEnumerable**. These interfaces enable the users to implement the methods and properties that support binding, enumerating and looping using **ForEach** construct. The result is that your strongly typed collections can be bound directly to our grid grouping control as a datasource.

1. Create a class **Product** whose instances represent the records and properties represent the record fields.

[C#]

```
class Product
{
```

```
string productName, qtyPerUnit;

public Product(string name, string qty)
{
    this.productName = name;
    this.qtyPerUnit = qty;
}

public string ProductName
{
    get
    {
        return productName;
    }
    set
    {
        productName = value;
    }
}
public string QuantityPerUnit
{
    get
    {
        return qtyPerUnit;
    }
    set
    {
        qtyPerUnit = value;
    }
}
```

**[VB .NET]**

```
Class Product
    Private ProdName As String
    Private QtyPerUnit As String

    Public Sub New(ByVal name As String, ByVal qty As String)
        ProdName = name
        QtyPerUnit = qty
    End Sub
    Public Property ProductName() As String
        Get
            Return ProdName
        End Get
    End Property

```

```
End Get
Set(ByVal Value As String)
    ProdName = Value
End Set
End Property
Public Property QuantityPerUnit() As String
    Get
        Return QtyPerUnit
    End Get
    Set(ByVal Value As String)
        QtyPerUnit = Value
    End Set
End Property
End Class
```

2. Create another class **Products** that derives the CollectionBase class. To make it Strongly Typed, add a default property that will return a typed object. Also, implement an **ADD** method.

[C#]

```
using System;
using System.Collections;

class Products : CollectionBase
{
    // Default Property.
    public Product this[int index]
    {
        get { return (Product)List[index]; }
        set { List[index] = (Product)value; }
    }

    public int Add(Product item)
    {
        return List.Add(item);
    }
}
```

[VB .NET]

```
Imports System
Imports System.Collections

Public Class Products
```

```
Inherits CollectionBase

// Using Default Property Indexer.
Default Public Property Item(ByVal Index As Integer) As Product
    Get
        Return CType(List.Item(Index), Product)
    End Get
    Set(ByVal Value As Product)
        List.Item(Index) = Value
    End Set
End Property

Public Function Add(ByVal Item As Product) As Integer
    Return List.Add(Item)
End Function
End Class
```

3. Create a **Products Collection** by instantiating Products class and adding few records into it.

[C#]

```
Products MyProducts = new Products();
MyProducts.Add(new Product("Chai", "10 boxes x 20 bags"));
MyProducts.Add(new Product("Aniseed Syrup", "12 - 550 ml bottles"));
MyProducts.Add(new Product("Sir Rodney's Marmalade", "30 gift boxes"));
```

[VB.NET]

```
Dim MyProducts As Products = New Products()
MyProducts.Add(New Product("Chai", "10 boxes x 20 bags"))
MyProducts.Add(New Product("Aniseed Syrup", "12 - 550 ml bottles"))
MyProducts.Add(New Product("Sir Rodney's Marmalade", "30 gift boxes"))
```

4. Finally assign this collection to the grouping grid's **DataSource**.

[C#]

```
this.gridGroupingControl1.DataSource = MyProducts;
```

[VB.NET]

```
Me.GridGroupingControl1.DataSource = MyProducts
```

5. Run the sample. Here is a sample screen shot.

Products: 3 Items		
	ProductName	QuantityPerUnit
	Chai	10 boxes x 20 bags
	Aniseed Syrup	12 - 550 ml bottles
	Sir Rodney's Marmalade	30 gift boxes

Figure 274: Creating a Strongly Typed Collection by using the CollectionBase Class



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Custom Collections\Collection Base Demo

#### 4.3.4.2.3.4.2 Generic Collection

This section deals with the implementation of **Generic Collection**. Generics refer to those classes, structure, methods and interfaces that have place holders for the types they can contain or use. A generic collection class uses type parameters as place holders for the type of objects it stores, for the type of its fields and the parameter types for its methods. The actual types are assigned to these place holders while creating the instances.

It provides a standard way to create a non type-specific collection. Hence we can get the immediate benefit of type safety without having to derive from a base collection type and implement type-specific members. Through generics, it is possible to have a single array class to store a list of Players or even a list of Products.

The .NET 2.0 framework provides a number of generic collections to work with. The generic collection classes have been defined in the **System.Collections.Generic** namespace. Some of them are listed below.

- **List<T>** is the generic version of ArrayList based on the generic interface **IList**.
- **BindingList<T>** is the generic collection based on **IBindingList**.
- **BindingListView<T>** is a generic collection based on **IBindingListView**.

where T is the place-holder for the type parameter.

Off these classes, the IBindingList implementation has some specialities. The BindingList<T> automatically raises ListChanged event when the INotifyPropertyChanged.PropertyChanged event is raised in a child object. It also raises it when items are added or removed. When you implement custom collection and want the grid to react to changes in datasource you should implement IBindingList and raise ListChanged events.

## Implementation

Follow the steps below to implement a generic collection and bind it to our grouping grid control. This implementation uses BindingList<T> class.

1. Create a class (**CustomClass**) whose objects represent the records and properties represent the record fields. This class implements INotifyPropertyChanged interface in order to trigger the grid to react to changes in the list.

[C#]

```
public class CustomClass : INotifyPropertyChanged
{
    int id;
    string first_name;
    string last_name;
    string address;
    string city;

    public CustomClass(int id, string fname, string lname, string
addr, string city)
    {
        this.id = id;
        first_name = fname;
        last_name = lname;
        address = addr;
        this.city = city;
    }

    public int ID
    {
        get { return id; }
        set
        {
            if (id != value)
            {
                id = value;
                RaisePropertyChanged("ID");
            }
        }
    }
}
```

```
        }

    }

    public string FirstName
    {
        get { return first_name; }
        set
        {
            if (first_name != value)
            {
                first_name = value;
                RaisePropertyChanged("FirstName");
            }
        }
    }

    public string LastName
    {
        get { return last_name; }
        set
        {
            if (last_name != value)
            {
                last_name = value;
                RaisePropertyChanged("LastName");
            }
            last_name = value;
        }
    }

    public string Address
    {
        get { return address; }
        set
        {
            if (address != value)
            {
                address = value;
                RaisePropertyChanged("Address");
            }
        }
    }

    public string City
    {
        get { return city; }
```

```
    set
    {
        if (city != value)
        {
            city = value;
            RaisePropertyChanged("City");
        }
    }

    void RaisePropertyChanged(string name)
    {
        if (PropertyChanged != null)
            PropertyChanged(this, new PropertyChangedEventArgs(name));
    }

    // INotifyPropertyChanged Members.
    public event PropertyChangedEventHandler PropertyChanged;
}
```

**[VB.NET]**

```
Public Class CustomClass : Implements INotifyPropertyChanged
    Dim cusid As Integer
    Dim first_name As String, last_name As String, addr As String,
    cityname As String

    Public Sub New(ByVal id As Integer, ByVal fname As String, ByVal
    lname As String, ByVal addr As String, ByVal city As String)
        Me.cusid = id
        Me.first_name = fname
        Me.last_name = lname
        Me.addr = addr
        Me.cityname = city
    End Sub

    Public Property ID() As Integer
        Get
            Return cusid
        End Get
        Set(ByVal value As Integer)
            If cusid <> value Then
                cusid = value
                RaisePropertyChanged("ID")
            End If
        End Set
    End Property
```

```
End Property

Public Property FirstName() As String
    Get
        Return first_name
    End Get
    Set(ByVal value As String)
        If first_name <> value Then
            first_name = value
            RaisePropertyChanged("FirstName")
        End If
    End Set
End Property

Public Property LastName() As String
    Get
        Return last_name
    End Get
    Set(ByVal value As String)
        If last_name <> value Then
            last_name = value
            RaisePropertyChanged("LastName")
        End If
    End Set
End Property

Public Property Address() As String
    Get
        Return addr
    End Get
    Set(ByVal value As String)
        If addr <> value Then
            addr = value
            RaisePropertyChanged("Address")
        End If
    End Set
End Property

Public Property city() As String
    Get
        Return cityname
    End Get
    Set(ByVal value As String)
        If cityname <> value Then
            cityname = value
            RaisePropertyChanged("City")
        End If
    End Set
End Property
```

```
        End If
    End Set
End Property

Sub RaisePropertyChanged(ByVal name As String)
    RaiseEvent PropertyChanged(Me, New
    PropertyChangedEventArgs(name))
End Sub

Public Event PropertyChanged As PropertyChangedEventHandler
Implements INotifyPropertyChanged.PropertyChanged

End Class
```

2. Instantiate `BindingList <T>` class by specifying the type of the collection as `CustomClass` and add few records into it. This will create a collection of `CustomClass` type objects.

**[C#]**

```
using System.Collections.Generic;

BindingList<CustomClass> bl = new BindingList<CustomClass>();
bl.Add(new CustomClass(0101, "Charlotte", "Cooper", "49 Gilbert St.",
"London"));
bl.Add(new CustomClass(0102, "Shelley", "Burke", "P.O. Box 78934", "New
Orleans"));
bl.Add(new CustomClass(0103, "Regina", "Murphy", "707 Oxford Rd.", "Ann
Arbor"));
bl.Add(new CustomClass(0104, "Yoshi", "Nagase", "9-8 Sekimai Musashino-
shi", "Tokyo"));
bl.Add(new CustomClass(0105, "Mayumi", "Ohno", "Calle del Rosal 4",
"Oviedo"));
```

**[VB.NET]**

```
Imports System.Collections.Generic

Dim bl As BindingList(Of CustomClass) = New BindingList(Of
CustomClass)()
bl.Add(New CustomClass(101, "Charlotte", "Cooper", "49 Gilbert St.",
"London"))
bl.Add(New CustomClass(102, "Shelley", "Burke", "P.O. Box 78934", "New
Orleans"))
bl.Add(New CustomClass(103, "Regina", "Murphy", "707 Oxford Rd.", "Ann
Arbor"))
bl.Add(New CustomClass(104, "Yoshi", "Nagase", "9-8 Sekimai Musashino-
shi", "Tokyo"))
```

```
bl.Add(New CustomClass(105, "Mayumi", "Ohno", "Calle del Rosal 4",
"Oviedo"))
```

3. Assign this list to the grouping grid's **DataSource**.

[C#]

```
this.gridGroupingControl1.DataSource = bl;
```

[VB .NET]

```
Me.GridGroupingControl1.DataSource = bl
```

4. Finally, run the sample. Your grid will look like this.

	LastName	Address	FirstName	ID	City
	Cooper	49 Gilbert St.	Charlotte	101	London
	Burke	P.O. Box 78934	Shelley	102	New Orleans
	Murphy	707 Oxford Rd.	Regina	103	Ann Arbor
	Nagase	9-8 Sekimai Musashino-shi	Yoshi	104	Tokyo
	Ohno	Calle del Rosal 4	Mayumi	105	Oviedo

Figure 275: Implementing and binding a Generic Collection to the Grid Grouping Control



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Custom Collections\Generic Collection Demo

#### 4.3.4.2.4 Unbound Mode

The Grid Grouping control can be operated in **Unbound Mode**. In unbound mode, you can add your own columns to the grouping grid along with the other bound columns.

#### Implementation

This section demonstrates how to add custom columns to a grouping grid. The **TableDescriptor.UnboundFields.Add()** method will allow you to add unbound fields to the grouping grid. The unbound values can be provided in QueryValue event and any changes in the values can be stored back to the data store by handling SaveValue event. Additionally, you can handle QueryCellStyleInfo event to customize the unbound cells individually.

The values must be saved somewhere because the grouping grid does not maintain any data structure to store the cell values. Since the values are unbound, they cannot be stored into the bound data source too. In this example, a HashTable is used to save the values of the unbound column.

The example displays an unbound CheckBox column along with other bound columns using a grouping grid.

1. Create a Grid Grouping control and bind it to a data store.

#### [C#]

```
private Syncfusion.Windows.Forms.Grid.Grouping.GridGroupingControl
gridGroupingControl1;

// Define a Grouping Grid.
this.gridGroupingControl1 = new
Syncfusion.Windows.Forms.Grid.Grouping.GridGroupingControl();
this.gridGroupingControl1.Size = new System.Drawing.Size(160,200 );

// Create a Data Store.
DataTable dt = new DataTable("MyTable");
int nCols = 2;
int nRows = 5;
for(int i = 0; i < nCols; i++)
dt.Columns.Add(new DataColumn(string.Format("Col{0}", i)));
for(int i = 0; i < nRows; ++i)
{
    DataRow dr = dt.NewRow();
    for(int j = 0; j < nCols; j++)
        dr[j] = string.Format("row{0} col{1}", i, j);
    dt.Rows.Add(dr);
}

// Bind the data source to the grouping grid.
this.gridGroupingControl1.DataSource = dt;
```

#### [VB.NET]

```

' Define a Grouping Grid.
Private gridGroupingControl1 As Syncfusion.Windows.Forms.Grid.Grouping.GridGroupingControl

Me.gridGroupingControl1 = New Syncfusion.Windows.Forms.Grid.Grouping.GridGroupingControl()
Me.gridGroupingControl1.Size = New System.Drawing.Size(160,200)

' Create a Data Store.
Dim dt As DataTable = New DataTable("MyTable")

Dim nCols As Integer = 2
Dim nRows As Integer = 5

Dim i As Integer = 0
Do While i < nCols
    dt.Columns.Add(New DataColumn(String.Format("Col{0}", i)))
    i += 1
Loop

i = 0
Do While i < nRows
    Dim dr As DataRow = dt.NewRow()
    Dim j As Integer = 0
    Do While j < nCols
        dr(j) = String.Format("row{0} col{1}", i, j)
        j += 1
    Loop
    dt.Rows.Add(dr)
    i += 1
Loop

' Bind the data source to the grouping grid.
Me.GridGroupingControl1.DataSource = dt

```

2. Create a FieldDescriptor that well describes your custom column and add it to the UnboundFieldDescriptor collection of the grouping grid.

[C#]

```

FieldDescriptor unboundField = new FieldDescriptor("CheckboxCol", "", false, "");
unboundField.ReadOnly = false;
this.gridGroupingControl1.TableDescriptor.UnboundFields.Add(unboundField);

```

**[VB.NET]**

```
Dim unboundField As FieldDescriptor = New  
FieldDescriptor("CheckboxCol", "", False, "")  
unboundField.ReadOnly = False  
Me.gridGroupingControl1.TableDescriptor.UnboundFields.Add(unboundField)
```

3. Setup check boxes in the unbound column. You can also customize the unbound cells through the Appearance property.

**[C#]**

```
gridGroupingControl1.TableDescriptor.Columns["CheckboxCol"].Appearance.  
AnyRecordFieldCell.CellType = "CheckBox";  
gridGroupingControl1.TableDescriptor.Columns["CheckboxCol"].Appearance.  
AnyRecordFieldCell.CheckBoxOptions.CheckedValue = "True";  
gridGroupingControl1.TableDescriptor.Columns["CheckboxCol"].Appearance.  
AnyRecordFieldCell.CheckBoxOptions.UncheckedValue = "False";  
gridGroupingControl1.TableDescriptor.Columns["CheckboxCol"].Appearance.  
AnyRecordFieldCell.HorizontalAlignment =  
GridHorizontalAlignment.Center;  
gridGroupingControl1.TableDescriptor.Columns["CheckboxCol"].Appearance.  
AnyRecordFieldCell.VerticalAlignment = GridVerticalAlignment.Middle;
```

**[VB.NET]**

```
gridGroupingControl1.TableDescriptor.Columns("CheckboxCol").Appearance.  
AnyRecordFieldCell.CellType = "CheckBox"  
gridGroupingControl1.TableDescriptor.Columns("CheckboxCol").Appearance.  
AnyRecordFieldCell.CheckBoxOptions.CheckedValue = "True"  
gridGroupingControl1.TableDescriptor.Columns("CheckboxCol").Appearance.  
AnyRecordFieldCell.CheckBoxOptions.UncheckedValue = "False"  
gridGroupingControl1.TableDescriptor.Columns("CheckboxCol").Appearance.  
AnyRecordFieldCell.HorizontalAlignment = GridHorizontalAlignment.Center  
gridGroupingControl1.TableDescriptor.Columns("CheckboxCol").Appearance.  
AnyRecordFieldCell.VerticalAlignment = GridVerticalAlignment.Middle
```

4. Handle QueryValue and SaveValue events to set and save back the unbound values. Define a HashTable to store the unbound values.

**[C#]**

```
Hashtable unboundValues = new Hashtable();  
  
private void gridGroupingControl1_QueryValue(object sender,  
FieldValueEventArgs e)
```

```
{  
    if (e.Field.Name == "CheckboxCol")  
    {  
        string key = e.Record.GetValue("Col1").ToString();  
        if (key != null)  
        {  
            object val = unboundValues[key];  
            e.Value = val;  
        }  
    }  
  
private void gridGroupingControl1_SaveValue(object sender,  
FieldValueEventArgs e)  
{  
    if (e.Field.Name == "CheckboxCol")  
    {  
        string key = e.Record.GetValue("Col1").ToString();  
        if (key != null)  
        {  
            object val = e.Value;  
            unboundValues[key] = val;  
        }  
    }  
}
```

**[VB.NET]**

```
Hashtable(unboundValues = New Hashtable())  
  
Private Sub gridGroupingControl1_QueryValue(ByVal sender As Object,  
ByVal e As FieldValueEventArgs) Handles gridGroupingControl1.QueryValue  
If e.Field.Name = "CheckboxCol" Then  
    Dim key As String = e.Record.GetValue("Col1").ToString()  
    If Not key Is Nothing Then  
        Dim val As Object = unboundValues(key)  
        e.Value = val  
    End If  
End If  
End Sub  
  
Private Sub gridGroupingControl1_SaveValue(ByVal sender As Object,  
ByVal e As FieldValueEventArgs) Handles gridGroupingControl1.SaveValue  
If e.Field.Name = "CheckboxCol" Then  
    Dim key As String = e.Record.GetValue("Col1").ToString()  
    If Not key Is Nothing Then
```

```
    Dim val As Object = e.Value
    unboundValues(key) = val
End If
End If
End Sub
```

5. Customize the unbound cells by handling the QueryCellStyleInfo event.

[C#]

```
this.gridGroupingControl1.QueryCellStyleInfo += new
Syncfusion.Windows.Forms.Grid.Grouping.GridTableCellStyleInfoEventHandler(this.gridGroupingControl1_QueryCellStyleInfo);

private void gridGroupingControl1_QueryCellStyleInfo(object sender,
GridTableCellStyleInfoEventArgs e)
{
    if (e.TableCellIdentity.ColumnIndex == 3 &&
e.TableCellIdentity.RowIndex > 2)
    {
        if (e.TableCellIdentity.RowIndex % 4 == 0)
            e.StyleCellValue = false;
        else
            e.StyleCellValue = true;
    }
}
```

[VB .NET]

```
Private Sub gridGroupingControl1_QueryCellStyleInfo(ByVal sender As
Object, ByVal e As GridTableCellStyleInfoEventArgs) Handles
gridGroupingControl1.QueryCellStyleInfo
    If e.TableCellIdentity.ColumnIndex = 3 AndAlso
e.TableCellIdentity.RowIndex > 2 Then
        If e.TableCellIdentity.RowIndex Mod 4 = 0 Then
            e.StyleCellValue = False
        Else
            e.StyleCellValue = True
        End If
    End If
End Sub
```

6. Run the sample. Here is a sample screenshot.

	Contact ID	Contact Name	Unbound
1	ALFKI		<input checked="" type="checkbox"/>
2	ARCOS		<input type="checkbox"/>
3	BERGS		<input checked="" type="checkbox"/>
4	CHOPS		<input type="checkbox"/>
5	ERNSH		<input checked="" type="checkbox"/>
6	FRANK		<input type="checkbox"/>
7	KETHS		<input checked="" type="checkbox"/>
8	PALPS		<input type="checkbox"/>
9	RICHARD		<input checked="" type="checkbox"/>
10	SAMS		<input type="checkbox"/>

Figure 276: Adding Unbound Columns to the Grid Grouping Control

#### 4.3.4.3 Data Representation

This section deals with the different layouts supported by the grid grouping control. These layouts help the grouping grid to organize the data display. For example, if you want to view the data arranged based on the values of a specific field, then you can group the data columns by the required field.

In this lesson, you will learn about the following topics.

##### 4.3.4.3.1 Grouping

A Group represents a collection of records that belong to a category. The Grid Grouping control allows the user to group the data by one or more columns. When grouping is applied, the data will be organized into a hierarchical structure based on the matching field values. The records having identical values in the grouped column will be combined to form a group. Each group is identified by its GroupCaptionSection that can be expanded to bring the underlying records into view. The GroupCaptionSection carries the information about a particular group like the group name, number of items(records) in the group, etc. It also contains plus / minus buttons that allows the user to expand / collapse the groups individually. By default, a grid table has one group.

##### GroupedColumns Collection

The **GroupedColumns** collection defines the fields to group by and the sort order. The collection can have multiple entries resulting in nested groups. The GroupedColumns collection of grouping grid can be accessed via its TableDescriptor. The collection consists of various properties, methods and events that allows the user to manage the elements in it.

## Adding Data Groups

### Simple Grouping

The data can be grouped by adding the column name to the **TableDescriptor.GroupedColumns** property.

[C#]

```
this.gridGroupingControl1.TableDescriptor.GroupedColumns.Add("Title");
```

[VB .NET]

```
Me.gridGroupingControl1.TableDescriptor.GroupedColumns.Add("Title")
```

The below grid displays the data columns from the **Employees** Table grouped by the values of **Title** field.

The screenshot shows a Windows Form with a grid control. The title bar says "Employees". The grid has four columns: EmployeeID, LastName, FirstName, and Title. The "Title" column is currently sorted in ascending order, as indicated by the blue header bar with an upward arrow icon. The data is grouped by the "Title" field. There are three groups visible: "Title: Inside Sales Coordinator - 1 Items", "Title: Sales Manager - 1 Items", and "Title: Sales Representative - 6 Items". The first group contains one item: EmployeeID 8, LastName Callahan, FirstName Laura, Title Inside Sales Coordinator. The second group contains one item: EmployeeID 5, LastName Buchanan, FirstName Steven, Title Sales Manager. The third group contains six items: EmployeeID 1, LastName Davolio, FirstName Nancy, Title Sales Representative; EmployeeID 3, LastName Leverling, FirstName Janet, Title Sales Representative; EmployeeID 4, LastName Peacock, FirstName Margaret, Title Sales Representative; and EmployeeID 6, LastName Suyama, FirstName Michael, Title Sales Representative.

EmployeeID	Lastname	Firstname	Title
<b>Title: Inside Sales Coordinator - 1 Items</b>			
8	Callahan	Laura	Inside Sales Coordinator
<b>Title: Sales Manager - 1 Items</b>			
5	Buchanan	Steven	Sales Manager
<b>Title: Sales Representative - 6 Items</b>			
1	Davolio	Nancy	Sales Representative
3	Leverling	Janet	Sales Representative
4	Peacock	Margaret	Sales Representative
6	Suyama	Michael	Sales Representative

Figure 277: ListSortDirection

By default, the grouping of a column sorts the records in the ascending order of their GroupedColumn values. It is possible to specify the sort order while grouping. The code below arranges the data in the descending order of their Title field values.

[C#]

```
this.gridGroupingControl1.TableDescriptor.GroupedColumns.Add("Title",
```

```
ListSortDirection.Descending);
```

**[VB .NET]**

```
Me.gridGroupingControl1.TableDescriptor.GroupedColumns.Add("Title",  
ListSortDirection.Descending)
```

The below screenshot reflects this process.

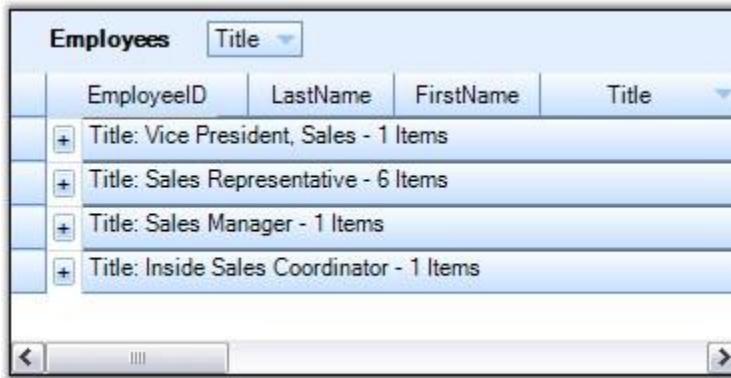


Figure 278: Nested Tables Grouping

When multiple tables are used in nested manner, a child table can also be grouped by getting access to the GroupedColumns property of the desired ChildTableDescriptor. The code below shows this process.

**[C#]**

```
this.gridGroupingControl1.TableDescriptor.Relations[0].ChildTableDescriptor.GroupedColumns.Add("CategoryName", ListSortDirection.Descending);
```

**[VB .NET]**

```
Me.gridGroupingControl1.TableDescriptor.Relations(0).ChildTableDescriptor.GroupedColumns.Add("CategoryName", ListSortDirection.Descending)
```



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\Version  
Number\Windows\Grid.Grouping.Windows\Samples\2.0\Grouping\Grouping Demo

## Multi Column Grouping

Grid Grouping control provides inbuilt support to group the data by more than one column. It is as simple as adding the column names to the GroupedColumns collection. With multicolumn grouping, the grouping grid organizes the data in a hierarchical structure showing groups in different levels. In the below image, you see the Employees data grouped by Title and Country columns.



Figure 279: Multi Column Grouping



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Grouping\Multi Column Grouping Demo

### Grouping Through Designer

The Grouping can also be done at design time. After binding the dataset to the grouping grid, open the TableDescriptor node in the property grid of the Grid Grouping control. In that, accessing the GroupedColumns property will open the SortColumnDescriptorCollection Editor. Clicking the Add button will add an existing column from the dataset. By using the drop down Name, you can change the column by which you want to group the table data. You can also specify the sort order for that column using the SortDirection property.

The below image depicts this process.

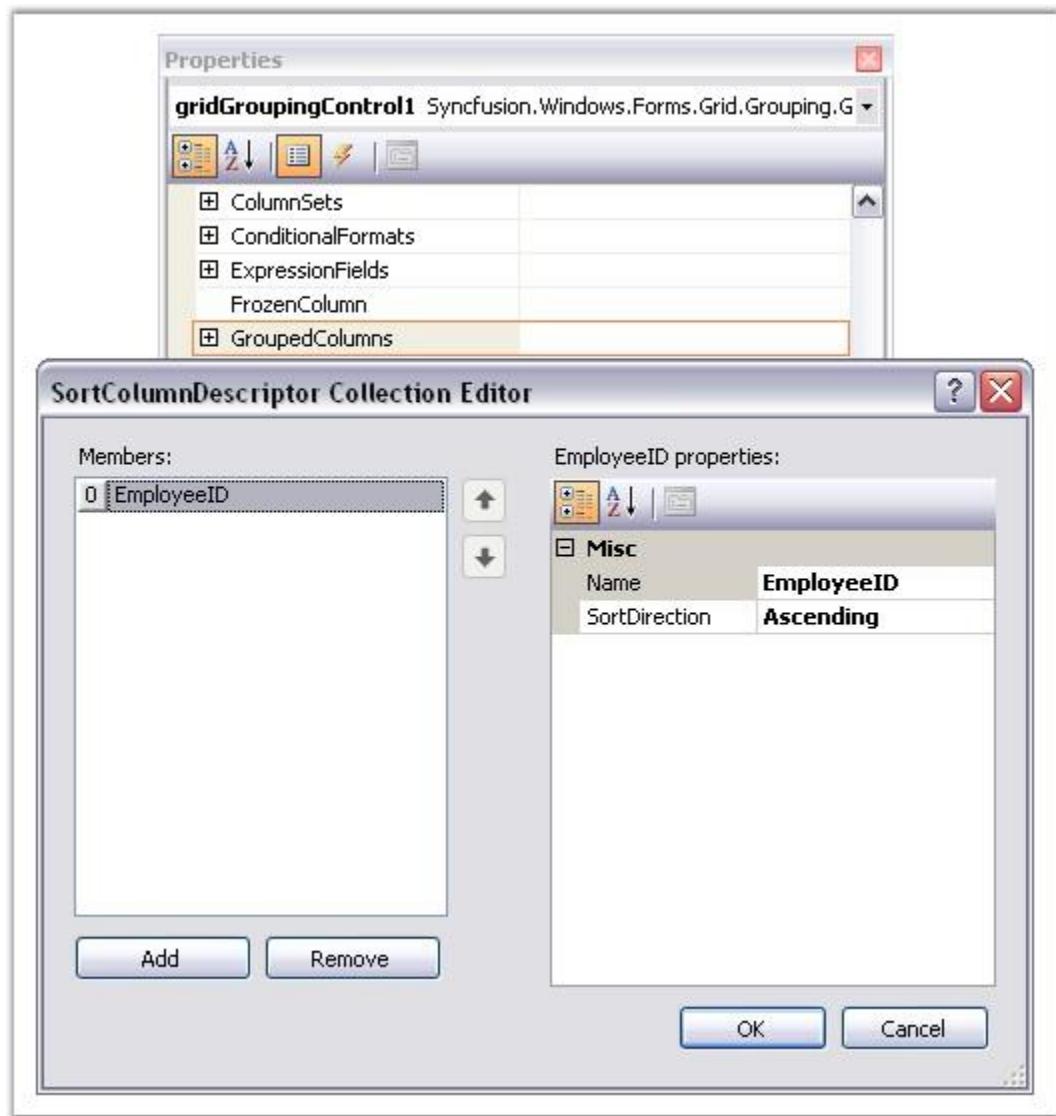


Figure 280: Design Time Grouping

### Using GroupDropArea

The table data can also be grouped simply by dragging the desired column header and dropping it into the GroupDropArea. Removing the column header from the drop area will ungroup the data. You can also be able to change the grouping order by a simple drag and drop action on the column headers.

For more information on GroupDropArea, refer [GroupDropArea](#).

### Preventing a Column from Grouping

To disallow a column being grouped by, the **AllowGroupByColumn** property should be set to False for that column. This property determines whether the grid can be grouped by a column when the user drags the column to the GroupDropArea.

**[C#]**

```
this.gridGroupingControl1.TableDescriptor.Columns[0].AllowGroupByColumn  
= false;
```

**[VB .NET]**

```
Me.gridGroupingControl1.TableDescriptor.Columns(0).AllowGroupByColumn =  
False
```

### **Clearing Groups**

The **GroupedColumns.Clear()** method will remove all the elements from the **GroupedColumns** Collection and hence the data will get ungrouped.

**[C#]**

```
this.gridGroupingControl1.TableDescriptor.GroupedColumns.Clear();
```

**[VB .NET]**

```
Me.gridGroupingControl1.TableDescriptor.GroupedColumns.Clear()
```

### **Removing a given Group**

The **GroupedColumns** property provide two methods to remove a specific group from the collection. **Remove()** method deletes the column with a given name from the **GroupedColumns** collection. As a result, the table data is ungrouped by that column. The **RemoveAt()** method deletes the element at the specified index from the collection.

**[C#]**

```
// Removes the first element.  
this.gridGroupingControl1.TableDescriptor.GroupedColumns.RemoveAt(0);  
  
// Removes the Title element from the columns collection.  
this.gridGroupingControl1.TableDescriptor.GroupedColumns.Remove("Title");
```

**[VB .NET]**

```
' Removes the first element.
Me.gridGroupingControll1.TableDescriptor.GroupedColumns.RemoveAt(0)

' Removes the Title element from the columns collection.
Me.gridGroupingControll1.TableDescriptor.GroupedColumns.Remove("Title")
```

#### 4.3.4.3.1.1 GroupDropArea

**GroupDropArea** provides a drop panel onto which the user can drag and drop the column headers to group the table data by those columns. Its visibility can be controlled by the **ShowGroupDropArea** property. Once it is set to true, a Drop Panel will be added at the top of the grouping grid.

Following code example illustrates how to enable the Group Drop Area.

##### [C#]

```
this.gridGroupingControll1.ShowGroupDropArea = true;
```

##### [VB .NET]

```
Me.gridGroupingControll1.ShowGroupDropArea = True
```

Here are the runtime screens showing the effect of setting the **ShowGroupDropArea** property.

EmployeeID	LastName	FirstName	Title
1	Davolio	Nancy	Sales Representative
2	Fuller	Andrew	Vice President, Sales
3	Leverling	Janet	Sales Representative
4	Peacock	Margaret	Sales Representative
5	Buchanan	Steven	Sales Manager
6	Suyama	Michael	Sales Representative

Figure 281: Startup Grid without GroupDropArea

Employees    Drag a column header here to group by that column			
EmployeeID	LastName	FirstName	Title
1	Davolio	Nancy	Sales Representative
2	Fuller	Andrew	Vice President, Sales
3	Leverling	Janet	Sales Representative
4	Peacock	Margaret	Sales Representative

Figure 282: Grid with GroupDropArea

Employees    Drag a column header here to group by that column			
EmployeeID	LastName	FirstName	Title
1	Davolio	Nancy	Sales Representative
2	Fuller	Andrew	Vice President, Sales
3	Leverling	Janet	Sales Representative
4	Peacock	Margaret	Sales Representative
5	Buchanan	Steven	Sales Manager

Figure 283: Drag & Drop Title column header to the Drop Area

Employees    Title ▲			
EmployeeID	LastName	FirstName	Title
[+]	Title: Inside Sales Coordinator - 1 Items		
[+]	Title: Sales Manager - 1 Items		
[+]	Title: Sales Representative - 6 Items		
[+]	Title: Vice President, Sales - 1 Items		

Figure 284: Grouped Grid by Title Column



Figure 285: GroupDropArea with Multiple Column Headers

**See Also****4.3.4.3.1.1.1 Adding GroupDropArea**

ShowGroupDropArea property will enable the GroupDropArea only for the table at the top level. When nested tables are used, the drop areas for the child tables need to be added at run time. It is achieved by calling the **AddGroupDropArea** method, by specifying the respective child table name in its parameter.

In this example, the grid is bound to a hierarchical dataset containing three tables Categories, Products and OrderDetails. The following code examples illustrate how to add the group drop area for the child tables Products and OrderDetails.

**[C#]**

```
// ShowGroupDropArea adds Group Drop Area for the parent Categories
// table.
this.gridGroupingControl1.ShowGroupDropArea = true;

// Adding Group Drop Areas for nested tables.
this.groupingGrid1.AddGroupDropArea("Products");
this.groupingGrid1.AddGroupDropArea("OrderDetails");
```

**[VB .NET]**

```
' ShowGroupDropArea adds Group Drop Area for the parent Categories
'table.
Me.gridGroupingControl1.ShowGroupDropArea = True

' Adding Group Drop Areas for nested tables.
Me.gridGroupingControl1.AddGroupDropArea("Products")
```

```
Me.gridGroupingControll1.AddGroupDropArea("OrderDetails")
```

Given below is a sample screenshot.

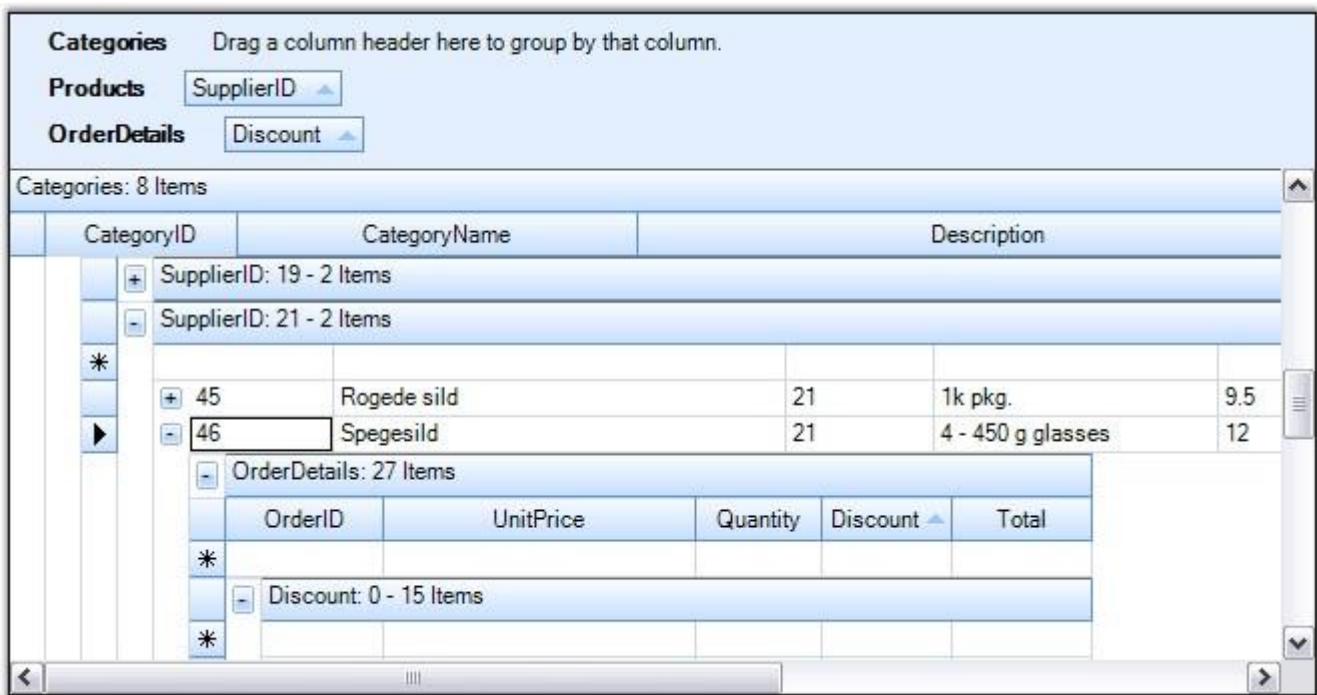


Figure 286: Adding Group Drop Areas to the Child Tables (Product and OrderDetails)

#### 4.3.4.3.1.1.2 Customizing GroupDropArea

Grid Group Drop Area is made up of a collection of Grid controls packed in a panel named GroupDropPanel. A Splitter provides the insulation between the Group Drop Panel and the Grid Table Panel by using which the size of the Drop Panel can be adjusted at run time. While formatting the Group Drop Area, the user should take care of these controls too.

#### Properties affecting GroupDropArea

The table given below lists the properties that allow you to customize the look and feel of the Grid Group Drop Area.

Property	Description
gridGroupingControl1.GridGroupDropArea	Lists the properties & events to customize the drop area.

gridGroupingControl1.GroupDropPanel	Lets the user to control the drop panel behavior.
gridGroupingControl1.Splitter	Provides the splitter related properties.

## Example

In this example, the grouping grid is built with hierarchical dataset created at runtime. The formatting of the Group Drop Area can be controlled by handling the `PrepareViewStyleInfo` event for each of the grids in the Group Drop Panel.

1. Formatting of the Splitter and GroupDropPanel.

**[C#]**

```
// Splitter Color.  
this.gridGroupingControl1.Splitter.BackColor = Color.Red;  
  
// Panel Color.  
this.gridGroupingControl1.GroupDropPanel.BackColor = Color.YellowGreen;
```

**[VB .NET]**

```
' Splitter Color.  
Private Me.gridGroupingControl1.Splitter.BackColor = Color.Red  
  
' Panel Color.  
Private Me.gridGroupingControl1.GroupDropPanel.BackColor =  
Color.YellowGreen
```

2. The `PrepareViewStyleInfo` event for each of the grids can be hooked by looping through the controls in the panel.

**[C#]**

```
foreach (Control ctl in  
this.gridGroupingControl1.GroupDropPanel.Controls)  
{  
    GridGroupDropArea groupDropArea = ctl as GridGroupDropArea;  
  
    switch (groupDropArea.Model.Table.TableDescriptor.Name)  
    {  
        case "ParentTable":  
            groupDropArea.Model.ColCount = 80;
```

```
        groupDropArea.PrepareViewStyleInfo += new  
GridPrepareViewStyleInfoEventHandler(ParentTable_PrepareViewStyleInfo);  
        break;  
  
    case "ChildTable":  
        groupDropArea.Model.ColCount = 80;  
        groupDropArea.PrepareViewStyleInfo += new  
GridPrepareViewStyleInfoEventHandler(ChildTable_PrepareViewStyleInfo);  
        break;  
    }  
}
```

[VB .NET]

```
Dim ctl As Control
For Each ctl In Me.gridGroupingControl1.GroupDropPanel.Controls
Dim groupDropArea As GridGroupDropArea = ctl
Select Case groupDropArea.Model.Table.TableDescriptor.Name
    Case "ParentTable"
        groupDropArea.Model.ColCount = 80
        AddHandler groupDropArea.PrepareViewStyleInfo, AddressOf
        ParentTable_PrepareViewStyleInfo

    Case "ChildTable"
        groupDropArea.Model.ColCount = 80
        AddHandler groupDropArea.PrepareViewStyleInfo, AddressOf
        ChildTable_PrepareViewStyleInfo
End Select
Next ctl
```

- ### 3. Setting the style properties in the PrepareViewStyleInfo event.

[C#]

```
private void ParentTable_PrepareViewStyleInfo(object sender,  
GridPrepareViewStyleInfoEventArgs e)  
{  
    // Setting color to the text displaying the table name.  
    if (e.ColumnIndex == 2 && e.RowIndex == 2)  
    {  
        e.Style.Text = "ParentTable";  
        e.Style.Font.Bold = true;  
        e.Style.BackColor = Color.YellowGreen;  
        e.Style.TextColor = Color.Blue;  
        e.Style.CellType = "Static";  
        e.Style.HorizontalAlignment = GridHorizontalAlignment.Left;
```

```
        e.Style.Enabled = false;
    }
    // Setting color to the drop area.
    else if (e.Style.Text.StartsWith("Drag a"))
    {
        e.Style.Text = "Drag and Drop Parent Table Column headers";
        e.Style.BackColor = Color.White;
    }
    // Setting color to the dropped columns.
    else if (e.Style.Text.StartsWith("Par"))
    {
        e.Style.BackColor = Color.Tomato;
        e.Style.Themed = false;
    }
    // Setting color to the remaining part.
    else
        e.Style.BackColor = Color.YellowGreen;
}

private void ChildTable_PreparesViewStyleInfo(object sender,
GridPrepareViewStyleInfoEventArgs e)
{
    // Setting color to the text displaying the table name.
    if (e.ColumnIndex == 2 && e.RowIndex == 2)
    {
        e.Style.Text = "ChildTable ";
        e.Style.Font.Bold = true;
        e.Style.BackColor = Color.YellowGreen;
        e.Style.TextColor = Color.Yellow;
        e.Style.CellType = "Static";
        e.Style.HorizontalTextAlignment = GridHorizontalTextAlignment.Left;

        e.Style.Enabled = false;
    }
    // Setting color to the drop area.
    else if (e.Style.Text.StartsWith("Drag a"))
    {
        e.Style.Text = "Drag and Drop Parent Table Column headers";
        e.Style.BackColor = Color.Orange;
        e.Style.TextColor = Color.White;
    }
    // Setting color to the dropped columns.
    else if (e.Style.Text.StartsWith("Child"))
    {
        e.Style.BackColor = Color.Orange;
    }
}
```

```
        e.Style.TextColor = Color.White;
        e.Style.Themed = false;
    }
    // Setting color to the remaining part.
    else
        e.Style.BackColor = Color.YellowGreen;
}
```

**[VB.NET]**

```
Private Sub ParentTable_PreparesViewStyleInfo(ByVal sender As Object,
ByVal e As GridPrepareViewStyleEventArgs)

    ' Setting color to the text displaying the table name.
    If e.ColumnIndex = 2 AndAlso e.RowIndex = 2 Then
        e.Style.Text = "ParentTable"
        e.Style.Font.Bold = True
        e.Style.BackColor = Color.YellowGreen
        e.Style.TextColor = Color.Blue
        e.Style.CellType = "Static"
        e.Style.HorizontalContentAlignment = GridHorizontalContentAlignment.Left
        e.Style.Enabled = False

    ' Setting color to the drop area.
    ElseIf e.Style.Text.StartsWith("Drag a") Then
        e.Style.Text = "Drag and Drop Parent Table Column headers"
        e.Style.BackColor = Color.White

    ' Setting color to the dropped columns.
    ElseIf e.Style.Text.StartsWith("Par") Then
        e.Style.BackColor = Color.Tomato
        e.Style.Themed = False

    ' Setting color to the remaining part.
    Else
        e.Style.BackColor = Color.YellowGreen
    End If
End Sub

Private Sub ChildTable_PreparesViewStyleInfo(ByVal sender As Object,
ByVal e As GridPrepareViewStyleEventArgs)

    ' Setting color to the text displaying the table name.
    If e.ColumnIndex = 2 AndAlso e.RowIndex = 2 Then
        e.Style.Text = "ChildTable "
        e.Style.Font.Bold = True
```

```
e.Style.BackColor = Color.YellowGreen
e.Style.TextColor = Color.Yellow
e.Style.CellType = "Static"
e.Style.HorizontalContentAlignment = GridHorizontalAlignment.Left
e.Style.Enabled = False

' Setting color to the drop area.
ElseIf e.Style.Text.StartsWith("Drag a") Then
    e.Style.Text = "Drag and Drop Parent Table Column headers"
    e.Style.BackColor = Color.Orange
    e.Style.TextColor = Color.White

' Setting color to the dropped columns.
ElseIf e.Style.Text.StartsWith("Child") Then
    e.Style.BackColor = Color.Orange
    e.Style.TextColor = Color.White
    e.Style.Themed = False

' Setting color to the remaining part.
Else
    e.Style.BackColor = Color.YellowGreen
End If
End Sub
```

4. Here is a sample output.

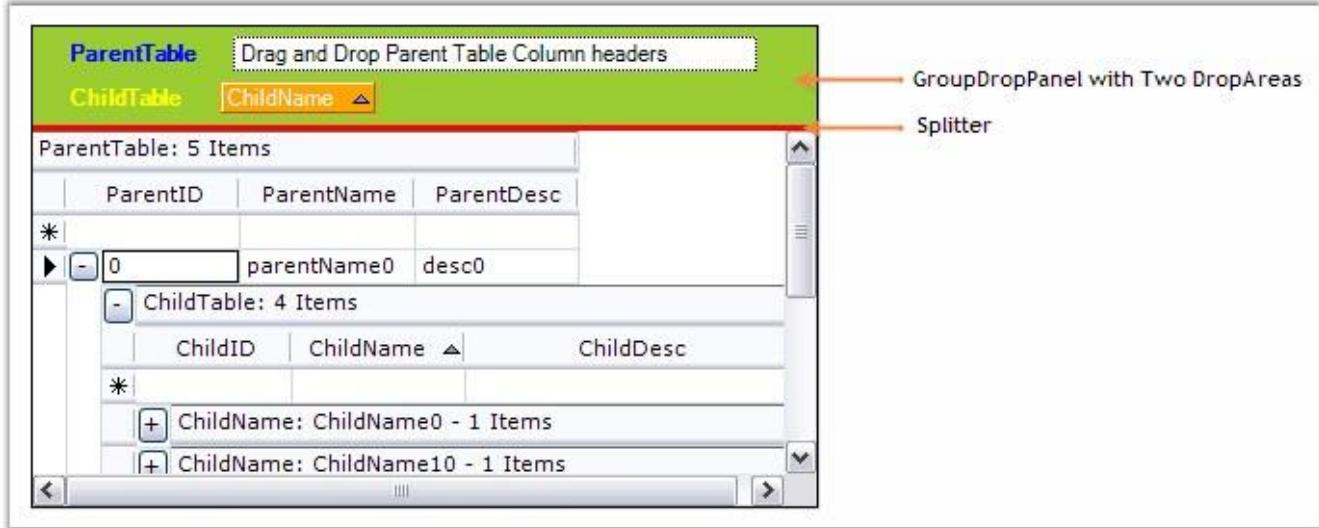


Figure 287: Formatting the Group Drop Area by using the PreviewViewStyleInfo Event

#### 4.3.4.3.1.1.3 Display GroupDropArea in Hierarchy

This feature allows users to display items in the GroupDropArea in a hierarchical order. The items will follow a stacked order.

Hierarchical grouping enables the following interactive features in the GridGrouping control:

- Dynamically remove columns from the grouping area.
- Switch the tree line placement between the top and bottom of hierarchy levels.
- Resize the group drop area dynamically up to the last level of the hierarchy.
- Set the tree lines with desired color.

#### Properties

Property	Description	Type	Data Type
HierarchicalGroupDropArea	Gets or sets a value to enable the GroupDropArea hierarchy in the grid.	Boolean	Boolean, true/false
GridGroupDropArea.AllowRemove	Gets or sets whether the GroupDropArea should support removal of groups dynamically.	Boolean	Boolean, true/false
GridGroupDropArea.TreeLinePlacement	Gets or sets the location for the tree line which will be drawn between the hierarchy items.	Enum	enumeration
GridGroupDropArea.DynamicResizing	Gets or sets the value to resize the GroupDropArea dynamically.	Boolean	Boolean, true/false
GridGroupDropArea.TreeLineColor	Gets or sets the color of the tree lines.	Color	Color

#### 4.3.4.3.1.1.3.1 Sample Link

{installed  
drive}\AppData\Local\Syncfusion\EssentialStudio\{version}\Windows\Grid.Grouping.Windows\Samples\2.0\Grouping\GroupingDemo

#### 4.3.4.3.1.1.3.2 Adding Hierarchical GroupDropArea to an Application

To enable this feature, the **HierarchicalGroupDropArea** property must be set to true.

**[C#]**

```
this.gridGroupingControl1.HierarchicalGroupDropArea = true;
```

**[VB]**

```
Me.gridGroupingControl1.HierarchicalGroupDropArea = True
```

To enable other features supported within the hierarchical GroupDropArea, the following properties can be used:

**[C#]**

```
// Support to dynamically remove the column from being grouped (adds support in default GroupDropArea too).
this.gridGroupingControl1.GridGroupDropArea.AllowRemove = true;

// Support to switch the tree line placement to the top and bottom between hierarchy levels.
this.gridGroupingControl1.GridGroupDropArea.TreeLinePlacement =
TreeLinePlacement.Bottom;

// Support to resize the GroupDropArea dynamically up to the last level of the hierarchy.
this.gridGroupingControl1.GridGroupDropArea.DynamicResizing = true;

// Support to set the tree lines to a desired color.
this.gridGroupingControl1.GridGroupDropArea.TreeLineColor = Color.Red;
```

**[VB]**

```
'Support to dynamically remove the column from being grouped (adds support in default GroupDropArea too).
Me.gridGroupingControl1.GridGroupDropArea.AllowRemove = True
```

```
'Support to switch the tree line placement to the top and bottom
between hierarchy levels.
```

```
Me.gridGroupingControl1.GridGroupDropArea.TreeLinePlacement =
TreeLinePlacement.Bottom
```

```
'Support to resize the GroupDropArea dynamically up to the last level
of the hierarchy.
```

```
Me.gridGroupingControl1.GridGroupDropArea.DynamicResizing = True
```

```
'Support to set the tree lines to a desired color.
```

```
Me.gridGroupingControl1.GridGroupDropArea.TreeLineColor = Color.Red
```

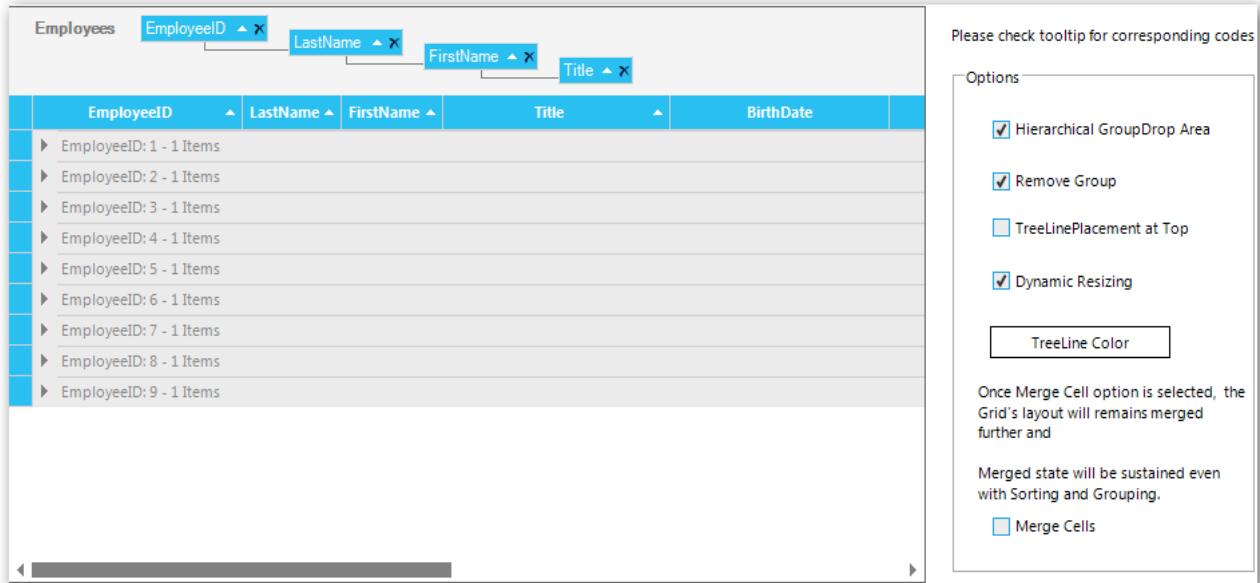


Figure 288: Hierarchical GroupDropArea

#### 4.3.4.3.1.2 GroupByOptions

Grid Grouping control provides a number of options that allows you to control the look and behavior of the groups. You can control the caption text, where and if AddNew row will be displayed and whether captions, headers, footers, preview rows and summaries will be displayed.

##### The GridGroupOptionsStyleInfo class

The **GridGroupOptionsStyleInfo** class, derives **StyleInfoBase**, defines the properties to control the look and feel of the groups. A grouping grid distinguishes between three different kinds of group options which are listed below.

Group Option	Description
TopLevelGroupOptions	Lets you control the look & behavior of top level group.
ChildGroupOptions	Lets you control the look & behavior of child groups.
NestedTableGroupOptions	Lets you control the look & behavior of groups in nested child relations.

### Properties for the Group Options

The table below describes the properties defined in the **GridGroupOptionsStyleInfo** class. These properties are available for all kinds of Group Options.

Group Option	Description
CaptionText	Lets you control the caption text that is displayed.
ShowCaption	Indicates whether a caption row is visible.
ShowCaptionPlusMinus	Indicates whether a plus/minus cell is to be displayed next to the caption.
ShowAddNewBeforeDetails	When true, AddNew record is shown at the top of a group.
ShowAddNewAfterDetails	When true, AddNew record is shown at the bottom of a group.
ShowColumnHeaders	Indicates whether the column headers are visible.
ShowEmptyGroups	Indicates whether a preview is visible when the group is collapsed.
ShowGroupHeader	Indicates whether a header is visible.
ShowGroupFooter	Indicates whether a footer is visible.

ShowGroupPreview	Indicates whether a preview is visible when the group is collapsed.
ShowSummaries	Indicates whether summaries are visible.
ShowGroupSummaryWhenCollapsed	Indicates whether summary items are visible when the group is collapsed.
ShowFilterBar	Indicates whether a filter bar is visible.
ShowStackedHeaders	Indicates whether the stacked headers are visible.
ShowGroupIndentAsCoveredRange	Indicates whether to treat all the indent cells for a group as a single covered cell.
ShowCaptionSummaryCells	Indicates whether a group caption should display summaries in columns instead of only one large caption bar.

[Next chapter](#) in this section discuss these properties in detail with a suitable example.

#### 4.3.4.3.1.2.1 Working with Group Elements

This section explores the various properties that can be used to manipulate the different group elements.

#### Group Headers and Footers

The headers and footers of a group can be used to display any information that is common to all the elements of that group. You can toggle the display of these headers and footers, by using the below given boolean properties.

- <GroupOptions>.ShowGroupHeader
- <GroupOptions>.ShowGroupFooter

where the **<GroupOptions>** can be any one of the following: **TopLevelGroupOptions** to affect only the top most group, **ChildGroupOptions** to affect the child groups or **NestedTableGroupOptions** to affect the groups in nested tables.

You can also able to set the header / footer attributes such as HeaderSectionHeight and FooterSectionHeight, by using the below given properties.

- `TableOptions.GroupHeaderSectionHeight`
- `TableOptions.GroupFooterSectionHeight`

Group headers and footers can be populated by handling the `QueryCellStyleInfo` event wherein you can check for the Header / Footer cell type and provide the data.

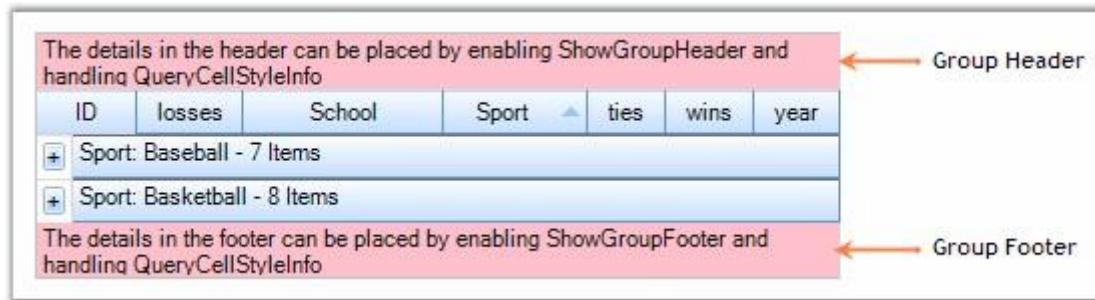


Figure 289: Group Header and Footer

#### GroupPreviewRows

`GroupPreviewSection` is the suitable place when you want to display memo fields or add custom notes for a given group. It can be enabled by setting the `<GroupOptions>.ShowGroupPreview` property to True. You can adjust the size of the preview row through the `TableOptions.GroupPreviewSectionHeight` property. The `QueryCellStyleInfo` event can be used to populate the preview rows.

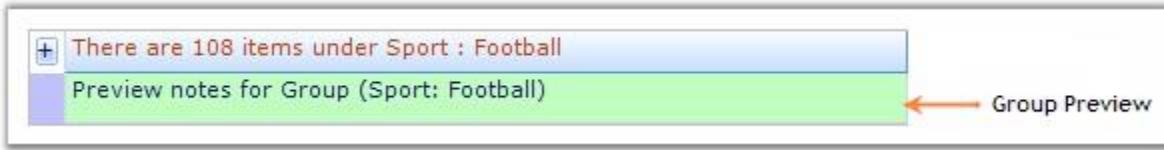


Figure 290: Group Preview Section

#### AddNew Records

Each group can optionally have an `AddNew` row where you can provide the values for a new record. Once a new record is entered, the record will be sorted into the existing record set and will be assigned a group's category automatically. The visibility of the `AddNewRecord` can be controlled through the following two boolean properties.

- `<GroupOptions>.ShowAddNewRecordBeforeDetails` - adds the `AddNew` row at the top of a group.
- `<GroupOptions>.ShowAddNewRecordAfterDetails` - adds the `AddNew` row at the bottom of a group.

	ID	losses	School	Sport	ties	wins	year
*							
	[+]	Sport: Baseball - 7 Items					
		[+]	Sport: Basketball - 8 Items				
*							

Figure 291: AddNew Rows

### GroupCaptionSection

This is the first section within a group that provides a caption bar above the column headers. The GroupCaptionRows are unbound rows that are created only to combine the records into a group. By default, they display the group category and the number of items in that group. The following properties can be used to control the CaptionSection display.

- **<GroupOptions>.ShowCaption** - enables the display of caption section; True by default.
- **<GroupOptions>.CaptionText** - used to get / set the caption text.

ID	losses	School	Sport	ties	wins	year
	[+]	Sport: Baseball - 7 Items				
20	23	Maryland	Baseball	0	34	2003
21	13	Wake Forest	Baseball	0	47	2003

Figure 292: Group Caption Section

### CaptionText Tokens

The following table lists the available token formats for **<GroupOptions>.CaptionText**.

Token	Description
{TableName}	Displays the CaptionSection.ParentTableDescriptor.Name.
{CategoryName}	Displays the CaptionSection.ParentGroup.Name.
{CategoryCaption}	Displays the Header Text of the column that this group belongs to.

{Category}	Displays the CaptionSection.ParentGroup.Category.
{RecordCount}	Displays the CaptionSection.ParentGroup.GetFilteredRecordCount().
Summary Tokens	Allows you to display any item you enter as a Summary Column. See discussion below.

### Custom Summary Tokens

Any summary item you add can be included in the CaptionText. You have the option of hiding summaries, so it is possible to add summaries only for the purpose of displaying values in the CaptionText. If you have added a summary row named Row1, and a Summary columns named Column1, then you can also use the value of this summary item in the caption with the token {Row1.Column1}.

### Example

Here is a sample implementation that illustrates the usage of the above properties.

5. Set up a Grid Grouping control and bind a data source into it.
6. Setup the necessary Group Options as required.

#### [C#]

```
// Group Options Settings.
this.gridGroupingControl1.ShowGroupDropArea = true;
this.gridGroupingControl1.TopLevelGroupOptions.ShowGroupHeader = true;
this.gridGroupingControl1.TopLevelGroupOptions.ShowGroupFooter = true;
this.gridGroupingControl1.TopLevelGroupOptions.ShowCaption = true;
this.gridGroupingControl1.TopLevelGroupOptions.ShowGroupPreview = true;
this.gridGroupingControl1.ChildGroupOptions.ShowGroupPreview = true;
this.gridGroupingControl1.TableOptions.GroupFooterSectionHeight = 30;
this.gridGroupingControl1.TableOptions.GroupHeaderSectionHeight = 30;
this.gridGroupingControl1.TableOptions.GroupPreviewSectionHeight = 25;
this.gridGroupingControl1.TopLevelGroupOptions.ShowAddNewRecordBeforeDetails = true;
this.gridGroupingControl1.TopLevelGroupOptions.ShowAddNewRecordAfterDetails = true;
this.gridGroupingControl1.ChildGroupOptions.CaptionText = "There are
{RecordCount} items under {CategoryName} : {Category}";
```

#### [VB .NET]

```
' Group Options Settings.
```

```
Me.gridGroupingControll1.ShowGroupDropArea = True
Me.gridGroupingControll1.TopLevelGroupOptions.ShowGroupHeader = True
Me.gridGroupingControll1.TopLevelGroupOptions.ShowGroupFooter = True
Me.gridGroupingControll1.TopLevelGroupOptions.ShowCaption = True
Me.gridGroupingControll1.TopLevelGroupOptions.ShowGroupPreview = True
Me.gridGroupingControll1.ChildGroupOptions.ShowGroupPreview = True
Me.gridGroupingControll1.TableOptions.GroupFooterSectionHeight = 30
Me.gridGroupingControll1.TableOptions.GroupHeaderSectionHeight = 30
Me.gridGroupingControll1.TableOptions.GroupPreviewSectionHeight = 25
Me.gridGroupingControll1.TopLevelGroupOptions.ShowAddNewRecordBeforeDetails = True
Me.gridGroupingControll1.TopLevelGroupOptions.ShowAddNewRecordAfterDetails = True
Me.gridGroupingControll1.ChildGroupOptions.CaptionText = "There are {RecordCount} items under {CategoryName} : {Category}"
```

7. Handle the QueryCellStyleInfo event to manipulate the group elements.

[C#]

```
this.gridGroupingControll1.QueryCellStyleInfo += new
GridTableCellStyleInfoEventHandler(gridGroupingControll1_QueryCellStyleInfo);

void gridGroupingControll1_QueryCellStyleInfo(object sender,
GridTableCellStyleInfoEventArgs e)
{
    if (e.TableCellIdentity.TableCellType ==
GridTableCellType.GroupFooterSectionCell ||
e.TableCellIdentity.TableCellType ==
GridTableCellType.GroupHeaderSectionCell)
    {
        e.Style.Enabled = false;
        if (e.TableCellIdentity.TableCellType ==
GridTableCellType.GroupFooterSectionCell)
            e.Style.Text = "The details in the footer can be placed by
enabling ShowGroupFooter and handling QueryCellStyleInfo";
        if (e.TableCellIdentity.TableCellType ==
GridTableCellType.GroupHeaderSectionCell)
            e.Style.Text = "The details in the header can be placed by
enabling ShowGroupHeader and handling QueryCellStyleInfo";
    }

    if (e.TableCellIdentity.TableCellType ==
GridTableCellType.GroupPreviewCell)
    {
        Element el = e.TableCellIdentity.DisplayElement;
        e.Style.CellValue = "Preview notes for Group (" +
```

```
el.ParentGroup.Name + ":" + el.ParentGroup.Category.ToString() + ")";
}
```

**[VB.NET]**

```
Private Sub gridGroupingControll1_QueryCellStyleInfo(ByVal sender As Object, ByVal e As GridTableCellStyleInfoEventArgs) Handles gridGroupingControll1.QueryCellStyleInfo

If e.TableCellIdentity.TableCellType =
GridTableCellType.GroupFooterSectionCell OrElse
e.TableCellIdentity.TableCellType =
GridTableCellType.GroupHeaderSectionCell Then
    e.Style.Enabled = False
    If e.TableCellIdentity.TableCellType =
GridTableCellType.GroupFooterSectionCell Then
        e.Style.Text = "The details in the footer can be placed by
enabling ShowGroupFooter and handling QueryCellStyleInfo"
    End If
    If e.TableCellIdentity.TableCellType =
GridTableCellType.GroupHeaderSectionCell Then
        e.Style.Text = "The details in the header can be placed by
enabling ShowGroupHeader and handling QueryCellStyleInfo"
    End If
    End If

If e.TableCellIdentity.TableCellType =
GridTableCellType.GroupPreviewCell Then
    Dim el As Element = e.TableCellIdentity.DisplayElement
    e.Style.CellValue = "Preview notes for Group (" &
    el.ParentGroup.Name & ":" & el.ParentGroup.Category.ToString() &
    ")"
End If
End Sub
```

8. You can control the appearance of different group elements by using the **Appearance** property.

**[C#]**

```
this.gridGroupingControll1.Appearance.AddNewRecordFieldCell.Interior =
new BrushInfo(Color.FromArgb(255, 255, 192));
this.gridGroupingControll1.Appearance.GroupCaptionCell.Interior = new
BrushInfo(SystemColors.Control);
this.gridGroupingControll1.Appearance.GroupCaptionCell.TextColor =
Color.FromArgb(192, 64, 0);
this.gridGroupingControll1.Appearance.GroupFooterSectionCell.Interior =
```

```
new BrushInfo(Color.Pink);
this.gridGroupingControll1.Appearance.GroupHeaderSectionCell.Interior =
new BrushInfo(Color.Pink);
this.gridGroupingControll1.Appearance.GroupIndentCell.Interior = new
BrushInfo(Color.FromArgb(192, 192, 255));
this.gridGroupingControll1.Appearance.GroupPreviewCell.Interior = new
BrushInfo(Color.FromArgb(192, 255, 192));
```

**[VB.NET]**

```
Me.gridGroupingControll1.Appearance.AddNewRecordFieldCell.Interior = New
BrushInfo(Color.FromArgb(255, 255, 192))
Me.gridGroupingControll1.Appearance.GroupCaptionCell.Interior = New
BrushInfo(SystemColors.Control)
Me.gridGroupingControll1.Appearance.GroupCaptionCell.TextColor =
Color.FromArgb(192, 64, 0)
Me.gridGroupingControll1.Appearance.GroupFooterSectionCell.Interior =
New BrushInfo(Color.Pink)
Me.gridGroupingControll1.Appearance.GroupHeaderSectionCell.Interior =
New BrushInfo(Color.Pink)
Me.gridGroupingControll1.Appearance.GroupIndentCell.Interior = New
BrushInfo(Color.FromArgb(192, 192, 255))
Me.gridGroupingControll1.Appearance.GroupPreviewCell.Interior = New
BrushInfo(Color.FromArgb(192, 255, 192))
```

9. Run the sample and group the table against any data column. Here is a sample screen shot that shows the grouped grid against 'Sport' column.

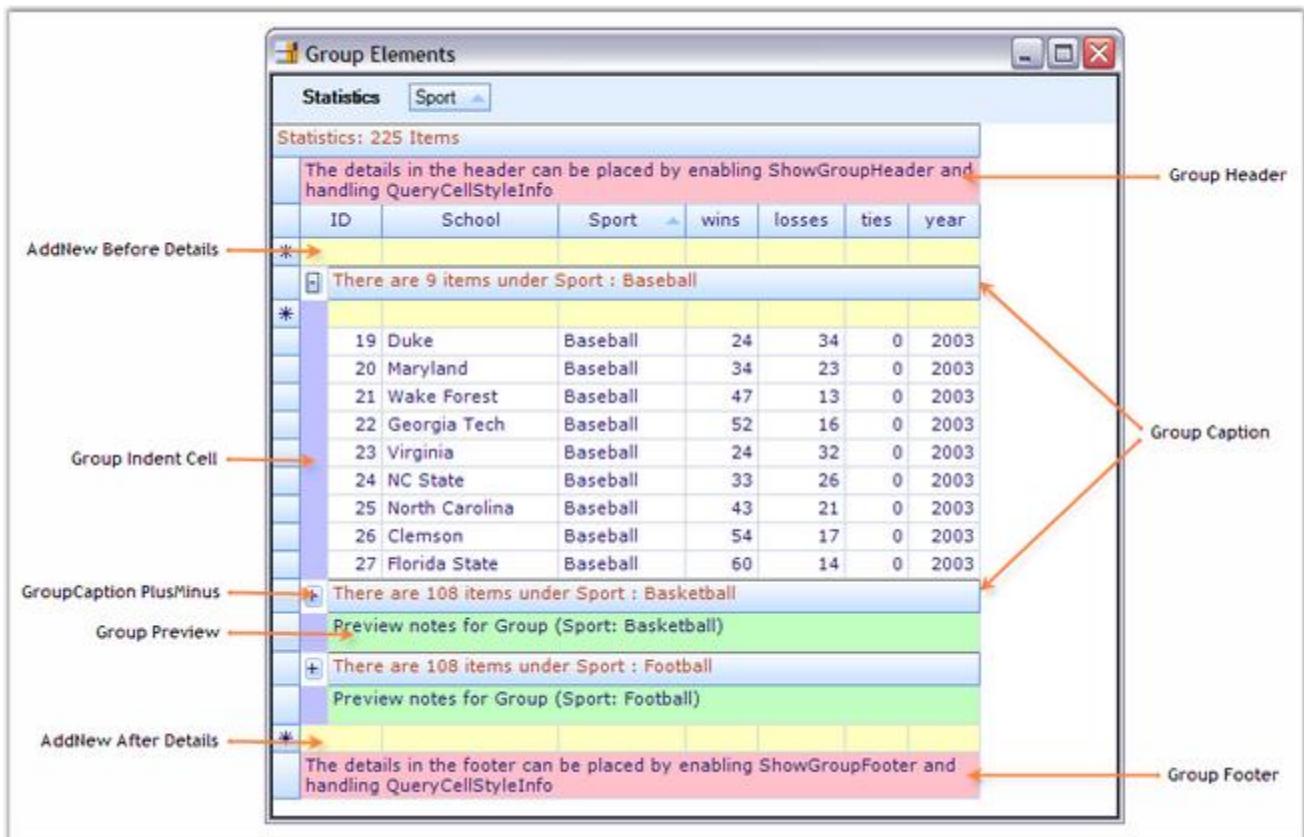


Figure 293: Manipulation and Customization of Group Elements



**Note:** For more details, refer the following browser samples:

- <Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Grouping Grid Options\Top-Level-Group Options Demo
- <Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Grouping Grid Options\Child-Group Options Demo
- <Install Location>\Syncfusion\EssentialStudio\Version Number\Window\Grid.Grouping.Windows\Samples\2.0\Grouping Grid Options\Nested-Table Group Options Demo

#### 4.3.4.3.1.2.2 Working with Groups

This section best demonstrates how to work with the group rows and also shows how the groups are organized into a grouping grid. The Grouping Grid architecture can be viewed as a binary where the different grid elements like group rows, summary rows, filter rows, etc. form the nodes of the tree having the data records at the bottom as leaf nodes. A group can be a final node with records or it can be a node with nested groups rooting a sub tree.

This lesson will guide you the ways to access the individual groups in a collection, to retrieve all the groups, to expand/collapse groups and will discuss some of the properties and events used to process the groups.

### **Expanding/Collapsing Groups**

All the groups can be expanded as well as collapsed at once by calling the respective methods, **Table.ExpandAllGroups** and **Table.CollapseAllGroups**. To expand or collapse a specific group, set **Group.IsExpanded** property to true or false respectively. Following code example illustrates this.

#### [C#]

```
// Expands all groups.  
this.gridGroupingControl1.Table.ExpandAllGroups();  
  
// Collapse all groups.  
this.gridGroupingControl1.Table.CollapseAllGroups();  
  
// Expand the group with index 3.  
this.gridGroupingControl1.Table.TopLevelGroup.Groups[3].IsExpanded =  
true;  
  
// Collapse the group with index 4.  
this.gridGroupingControl1.Table.TopLevelGroup.Groups[4].IsExpanded =  
false;
```

#### [VB.NET]

```
' Expands all groups.  
Me.gridGroupingControl1.Table.ExpandAllGroups()  
  
' Collapse all groups.  
Me.gridGroupingControl1.Table.CollapseAllGroups()  
  
' Expand the group with index 3.  
Me.gridGroupingControl1.Table.TopLevelGroup.Groups(3).IsExpanded = True
```

```
' Collapse the group with index 4.  
Me.gridGroupingControll.Table.TopLevelGroup.Groups(4).IsExpanded =  
False
```

## Accessing a Given Group

The Table.TopLevelGroup.Groups collection maintains the details of the individual groups in this collection which can be used to retrieve the details of any group.

Below code lets you access the details of a group with the category Sport. It also defines a method named IterateGroup that is used to iterate through the records and also the nested groups in a given group. It provides you with the group details such as the level of the group, number of items in that group, its category, and so on.

## Accessing all the groups

Table.TopLevelGroup is the main topmost group in a grouping grid. It forms the root node of the group hierarchy where its categorized records and the nested groups form the child nodes. To access all the groups, you can make use of the same IterateThrough method by passing the TopLevelGroup as the method parameter. Then this method will loop through the categorized records and nested groups of the top level group and will print the details of all the groups.

[C#]

```
// Call IterateThrough method for a given group.  
Group g =  
this.gridGroupingControll.Table.TopLevelGroup.Groups["Sport"];  
IterateGroup(g);  
  
// Call IterateThrough method for all the groups in a grid table.  
IterateGroup(this.gridGroupingControll.Table.TopLevelGroup);  
  
// IterateThrough method iterates through the records and the nested  
groups.  
public void IterateThrough(Group g)  
{  
    System.Diagnostics.Trace.WriteLine("GroupLevel = "+g.GroupLevel);  
    System.Diagnostics.Trace.WriteLine(g.Info);  
    foreach(Record r in g.Records)  
    {  
        System.Diagnostics.Trace.WriteLine(r.Info);  
    }  
}
```

```
foreach(Group gr in g.Groups)
{
    IterateGroup(gr);
}
```

**[VB.NET]**

```
' Call IterateThrough method for a given group.
Dim g As Group
g = Me.gridGroupingControl1.Table.TopLevelGroup.Groups("Sport")
IterateThrough(g)

' Call IterateThrough method for all the groups in a grid table.
IterateThrough(Me.gridGroupingControl1.Table.TopLevelGroup)

'IterateThrough method iterates through the records and the nested
groups.
Public Sub IterateThrough(ByVal g As Group)
    System.Diagnostics.Trace.WriteLine("GroupLevel = "+
        g.GroupLevel.ToString())
    System.Diagnostics.Trace.WriteLine(g.Info)
    For Each r As Record In g.Records
        System.Diagnostics.Trace.WriteLine(r.Info)
    Next r
    For Each gr As Group In g.Groups
        IterateThrough(gr)
    Next gr
End Sub
```

**Accessing the group for a given record**

It is the grid.Table object that provides access to the records and the grouped elements. The Table.Records collection returns a read only collection of the data records. The following code can be used to get access to the group for a particular record. Record.ParentGroup property is used to obtain the group that a record belongs to.

**[C#]**

```
System.Diagnostics.Trace.WriteLine(this.gridGroupingControl1.Table.Records[3].ParentGroup.Info);
```

**[VB.NET]**

```
System.Diagnostics.Trace.WriteLine(Me.gridGroupingControl1.Table.Record
```

```
s(3).ParentGroup.Info)
```

#### 4.3.4.3.1.2.3 Events

This section discusses some of the important events that could be handled to catch the grouping actions. Below is a list of such events.

Event	Description
GroupedColumns_Changing	Occurs before a property in the collection is changed.
GroupedColumns_Changed	Occurs after a property in the collection was changed.
GroupExpanding	Occurs before a group is expanded.
GroupExpanded	Occurs after a group was expanded.
GroupCollapsing	Occurs before a group is collapsed.
GroupCollapsed	Occurs after a group was collapsed.
SortingItemsInGroup	Occurs before the records for a group are sorted.
SortedItemsInGroup	Occurs after the records for a group were sorted.

#### Example

The GroupedColumns Changing/Changed events get fired when the list is modified i.e. when any item is added, removed or modified. It accepts an argument of type ListPropertyChangedEventArgs that lets you check the reason for a list change. The reason could be of ItemAdded, ItemInserted, ItemRemoved, ItemModified, ItemMoved, ItemPropertyChanged or the whole collection is modified.

The following code examples show you how to capture the events.

```
[C#]
```

```
// Subscribe to the events.
```

```
this.gridGroupingControl1.TableDescriptor.GroupedColumns.Changing +=  
new ListPropertyChangedEventHandler(GroupedColumns_Changing);  
this.gridGroupingControl1.TableDescriptor.GroupedColumns.Changed += new  
ListPropertyChangedEventHandler(GroupedColumns_Changed);  
  
// Event Handlers.  
// GroupedColumns_Changing event.  
void GroupedColumns_Changing(object sender,  
ListPropertyChangedEventArgs e)  
{  
    SortColumnDescriptor scd = e.Item as SortColumnDescriptor;  
    if (e.Action ==  
        Syncfusion.Collections.ListPropertyChangedType.Insert)  
        Console.WriteLine("Column Added - {0}", scd.Name);  
}  
  
// GroupedColumns_Changed event.  
void GroupedColumns_Changed(object sender, ListPropertyChangedEventArgs e)  
{  
    SortColumnDescriptor scd = e.Item as SortColumnDescriptor;  
    if (e.Action ==  
        Syncfusion.Collections.ListPropertyChangedType.Remove)  
        Console.WriteLine("Column Removed - {0}", scd.Name);  
}
```

**[VB .NET]**

```
' Subscribe to the events.  
AddHandler  
gridGroupingControl1.TableDescriptor.GroupedColumns.Changing, AddressOf  
GroupedColumns_Changing  
  
' Event Handlers.  
' GroupedColumns_Changing event.  
Private Sub GroupedColumns_Changed(ByVal sender As Object, ByVal e As  
ListPropertyChangedEventArgs)  
Dim scd As SortColumnDescriptor = CType(e.Item, SortColumnDescriptor)  
If e.Action = ListPropertyChangedType.Insert Then  
    Console.WriteLine("Column Added - {0}" + scd.Name)  
End If  
End Sub  
  
' GroupedColumns_Changed event.  
Private Sub GroupedColumns_Changing(ByVal sender As Object, ByVal e As  
ListPropertyChangedEventArgs)  
Dim scd As SortColumnDescriptor = CType(e.Item, SortColumnDescriptor)
```

```
If e.Action = ListPropertyChangedType.Remove Then
    Console.WriteLine("Column Removed - {0}" + scd.Name)
End If
End Sub
```

The [GroupExpanding/GroupExpanded](#) and [GroupCollapsing/GroupCollapsed](#) event handlers are best to use when you want to do some actions as a result of the group operations, GroupExpand and GroupCollapse.

The [SortingItemsInGroup](#) and [SortedItemsInGroup](#) events are raised when the records for a group are sorted. The grid should have at least one group for these events to occur. It does have no relationship with normal sorting. It occurs only when a grouped column is sorted.

### Tracking the changes in Nested Table

It is possible to get notified of the changes in other table descriptor other than the default table. This can be achieved by listening to the **Engine.PropertyChanging** event and using the **GetNestedChildTableDescriptorEvent** method. This method lets you get the information about a change in the table descriptor of the nested child table. For example, when a column was changed in a nested table, the above method allows you to get the details such as the table descriptor of the affected table and the original EventArgs which was raised in response to the column changes(eg. ColumnsChanged event). Once you have the event data, you can then check for whether just the width of the column has changed or if other settings were changed.

Following code example illustrates the usage of this method.

[C#]

```
protected override void Engine_PropertyChanging(object sender,
DescriptorPropertyChangedEventArgs e)
{
    if (e.PropertyName == "TableDescriptor")
    {
        TableDescriptor tableDescriptor = ((Engine)
sender).TableDescriptor;
        e = (DescriptorPropertyChangedEventArgs) e.Inner;

        if (e.PropertyName == "Relations")
            e = e.GetNestedChildTableDescriptorEvent(ref tableDescriptor);

        if (e.PropertyName == "Columns")
        {
            ListPropertyChangedEventArgs le =
(ListPropertyChangedEventArgs) e.Inner;
```

```
if (le.Action == ListPropertyChangedType.ItemPropertyChanged)
{
    if (le.Property == "Appearance" || le.Property == "Width"
    || le.Property == "ReadOnly" || le.Property == "HeaderText"
    || le.Action == ListPropertyChangedType.Remove
    || le.Action == ListPropertyChangedType.Move
    || le.Property == "AllowFilter")
    {
        return;
    }
}
else if (e.PropertyName == "Appearance")
{
    // Base class will end edit mode, which is not necessary.
    return;
}
base.Engine_PropertyChanging (sender, e);
}
```

**[VB.NET]**

```
Protected Overrides Sub Engine_PropertyChanging(ByVal sender As Object,
ByVal e As DescriptorPropertyChangedEventArgs)
    If e.PropertyName = "TableDescriptor" Then
        Dim tableDescriptor As TableDescriptor = CType(sender,
        Engine)).TableDescriptor
        e = CType(e.Inner, DescriptorPropertyChangedEventArgs)

    If e.PropertyName = "Relations" Then
        e = e.GetNestedChildTableDescriptorEvent(tableDescriptor)
    End If

    If e.PropertyName = "Columns" Then
        Dim le As ListPropertyChangedEventArgs = CType(e.Inner,
        ListPropertyChangedEventArgs)
        If le.Action = ListPropertyChangedType.ItemPropertyChanged Then
            If le.Property = "Appearance" OrElse le.Property = "Width"
            OrElse le.Property = "ReadOnly" OrElse le.Property =
            "HeaderText" OrElse le.Action = ListPropertyChangedType.Remove
            OrElse le.Action = ListPropertyChangedType.Move OrElse
            le.Property = "AllowFilter" Then
                Return
            End If
        End If
    End If
End If
```

```
ElseIf e.PropertyName = "Appearance" Then  
  
    ' Base class will end edit mode, which is not necessary.  
    Return  
End If  
  
 MyBase.Engine_PropertyChanging(sender, e)  
End Sub
```

#### 4.3.4.3.2 Sorting

Grid Grouping control allows you to sort the table data against one or more columns. The number of columns by which the data can be sorted is unlimited. When sorting is applied, the grid will rearrange the data to match with the current sort criteria.

##### SortedColumns Collection

The **SortedColumns** collection defines the sort order for records within groups. Multiple entries can be added with the first entry having the highest precedence when sorting records. The properties and methods in this collection lets you manage the elements in the collection. The collection can be viewed as a set of SortColumnDescriptors one for every column against which the data is sorted. The SortColumnDescriptor of a field contains the details like the name of a field, sort direction and optionally a custom comparer and categorization object. The custom comparer and categorizer will allow you to customize the sorting.

##### Sorting Methods

There are multiple ways through which you can apply sorting on table data. A simple one is just clicking the desired column headers by which the grouping grid needs to be sorted. Once the sorting is applied, the grid will display a sort icon in the respective column headers showing the sort direction. The sorting can also be done against multiple columns by holding the Ctrl key and doing a click on the desired column headers.

##### Through Designer

At design time, the data can be sorted by accessing **SortedColumns** property under the TableDescriptor section in the property grid of the Grid Grouping control. This will open the SortColumnDescriptorCollection Editor. In that Editor, clicking the Add button will add the existing columns into the collection. The **Name** and **SortDirection** in the property window of the editor will let you specify your desired field name to sort and the sort order. The image given below illustrates this process.

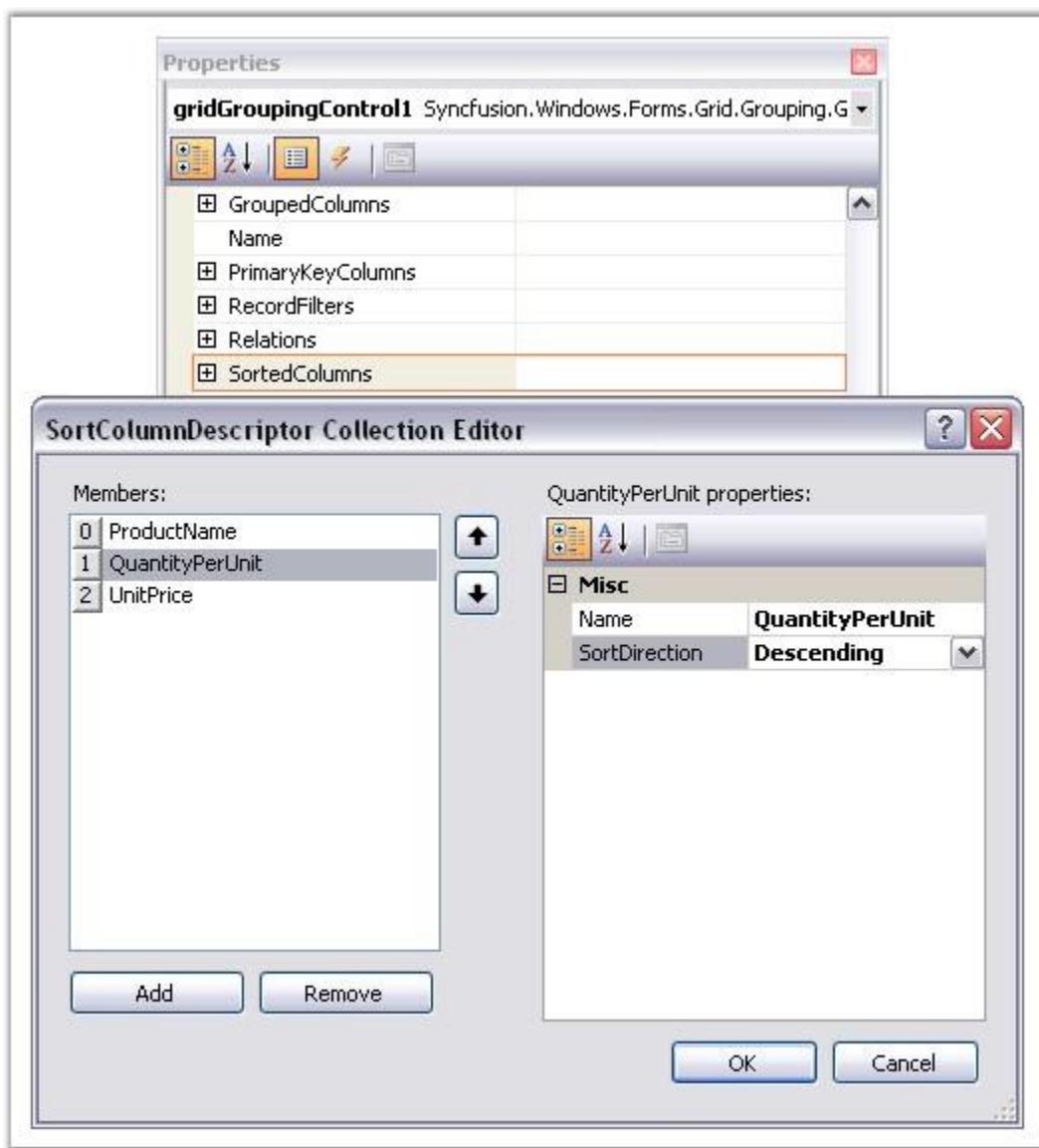


Figure 294: SortColumnDescriptor Collection Editor

### Programmatically

Sorting can be applied to the grid data by specifying the desired field name to the `TableDescriptor.SortedColumns.Add()` method.

**[C#]**

```
this.gridGroupingControl1.TableDescriptor.SortedColumns.Add("ProductName");
```

**[VB .NET]**

```
Me.gridGroupingControl1.TableDescriptor.SortedColumns.Add("ProductName")
```

**Multicolumn Sorting** can be achieved by adding the field names into the **SortedColumns** property and optionally specifying the sort direction. The following code example sorts the data by the ProductName and UnitPrice in ascending Order and by the column Quantity in descending Order.

**[C#]**

```
this.gridGroupingControl1.TableDescriptor.SortedColumns.Add("ProductName", ListSortDirection.Ascending);
this.gridGroupingControl1.TableDescriptor.SortedColumns.Add("QuantityPerUnit", ListSortDirection.Descending);
this.gridGroupingControl1.TableDescriptor.SortedColumns.Add("UnitPrice", ListSortDirection.Ascending);
```

**[VB .NET]**

```
Me.gridGroupingControl1.TableDescriptor.SortedColumns.Add("ProductName", ListSortDirection.Ascending)
Me.gridGroupingControl1.TableDescriptor.SortedColumns.Add("QuantityPerUnit", ListSortDirection.Descending)
Me.gridGroupingControl1.TableDescriptor.SortedColumns.Add("UnitPrice", ListSortDirection.Ascending)
```

Here is a sample output. To indicate the sort direction, a sort icon will be displayed in the column headers. When multicolumn sorting is applied, an index number will be displayed in the column headers along with the sort icon that facilitates the sort order. In the below example, the order of the sorting would be **ProductName(0)**, **Quantity(1)** and then **UnitPrice(2)**.

ProductID	ProductName	QuantityPerUnit	UnitPrice
17	Alice Mutton	20 - 1 kg tins	39
3	Aniseed Syrup	12 - 550 ml bottles	10
40	Boston Crab Meat	24 - 4 oz tins	18.4
60	Camembert Pierrot	15 - 300 g rounds	34
18	Carnarvon Tigers	16 kg pkg.	62.5
1	Chai	10 boxes x 20 bags	18
2	Chang	24 - 12 oz bottles	19
39	Chartreuse verte	750 cc per bottle	18
4	Chef Anton's Cajun Seasoning	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	36 boxes	21.35
48	Chocolade	10 pkgs.	12.75
38	Côte de Blaye	12 - 75 cl bottles	263.5
58	Escargots de Bourgogne	24 pieces	13.25
50	Filo Miv	16 - 2 kg boxes	7

Figure 295: Multi Column Sorting



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Sort\Multi Column Sort Demo

### Sorting By Display Member

Grid Grouping control sorts the grid based on the Value member of the grid data, by default. The user can also sort the grid data by Display members of the foreign-key combo boxes by setting up a foreign-key reference relation between the related tables.



**Note:** A foreign-key reference relation allows the user to look up values in a related table using an id column in the main table.

The following code example illustrates the usage of foreign key relation:

1. Save the location of the mainTable.Customer column, so that it can be swapped after the foreign table reference has been set.

[C#]

```
GridTableDescriptor td = this.gridGroupingControl1.TableDescriptor;
td.VisibleColumns.LoadDefault();
```

```
int lookUpIndex = td.VisibleColumns.IndexOf("Customer");
```

**[VB.NET]**

```
Dim td As GridTableDescriptor = Me.gridGroupingControl1.TableDescriptor  
td.VisibleColumns.LoadDefault()  
Dim lookUpIndex As Integer = td.VisibleColumns.IndexOf("Customer")
```

2. Add the foreign table to the Engine's source list.

**[C#]**

```
this.gridGroupingControl1.Engine.SourceListSet.Add(ForeignTableName,  
ForeignTable.DefaultView);
```

**[VB.NET]**

```
Me.gridGroupingControl1.Engine.SourceListSet.Add(ForeignTableName,  
ForeignTable.DefaultView)
```

3. Create and setup a RelationKind.ForeignKeyReference relation.

**[C#]**

```
GridRelationDescriptor rd = new GridRelationDescriptor();  
rd.Name = "CustomerColDisplay";  
rd.RelationKind = RelationKind.ForeignKeyReference;  
rd.ChildTableName = ForeignTableName;
```

**[VB.NET]**

```
Dim rd As GridRelationDescriptor = New GridRelationDescriptor()  
rd.Name = "CustomerColDisplay"  
rd.RelationKind = RelationKind.ForeignKeyReference  
rd.ChildTableName = ForeignTableName
```

4. Set any optional properties on the relation.

**[C#]**

```
// Display column.  
rd.ChildTableDescriptor.VisibleColumns.Add("CustomerName");  
  
// Sort it for dropdown display.
```

```
rd.ChildTableDescriptor.SortedColumns.Add("CustomerName");
```

**[VB.NET]**

```
' Display column.  
rd.ChildTableDescriptor.VisibleColumns.Add("CustomerName")  
  
' Sort it for dropdown display.  
rd.ChildTableDescriptor.SortedColumns.Add("CustomerName")
```

5. Add relation descriptor to MainTableDescriptor.

**[C#]**

```
td.Relations.Add(rd);
```

**[VB.NET]**

```
td.Relations.Add(rd)
```

6. Replace mainTable.Customer with foreignTable.CustomerName.

**[C#]**

```
string foreignCustomerColInMainTable = rd.Name + "_" + "CustomerName";  
td.VisibleColumns.Insert(CustomerColIndex,  
foreignCustomerColInMainTable);
```

**[VB.NET]**

```
Dim foreignCustomerColInMainTable As String = rd.Name & "_" &  
"CustomerName"  
td.VisibleColumns.Insert(CustomerColIndex,  
foreignCustomerColInMainTable)
```

Run the application. The following output is generated.



The screenshot shows a Windows application window titled "Sort By Display Member Demo". Inside, there is a data grid with four columns: "rID", "CompanyName", "ShipName", and "Customer". The "Customer" column contains dropdown arrows, indicating it is a foreign key column. The data in the grid is as follows:

rID	CompanyName	ShipName	Customer
1031	Folies gourmandes	Peacock	Anabela Domingues
1036	Wilman Kala	Peacock	Anne Granger
1027	Ernst Handel	White Clover Markets	Catherine Dewey
1025	Consolidated Holdings	Wellington Importadora	Eduardo Saavedra
1024	Drachenblut Delikatessen	Suprêmes délices	Elizabeth Brown
1020	Antonio Moreno Taquería	Vins et alcools Chevalier	Frédérique Citeaux
1028	Bon app'	White Clover Markets	Helvetica Nagy
1026	Folies gourmandes	Ottilie's Käseladen	Janine Labrune
1022	Around the Horn	Hanari Carnes	Karl Jablonski
1035	Toms Spezialitäten	Callahan	Karl Jablonski
1021	Berglunds snabbköp	Toms Spezialitäten	Laurence Lebihan
1033	Let's Stop N Shop	Dodsworth	Matti Karttunen
1029	B's Beverages	Buchanan	Palle Ibsen
1037	Wartian Herkku	Leverling	Paula Parente
1034	QUICK-Stop	Davolio	Pirkko Koskitalo
1030	Folk och fä HB	Suyama	Rita Müller
1023	Blauer See Delikatessen	Victuailles en stock	Victoria Ashworth
1032	Eastern Connection	Leverling	Zbyszek Piestrzeniewicz

Figure 296: Foreign-Key Relation

In the figure above, the CustomerName column is displayed in the Foreign Table where as a column named Customer is located in the Main Table. Customer column holds key values that match the values in a column named CustomerID in the Foreign Table.

Refer following sample in our sample browser:

```
<Install Location>\Syncfusion\EssentialStudio\Version
Number\Windows\Grid.Grouping.Windows\Samples\2.0\Sort\Sort By Display Member
Demo
```

#### 4.3.4.3.2.1 Enable or Disable Sorting

By default, the Grouping Grid supports automatic sorting. When you want to disable this automatic sorting, you can use the following methods to prevent sorting on specific columns.

### Properties used to control Sorting

Sorting on the grid data can be controlled by two boolean properties under TableOptions: **AllowSortColumns** and **AllowMultiColumnSort**. These properties are used to enable and disable the sorting action. They are set to True by default. To prevent sorting against multiple columns, you should set AllowMultiColumnSort to False whereas the AllowSortColumns property should be set to True to allow single column sorting. The screenshot given below highlights these properties in the property window.

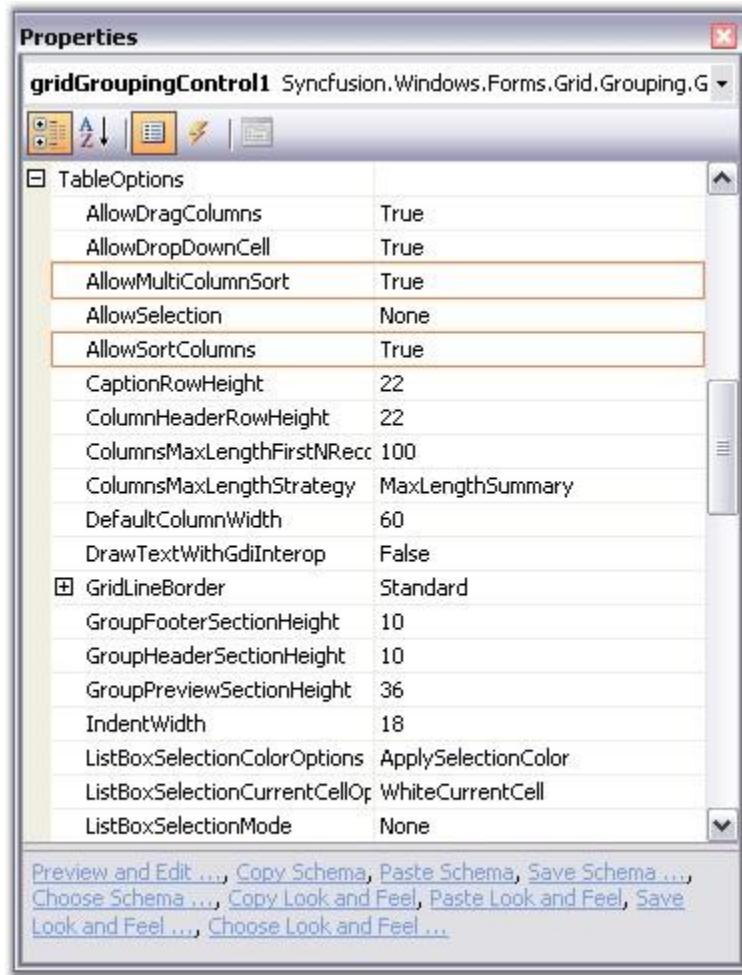


Figure 297: Sorting Properties

### Event used to Prevent Sorting

Sorting on specified columns can also be controlled by handling the TableControlQueryAllowSortColumn event. The event accepts an instance of GridQueryAllowSortColumnEventArgs as a parameter that contains the details of the column being affected. Using this instance, you can check for a particular column and cancel the sorting behavior.

The following code example prevents sorting on the CompanyName field.

**[C#]**

```
this.gridGroupingControl1.TableControlQueryAllowSortColumn+=new  
GridQueryAllowSortColumnEventHandler(gridGroupingControl1_TableControlQ  
ueryAllowSortColumn);  
  
private void  
gridGroupingControl1_TableControlQueryAllowSortColumn(object sender,  
GridQueryAllowSortColumnEventArgs e)  
{  
    if(e.Column.GetName() == "CompanyName")  
    {  
        e.AllowSort=false;  
    }  
}
```

**[VB .NET]**

```
AddHandler gridGroupingControl1.TableControlQueryAllowSortColumn,  
AddressOf gridGroupingControl1_TableControlQueryAllowSortColumn  
Private Sub gridGroupingControl1_TableControlQueryAllowSortColumn(ByVal  
sender As Object, ByVal e As GridQueryAllowSortColumnEventArgs)  
    If e.Column.GetName() = "CompanyName" Then  
        e.AllowSort=False  
    End If  
End Sub
```



**Note:** For more details, refer the following browser sample:

**<Install Location>\Syncfusion\EssentialStudio\Version  
Number\Windows\Grid.Grouping.Windows\Samples\2.0\Sort\Sort Method Demo**

#### **4.3.4.3.3 Summaries**

Grid Grouping control allows you to display summaries for each group. Summaries lets you derive additional information from your data like averages, maximums, summations, count, and so on.

For instance, you can get number of records or maximum value, and so on. They display the calculation results in separate display rows. The calculation of summary values is very fast with only  $O(\log_2 n)$  operations ( $n$  being the number of records in the table), because of the highly optimized balanced tree structures used in the grouping engine.

The grouping grid provides the following built-in summary types.

- Int32Aggregate, DoubleAggregate (Count, Min, Max, Sum)
- StringAggregate (MaxLength, Count)
- Count
- DistinctCount (Count, Values array)
- Vector (Values)
- DoubleVector (statistical methods: Median, Min, Max, 25% Quartile, 75% Quartile)
- Custom (Custom Summaries)

The engine supports summaries that operate on vectors such as Distinct Count, Median, 25% and 75% Quartile. Users may also easily add custom summaries.

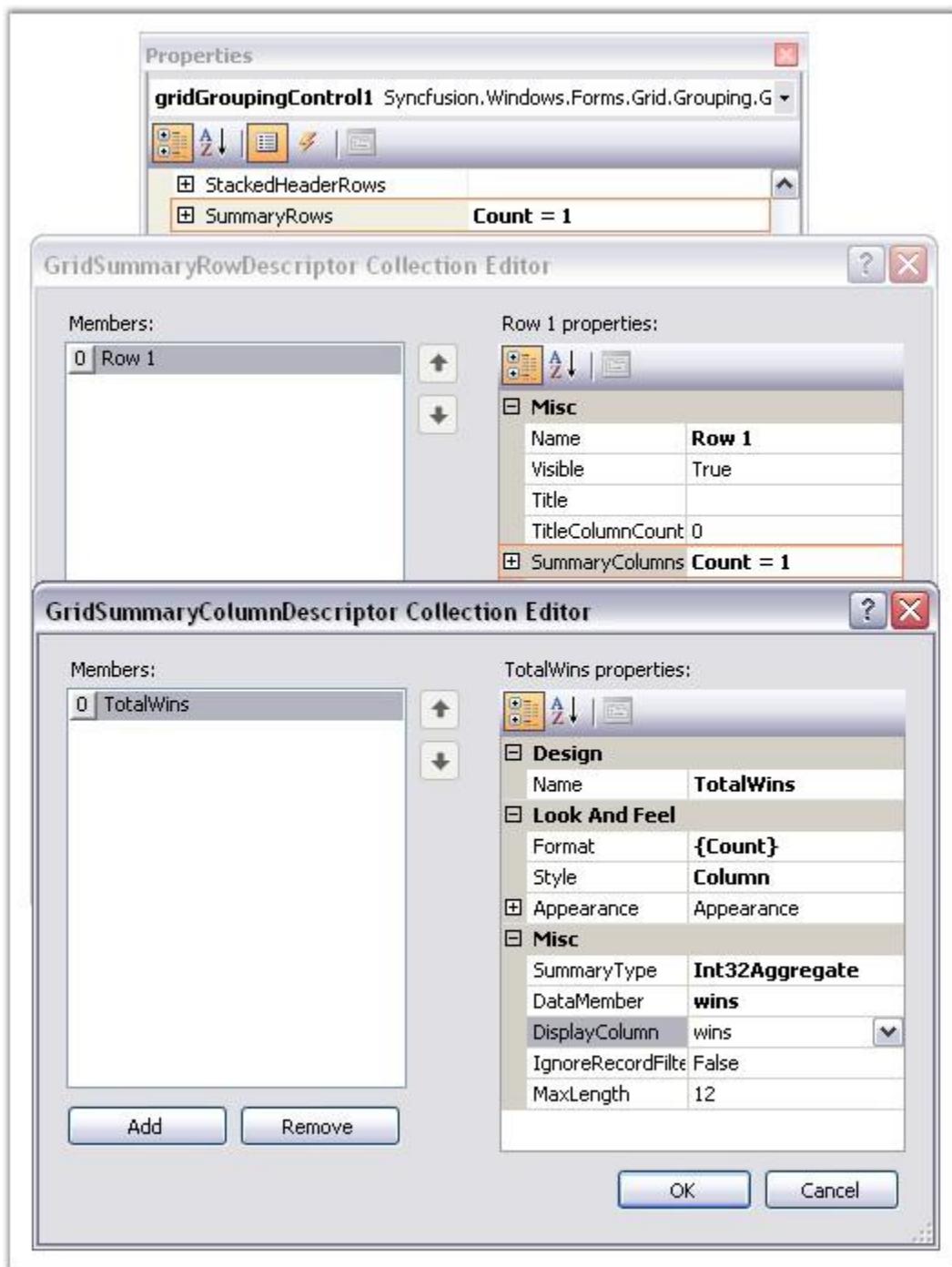
### **SummaryRows Collection**

The **TableDescriptor.SummaryRows** manages a collection of summary rows for the grid table. This collection implements an abstracted view to summaries which lets users define where to display the summary in the grid. Behind the scenes, the GridEngine adds many more hidden summaries to the Summaries collection. Examples for such hidden summaries are: maximum width for contents of a cell, filter bar choices, and display entries of a ForeignKeyKeyWords relation. You can have summaries for the individual columns (**SummaryColumns**) which can then be combined into a single Summary Row for display.

It is the SummaryDescriptorCollection that manages the summaries for a given table containing one entry for each summary. Each SummaryDescriptor in this collection has a MappingName that identifies a FieldDescriptor for which summaries should be calculated for, and a SummaryType property that defines the type of calculations to be performed. Possible options for SummaryType are: Count, BooleanAggregate, ByteAggregate, CharAggregate, DistinctCount, DoubleAggregate, Int32Aggregate, MaxLength, StringAggregate, Vector, DoubleVector, and Custom. By default, a SummaryDescriptor ignores the records that do not satisfy a filter criteria. This behavior can be changed with the IgnoreRecordFilterCriteria flag.

### **Summaries Through Designer**

Summaries can be set at design time itself through the property window of the grid grouping control. In the property window, the SummaryRows under the TableDescriptor node will let you manage the summaries for a grouping grid. Accessing the SummaryRows property will open the GridSummaryRowDescriptor collection editor. The editor contains a list of properties such as Title, SummaryColumn, Appearance, etc. that allows you to define the summaries for the desired columns and to control the appearance of these summaries.



*Figure 298: Property Settings to create a Summary for "Wins" Column*

### Through Code

This example shows a grouping grid bound with a Statistics table whose columns are ID, School, Sport, wins, losses, ties and year. Follow the steps below to create a summary for wins column that displays the sum of wins's values.

1. Setup a `SummaryColumn` by instantiating `GridSummaryColumnDescriptor` specifying the `SummaryType` and format.

#### [C#]

```
GridSummaryColumnDescriptor scd = new GridSummaryColumnDescriptor();
scd.Appearance.AnySummaryCell.Interior = new
BrushInfo(Color.FromArgb(192, 255, 162));
scd.DataMember = "wins";
scd.Format = "{Sum}";
scd.Name = "TotalWins";
scd.SummaryType = SummaryType.Int32Aggregate;
```

#### [VB .NET]

```
Dim scd As GridSummaryColumnDescriptor = New
GridSummaryColumnDescriptor()
scd.Appearance.AnySummaryCell.Interior = New
BrushInfo(Color.FromArgb(192, 255, 162))
scd.DataMember = "wins"
scd.Format = "{Sum}"
scd.Name = "TotalWins"
scd.SummaryType = SummaryType.Int32Aggregate
```

2. Define a `SummaryRow` and add the `SummaryColumn` into it.

#### [C#]

```
GridSummaryRowDescriptor srd = new GridSummaryRowDescriptor();
srd.SummaryColumns.Add(scd);
srd.Appearance.AnySummaryCell.Interior = new
BrushInfo(Color.FromArgb(255, 231, 162));
```

#### [VB .NET]

```
Dim srd As GridSummaryRowDescriptor = New GridSummaryRowDescriptor()
```

```
srd.SummaryColumns.Add(scd)
srd.Appearance.AnySummaryCell.Interior = new
BrushInfo(Color.FromArgb(255, 231, 162))
```

3. Finally add the Summary Row into the grouping grid.

[C#]

```
this.gridGroupingControl1.TableDescriptor.SummaryRows.Add(srd);
```

[VB.NET]

```
Me.gridGroupingControl1.TableDescriptor.SummaryRows.Add(srd)
```

4. Run the sample. The grid will look like this.

The screenshot shows a Windows Forms application window containing a grid control. The grid has columns labeled ID, School, Sport, wins, losses, ties, and year. Rows 214 through 225 represent individual entries for Wake Forest. A summary row is added at the bottom, containing the value 3161 for the 'wins' column. The 'Summary Row' is highlighted with a blue border, and the 'Summary Column' is highlighted with a green background. Arrows point from the text labels to their respective grid elements.

ID	School	Sport	wins	losses	ties	year
214	Virginia	Football	7	4	0	1992
215	Wake Forest	Football	7	5	0	2002
216	Wake Forest	Football	6	5	0	2001
217	Wake Forest	Football	2	9	0	2000
218	Wake Forest	Football	7	5	0	1999
219	Wake Forest	Football	3	8	0	1998
220	Wake Forest	Football	5	6	0	1997
221	Wake Forest	Football	3	8	0	1996
222	Wake Forest	Football	1	10	0	1995
223	Wake Forest	Football	3	8	0	1994
224	Wake Forest	Football	2	9	0	1993
225	Wake Forest	Football	8	4	0	1992
Summary Row						
3161						

Figure 299: Summary created Through Code for "Wins" Column



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\Version  
Number\Windows\Grid.Grouping.Windows\Samples\2.0\Calculate Summary\Summary Tutorial

#### 4.3.4.3.3.1 Exploring Summaries

In the previous chapter, we have learnt how to create simple summaries for a grid table. This chapter will explore the summaries into one more level to discuss the different forms of summaries. It is possible to have multiple summary rows for a single data table. We can define a summary for each group and also for each table when nested tables are used.

#### Multicolumn Summaries

A Summary Row can have any number of summary columns. To display summaries for more than one field, you must first create the summary columns for the desired fields. Then add those summary columns into a summary row. The code given below illustrates this.

##### [C#]

```
GridSummaryColumnDescriptor scd1 = new
GridSummaryColumnDescriptor("Wins", SummaryType.Int32Aggregate, "wins",
"{Sum}");
scd1.Appearance.AnySummaryCell.Interior = new
BrushInfo(Color.FromArgb(192, 255, 162));

GridSummaryColumnDescriptor scd2 = new
GridSummaryColumnDescriptor("Losses", SummaryType.Int32Aggregate,
"losses", "{Sum}");
scd2.Appearance.AnySummaryCell.Interior = new
BrushInfo(Color.LavenderBlush);

GridSummaryRowDescriptor srd = new GridSummaryRowDescriptor();
srd.SummaryColumns.AddRange(new GridSummaryColumnDescriptor[] { scd1,
scd2 });
srd.Appearance.AnySummaryCell.Interior = new
BrushInfo(Color.FromArgb(255, 231, 162));

this.gridGroupingControll1.TableDescriptor.SummaryRows.Add(srd);
```

##### [VB .NET]

```
Dim scd1 As GridSummaryColumnDescriptor = New
GridSummaryColumnDescriptor("Wins", SummaryType.Int32Aggregate, "wins",
"{Sum}")
scd1.Appearance.AnySummaryCell.Interior = New
BrushInfo(Color.FromArgb(192, 255, 162))

Dim scd2 As GridSummaryColumnDescriptor = New
```

```

GridSummaryColumnDescriptor("Losses", SummaryType.Int32Aggregate,
"losses", "{Sum}")
scd2.Appearance.AnySummaryCell.Interior = New
BrushInfo(Color.LavenderBlush)

Dim srd As GridSummaryRowDescriptor = New GridSummaryRowDescriptor()
srd.SummaryColumns.AddRange(New GridSummaryColumnDescriptor() {scd1,
scd2})
srd.Appearance.AnySummaryCell.Interior = New
BrushInfo(Color.FromArgb(255, 231, 162))

Me.gridGroupingControl1.TableDescriptor.SummaryRows.Add(srd)

```

Here is a sample screenshot displaying the summaries for the columns **wins** and **losses**.

ID	School	Sport	wins	losses	ties	year
216	Wake Forest	Football	6	5	0	2001
217	Wake Forest	Football	2	9	0	2000
218	Wake Forest	Football	7	5	0	1999
219	Wake Forest	Football	3	8	0	1998
220	Wake Forest	Football	5	6	0	1997
221	Wake Forest	Football	3	8	0	1996
222	Wake Forest	Football	1	10	0	1995
223	Wake Forest	Football	3	8	0	1994
224	Wake Forest	Football	2	9	0	1993
225	Wake Forest	Football	8	4	0	1992
			3161	2085		

Figure 300: Multi Column Summaries

### Multi Row Summaries

Grouping Grid allows you to have summaries for more than one row. It is achieved by defining a required number of summary row descriptors. Each of the summary rows can have its own format for calculating the summaries. Here is an example that shows how to add two different summary rows for a grid table.

```

[C#]

GridSummaryColumnDescriptor scd1 = new
GridSummaryColumnDescriptor("Wins", SummaryType.Int32Aggregate, "wins",
"{Sum}");
scd1.Appearance.AnySummaryCell.Interior = new
BrushInfo(Color.FromArgb(192, 255, 162));

GridSummaryColumnDescriptor scd2 = new

```

```
GridSummaryColumnDescriptor("Losses", SummaryType.Int32Aggregate,
"losses", "{Sum}");
scd2.Appearance.AnySummaryCell.Interior = new
BrushInfo(Color.LavenderBlush);

GridSummaryRowDescriptor srd = new GridSummaryRowDescriptor();
srd.SummaryColumns.AddRange(new GridSummaryColumnDescriptor[] { scd1,
scd2 });
srd.Appearance.AnySummaryCell.Interior = new
BrushInfo(Color.FromArgb(255, 231, 162));

GridSummaryColumnDescriptor scd3 = new
GridSummaryColumnDescriptor("Total", SummaryType.Count, "{Count}
Records.");
GridSummaryRowDescriptor srd = new GridSummaryRowDescriptor("Row2",
scd3);
srd2.Appearance.AnySummaryCell.Interior = new
BrushInfo(Color.FromArgb(255, 231, 162));

this.gridGroupingControl1.TableDescriptor.SummaryRows.AddRange(new
GridSummaryRowDescriptor[] { srd1, srd2 });
```

**[VB.NET]**

```
Dim scd1 As GridSummaryColumnDescriptor = New
GridSummaryColumnDescriptor("Wins", SummaryType.Int32Aggregate, "wins",
"{Sum}")
scd1.Appearance.AnySummaryCell.Interior = New
BrushInfo(Color.FromArgb(192, 255, 162))

Dim scd2 As GridSummaryColumnDescriptor = New
GridSummaryColumnDescriptor("Losses", SummaryType.Int32Aggregate,
"losses", "{Sum}")
scd2.Appearance.AnySummaryCell.Interior = New
BrushInfo(Color.LavenderBlush)

Dim srd1 As GridSummaryRowDescriptor = New GridSummaryRowDescriptor()
srd1.SummaryColumns.AddRange(New GridSummaryColumnDescriptor() { scd1,
scd2 })
srd1.Appearance.AnySummaryCell.Interior = New
BrushInfo(Color.FromArgb(255, 231, 162))

Dim scd3 As GridSummaryColumnDescriptor = New
GridSummaryColumnDescriptor("Total", SummaryType.Count, "{Count}
Records.")
Dim srd2 As GridSummaryRowDescriptor = New
GridSummaryRowDescriptor("Row2", scd3)
srd2.Appearance.AnySummaryCell.Interior = New
BrushInfo(Color.FromArgb(255, 231, 162))
```

```
Me.gridGroupingControl1.TableDescriptor.SummaryRows.AddRange (New  
GridSummaryRowDescriptor() {srd1, srd2})
```

Given below is a sample screenshot.

The screenshot shows a Windows application window containing a grid control. The grid has columns labeled ID, School, Sport, wins, losses, ties, and year. The data consists of 10 rows for Wake Forest Football from 1992 to 2002. Below the grid, a summary row is displayed with the values 3161 and 2085, followed by the text '225 Records.'

ID	School	Sport	wins	losses	ties	year
215	Wake Forest	Football	7	5	0	2002
216	Wake Forest	Football	6	5	0	2001
217	Wake Forest	Football	2	9	0	2000
218	Wake Forest	Football	7	5	0	1999
219	Wake Forest	Football	3	8	0	1998
220	Wake Forest	Football	5	6	0	1997
221	Wake Forest	Football	3	8	0	1996
222	Wake Forest	Football	1	10	0	1995
223	Wake Forest	Football	3	8	0	1994
224	Wake Forest	Football	2	9	0	1993
225	Wake Forest	Football	8	4	0	1992
			3161	2085		
225 Records.						

Figure 301: Multi Row Summaries

### Summaries for Nested Tables and Groups

Say your datasource has two tables nested, Orders and Order Details, with summaries for the parent table. The summaries that are set for the top level table are sufficient enough for the groups. You need to define summary rows only for the child tables. It can be achieved by creating summaries through the ChildTableDescriptor. The following code illustrates this process.

[C#]

```
// Adding Summaries for the Parent Table(Orders).  
GridSummaryColumnDescriptor scd = new  
GridSummaryColumnDescriptor("Sum", SummaryType.DoubleAggregate,  
"Freight", "{Sum:#}");  
GridSummaryRowDescriptor srd = new GridSummaryRowDescriptor("Sum", "$",  
scd);  
srd.Appearance.AnyCell.HorizontalAlignment =  
GridHorizontalAlignment.Right;  
srd.Appearance.AnyCell.BackColor = Color.FromArgb(255, 231, 162);  
this.gridGroupingControl1.TableDescriptor.SummaryRows.Add(srd);  
  
// Adding Summaries for the Child Table(Order Details).
```

```
scd = new GridSummaryColumnDescriptor("Sum",
SummaryType.Int32Aggregate, "Quantity", "{Sum:#}");
srd = new GridSummaryRowDescriptor("Sum", "Total", scd);
srd.Appearance.AnyCell.HorizontalAlignment =
GridHorizontalAlignment.Right;
srd.Appearance.AnyCell.BackColor = Color.FromArgb(255, 231, 162);
this.gridGroupingControl1.GetTableDescriptor("Order
Details").SummaryRows.Add(srd);
```

**[VB.NET]**

```
' Adding Summaries for the Parent Table(Orders).
Dim scd As GridSummaryColumnDescriptor = New
GridSummaryColumnDescriptor("Sum", SummaryType.DoubleAggregate,
"Freight", "{Sum:#}")
Dim srd As GridSummaryRowDescriptor = New
GridSummaryRowDescriptor("Sum", "$", scd)
srd.Appearance.AnyCell.HorizontalAlignment =
GridHorizontalAlignment.Right
srd.Appearance.AnyCell.BackColor = Color.FromArgb(255, 231, 162)
Me.gridGroupingControl1.TableDescriptor.SummaryRows.Add(srd)

' Adding Summaries for the Child Table(Order Details).
scd = New GridSummaryColumnDescriptor("Sum",
SummaryType.Int32Aggregate, "Quantity", "{Sum:#}")
srd = New GridSummaryRowDescriptor("Sum", "Total", scd)
srd.Appearance.AnyCell.HorizontalAlignment =
GridHorizontalAlignment.Right
srd.Appearance.AnyCell.BackColor = Color.FromArgb(255, 231, 162)
Me.gridGroupingControl1.GetTableDescriptor("Order
Details").SummaryRows.Add(srd)
```

Here is a sample screen shot.

The screenshot displays a hierarchical data grid with the following structure:

- Root Level:** Orders: 830 Items
- ShipCountry Group:** Austria - 40 Items (highlighted with a blue border)
- ShipCountry Sub-Groups:**
  - Austria - 40 Items
  - Argentina - 16 Items
- CustomerID Data:** (Listed under Argentina group)
 

CustomerID	EmployeeID	Freight	OrderDate	OrderID	RequiredDate
OCEAN	3	29.83	1/9/1997	10409	2/6/1997
RANCH	4	38.82	2/17/1997	10448	3/17/1997
CACTU	8	17.22	4/29/1997	10521	5/27/1997
OCEAN	7	8.12	5/8/1997	10531	6/5/1997
RANCH	4	22.57	10/24/1997	10716	11/21/1997
CACTU	9	1.1	12/17/1997	10782	1/14/1998
CACTU	2	19.76	1/7/1998	10819	2/4/1998
RANCH	9	90.85	1/13/1998	10828	1/27/1998
CACTU	4	2.84	2/11/1998	10881	3/11/1998
OCEAN	4	1.27	2/20/1998	10898	3/20/1998
RANCH	1	63.77	2/27/1998	10916	3/27/1998
CACTU	7	31.51	3/10/1998	10937	3/24/1998
OCEAN	7	49.56	3/18/1998	10958	4/15/1998
OCEAN	8	217.86	3/30/1998	10986	4/27/1998
RANCH	6	3.17	4/13/1998	11019	5/11/1998
- Order Details Sub-Table:** Order Details: 2 Items
 

Discount	ProductID	Quantity	UnitPrice
0	46	3	12
0	49	2	20
<b>Total</b>		<b>5</b>	
- Summary Rows:** (Yellow background rows)
 

CACTU	8	0.33	4/28/1998	11054	5/26/1998
\$		599			
\$		64943			

Annotations on the right side point to specific summary types:

- ChildTable Summary:** Points to the "Total" row in the Order Details sub-table.
- Group Summary:** Points to the summary rows for CACTU and the total amount.
- ParentTable Summary:** Points to the summary rows for the total amount.

Figure 302: Summaries for Nested Tables and Groups



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Calculate Summary\Nested-Table and Group Summary Demo

#### 4.3.4.3.3.2 Summary In Caption

Grid Grouping control provides built-in options to display group summaries for the columns in GroupCaptions instead of creating distinct rows for summaries. It can be easily achieved with few property settings. The below table describes these properties which can be accessed through the **GroupOptions**.

Property Name	Description
ShowCaptionSummaryCells	Decides whether GroupCaptionCells are allowed to display summaries for the columns.
ShowSummaries	Indicates whether summaries are visible.
CaptionSummaryRow	Lets you specify a summary row that should be displayed in the caption cells when ShowCaptionSummaryCells is set to true.
CaptionText	Lets you control the caption text to be displayed.

### Steps to create Caption Summaries

- First, define a summary for the grid table. Then group the table against a data column.

[C#]

```
// Adding Summaries.
GridSummaryColumnDescriptor scd = new
GridSummaryColumnDescriptor("Sum", SummaryType.DoubleAggregate,
"Freight", "{Sum:#}");
GridSummaryRowDescriptor srd = new GridSummaryRowDescriptor("Sum", "$",
scd);
srd.Appearance.AnyCell.HorizontalAlignment =
GridHorizontalAlignment.Right;
srd.Appearance.AnyCell.BackColor = Color.Cornsilk;
this.gridGroupingControll1.GetTableDescriptor("Orders").SummaryRows.Add(
srd);

this.gridGroupingControll1.ShowGroupDropArea = true;
this.gridGroupingControll1.TableDescriptor.GroupedColumns.Add("RequiredD
ate");
```

[VB .NET]

```
' Adding Summaries.
Dim scd As GridSummaryColumnDescriptor = New
GridSummaryColumnDescriptor("Sum", SummaryType.DoubleAggregate,
"Freight", "{Sum:#}")
Dim srd As GridSummaryRowDescriptor = New
```

```
GridSummaryRowDescriptor("Sum", "$", scd)
srd.Appearance.AnyCell.HorizontalAlignment =
GridHorizontalAlignment.Right
srd.Appearance.AnyCell.BackColor = Color.Cornsilk
Me.gridGroupingControll.GetTableDescriptor("Orders").SummaryRows.Add(srd)

Me.gridGroupingControll.ShowGroupDropArea = True
Me.gridGroupingControll.TableDescriptor.GroupedColumns.Add("RequiredData")
```

2. Enable Caption Summaries by setting **ShowCaptionSummaryCells** to **True** and by **turning off** the **ShowSummaries** property which will disable the creation of additional summary rows.

[C#]

```
// Creating summaries in caption.
this.gridGroupingControll.ChildGroupOptions.ShowCaptionSummaryCells =
true;
this.gridGroupingControll.ChildGroupOptions.ShowSummaries = false;
```

[VB.NET]

```
' Creating summaries in caption.
Me.gridGroupingControll.ChildGroupOptions.ShowCaptionSummaryCells =
True
Me.gridGroupingControll.ChildGroupOptions.ShowSummaries = False
```

3. Once caption summaries are enabled, your next step is to specify a summary to be displayed in the Caption Rows. This is done by assigning the summary name to the **CaptionSummaryRow** property. Optionally you can customize the caption text in the way you need.

[C#]

```
this.gridGroupingControll.ChildGroupOptions.CaptionSummaryRow = "Sum";
this.gridGroupingControll.ChildGroupOptions.CaptionText =
"{RecordCount} Items";
```

[VB.NET]

```
Me.gridGroupingControll.ChildGroupOptions.CaptionSummaryRow = "Sum"
Me.gridGroupingControll.ChildGroupOptions.CaptionText = "{RecordCount}
Items"
```

4. Finally, format the caption rows to improve the look and feel.

[C#]

```
// Providing a good look and enabling Caption Summary Cells as Record
Field Cells.
this.gridGroupingControl1.Appearance.GroupCaptionCell.BackColor =
this.gridGroupingControl1.Appearance.RecordFieldCell.BackColor;
this.gridGroupingControl1.Appearance.GroupCaptionCell.Borders.Top = new
GridBorder(GridBorderStyle.Standard);
this.gridGroupingControl1.Appearance.GroupCaptionCell.CellType =
"Static";
```

[VB .NET]

```
' Providing a good look and enabling Caption Summary Cells as Record
Field Cells.
Me.gridGroupingControl1.Appearance.GroupCaptionCell.BackColor =
Me.gridGroupingControl1.Appearance.RecordFieldCell.BackColor
Me.gridGroupingControl1.Appearance.GroupCaptionCell.Borders.Top = New
GridBorder(GridBorderStyle.Standard)
Me.gridGroupingControl1.Appearance.GroupCaptionCell.CellType = "Static"
```

5. When you run the sample, your grid will look similar to this.

The screenshot shows a Windows Form application window titled 'Orders' containing a grid control. The grid displays data from the 'Orders' table, specifically columns CustomerID, EmployeeID, Freight, and OrderDate. The data is grouped by CustomerID, with each group header followed by a plus sign (+) indicating expandable details. Group summaries are shown in the 'Freight' column for each group: 121 for the first group, 35 for the second, 123 for the third, 88 for the fourth, and 138 for the fifth. The last group contains three more rows: OLDWO, QUEEN, and TORTU, with their respective freight values. A sixth group, '4 Items', is partially visible. At the bottom of the grid, there is a summary row labeled '\$' with a value of 64943, followed by a downward arrow icon. Orange arrows point from the text labels 'Group Summary in Caption' and 'Table Summary' to the respective features in the grid. The grid has a standard Windows-style border and title bar.

Figure 303: Grouping Grid With Caption Summaries

Here is another screenshot that shows the grouping grid with Caption Summaries disabled.

The screenshot displays a Windows Form application window titled 'Orders'. At the top, there is a toolbar with a button labeled 'RequiredDate' and a small triangle indicating it's a dropdown. Below the toolbar is a table with four columns: 'CustomerID', 'EmployeeID', 'Freight', and 'OrderDate'. The first three columns have standard data, while the fourth column contains dates and counts of items. The table is grouped by 'RequiredDate'. The first two groups are collapsed, showing their respective summary rows. The third group is expanded, revealing its detail rows. The bottom of the table shows a summary row for the entire group. Arrows point from the text labels to specific parts of the interface: one arrow points to the collapsed group rows with the label 'Group Caption Cells'; another arrow points to the summary row with the label 'Group Summary'; and a third arrow points to the bottom-most summary row with the label 'Table Summary'.

Figure 304: Grouping Grid Without Caption Summaries



**Note:** For more details, refer the following browser sample:

```
<Install Location>\Syncfusion\EssentialStudio\Version
Number\Windows\Grid.Grouping.Windows\Samples\2.0\Calculate Summary\Summary In Caption
Demo
```

#### 4.3.4.3.3.3 Sort By Summary In Caption

This section illustrates how to sort the groups by the values of the summary. By default, when grouping is applied, it sorts the records by the values of the grouped column. When you want to change this default group order to make the grouping to sort the records by the values of group summaries, you have a couple of ways to achieve. You can use your own custom comparer to define the sort order. An alternate solution is to make use of the built-in method, that is specially designed to use in this scenario, named **SetGroupSummaryOrder**.

## SetGroupSummaryOrder Method

This method itself will set up a custom comparer for sorting groups if the groups should be sorted in a different order than the category. It can be defined for a given column say **Col1** by passing the summary name, a property in the summary and optionally the sort direction as the parameters. It makes use of these parameters to retrieve the summary values and then pass these values to a custom comparer which sets up a sort order based on these summary values. When the grid is grouped against the column **Col1**, then the groups are sorted in the order specified by the custom comparer instead of sorting in the default order. Here is a sample usage of this method.

### Example

This example uses an Orders Table bound to a grouping grid. Summaries are created for the column **Freight**. The group caption cells are made to display the group summaries for the **Freight** column. Now, our goal is to sort the table against **ShipCountry** field with the data records get arranged based on the caption summaries i.e. the groups should get sorted against the summary values rather than the category.

Follow these steps to sort the groups by the summary values.

1. Define a Summary Column Descriptor for the column **Freight** and add it into a SummaryRow of the Orders table.

#### [C#]

```
GridSummaryColumnDescriptor summaryColumn1 = new
GridSummaryColumnDescriptor("FreightAverage",
SummaryType.DoubleAggregate, "Freight", "{Average:###.00}");
GridSummaryRowDescriptor summaryRow1 = new GridSummaryRowDescriptor();
summaryRow1.Name = "Caption";
summaryRow1.SummaryColumns.Add(summaryColumn1);
this.gridGroupingControl1.TableDescriptor.SummaryRows.Add(summaryRow1);
```

#### [VB .NET]

```
Dim summaryColumn1 As New GridSummaryColumnDescriptor("FreightAverage",
SummaryType.DoubleAggregate, "Freight", "{Average:###.00}")
Dim summaryRow1 As New GridSummaryRowDescriptor()
summaryRow1.Name = "Caption"
summaryRow1.SummaryColumns.Add(summaryColumn1)
Me.gridGroupingControl1.TableDescriptor.SummaryRows.Add(summaryRow1)
```

2. Trigger caption summaries by setting appropriate properties.

**[C#]**

```
this.gridGroupingControl1.TableDescriptor.ChildGroupOptions.ShowCaptionSummaryCells = true;
this.gridGroupingControl1.TableDescriptor.ChildGroupOptions.CaptionSummaryRow = "Caption";
this.gridGroupingControl1.TableDescriptor.ChildGroupOptions.ShowSummaries = false;
```

**[VB .NET]**

```
Me.gridGroupingControl1.TableDescriptor.ChildGroupOptions.ShowCaptionSummaryCells = True
Me.gridGroupingControl1.TableDescriptor.ChildGroupOptions.CaptionSummaryRow = "Caption"
Me.gridGroupingControl1.TableDescriptor.ChildGroupOptions.ShowSummaries = False
```

3. Create a SortColumnDescriptor for the field ShipCountry. Change the default group order by using the method SetGroupSummaryOrder with its parameters conveying the summary name and the property in the summary. Then group the grid against this column.

**[C#]**

```
// Specify group sort order behavior when adding SortColumnDescriptor to GroupedColumns
this.gridGroupingControl1.TableDescriptor.GroupedColumns.Clear();
SortColumnDescriptor gsd = new SortColumnDescriptor("ShipCountry");

// specify a summary name and the property (values will be determined using reflection)
gsd.SetGroupSummarySortOrder(summaryColumn1.GetSummaryDescriptorName(), "Average");

this.gridGroupingControl1.TableDescriptor.GroupedColumns.Add(gsd);
```

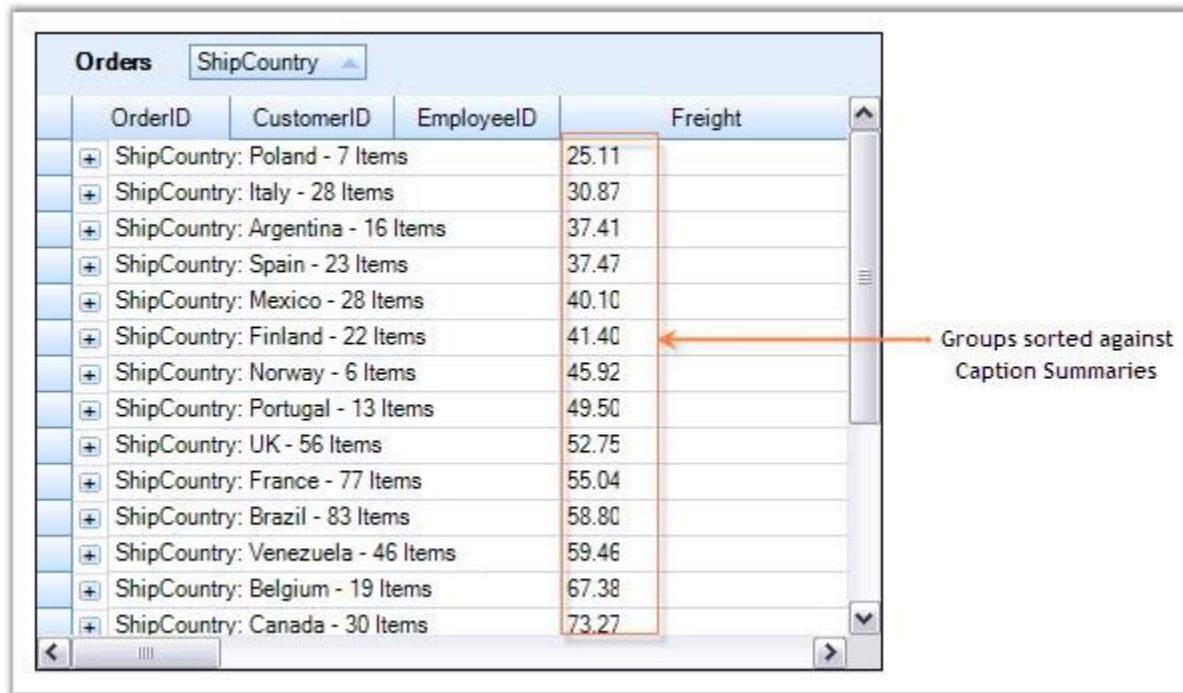
**[VB .NET]**

```
' Specify group sort order behavior when adding SortColumnDescriptor to GroupedColumns
Me.gridGroupingControl1.TableDescriptor.GroupedColumns.Clear()
Dim gsd As New SortColumnDescriptor("ShipCountry")

'specify a summary name and the property (values will be determined using reflection)
```

```
gsd.SetGroupSummarySortOrder(summaryColumn1.GetSummaryDescriptorName(),  
    "Average")  
  
Me.gridGroupingControl1.TableDescriptor.GroupedColumns.Add(gsd)
```

4. When you run the sample, you will see the groups are sorted against the summary values of Freight. Here is a sample screen shot.



The screenshot shows a Windows Forms application window titled 'Orders'. Inside, there is a grouping grid with columns: OrderID, CustomerID, EmployeeID, and Freight. The 'Freight' column is the summary column. The grid displays 14 groups, each starting with a plus sign and followed by a summary description and the number of items. The groups are sorted by their Freight values. An orange arrow points from the caption 'Groups sorted against Caption Summaries' to the first group in the list, which has a Freight value of 25.11.

OrderID	CustomerID	EmployeeID	Freight
ShipCountry: Poland - 7 Items			25.11
ShipCountry: Italy - 28 Items			30.87
ShipCountry: Argentina - 16 Items			37.41
ShipCountry: Spain - 23 Items			37.47
ShipCountry: Mexico - 28 Items			40.10
ShipCountry: Finland - 22 Items			41.40
ShipCountry: Norway - 6 Items			45.92
ShipCountry: Portugal - 13 Items			49.50
ShipCountry: UK - 56 Items			52.75
ShipCountry: France - 77 Items			55.04
ShipCountry: Brazil - 83 Items			58.80
ShipCountry: Venezuela - 46 Items			59.46
ShipCountry: Belgium - 19 Items			67.38
ShipCountry: Canada - 30 Items			73.27

Figure 305: Sorting Groups by Summary Values



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\Version  
Number\Windows\Grid.Grouping.Windows\Samples\2.0\Calculate Summary\Sort by Summary Demo

#### 4.3.4.3.4 Filters and Expressions

Grouping Grid supports record filters and expression fields. Record Filters let you display a subset of records that meets a given filter criteria. Expression Fields are unbound fields added to the grouping grid that can be used to display any calculation results based on other fields in the same record.

Further topics in this section discuss these concepts in detail with suitable examples.

#### *4.3.4.3.4.1 Expression Fields*

**Expression Fields** will allow you to add a column that holds calculated values based on other fields in the same record. These expression columns can be visible or invisible, can be used in grouping and sorting and may be employed as summary fields for summary rows. As with adding Summary Rows and Summary Columns, you can use collection editors to add Expression Fields.

#### **Expression Fields Collection**

The Expression Fields collection of a TableDescriptor defines expression fields. This collection is managed by ExpressionFieldDescriptor collection in which each entry termed as ExpressionFieldDescriptor defines one expression field. The data for expression fields are calculated at runtime based on the ExpressionFieldDescriptor.Expression text formula and can depend on other fields in the same record.

#### **Adding Expression Fields Through Designer**

In the property window of the grouping grid, if you open the TableDescriptor section, you will notice the ExpressionFields collection property. Clicking this will open the ExpressionFieldDescriptor Collection Editor. The editor displays the properties, necessary to setup expression fields. The table given below gives a brief description about some important properties.

Property Name	Description
Name	Specifies the name of the expression field.
Expression	Specifies the formula expression.
ResultType	Lets you specify the result type to which the expression should be converted to.
ForceImmediateSaveValue	Indicates whether the changes to the field in a record should trigger the SaveValue event; Make it to False to avoid triggering ListChanged events when the expression field is modified.

ReferencedFields	Saves a list of referenced field names used in the expression. Use semicolon as a delimiter to specify multiple fields.  This list will be used by the engine to determine which cells to update when ListChanged event is triggered.
------------------	---

You can add any number of expression fields to the table. The following image depicts this.

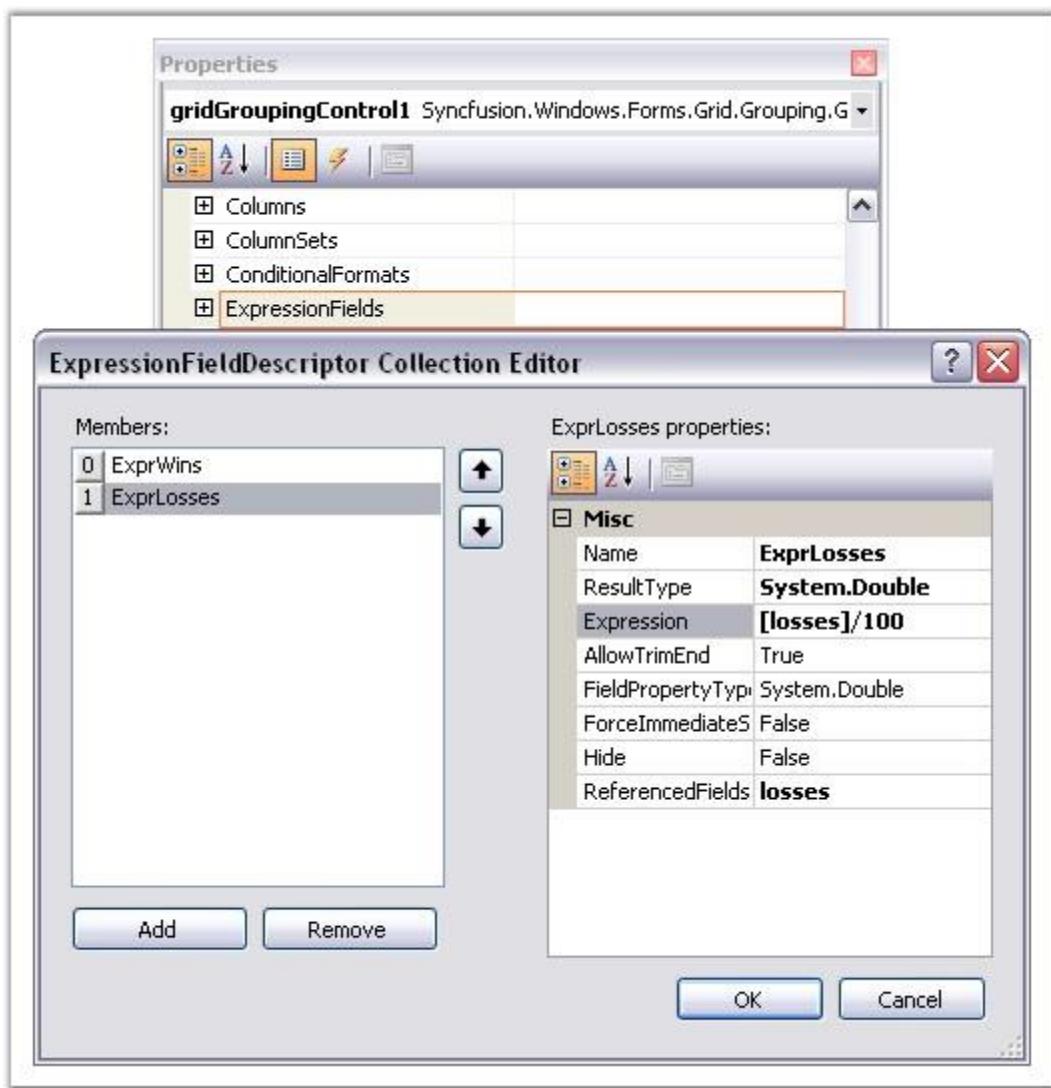


Figure 306: ExpressionFieldDescriptor Collection Editor

## Programmatically

Expression Fields can also be set through code. The following code example adds two expression fields to the Statistics table.

**[C#]**

```
// Define expression fields.  
ExpressionFieldDescriptor exp1 = new ExpressionFieldDescriptor("Winning %",  
"([wins] *100)/([wins]+[ties]+[losses])", "System.Double");  
ExpressionFieldDescriptor exp2 = new ExpressionFieldDescriptor("Loosing %",  
"([losses] *100)/([wins]+[ties]+[losses])", "System.Double");  
  
// Add the expression fields to the grid table.  
this.gridGroupingControl1.TableDescriptor.ExpressionFields.AddRange(new  
Syncfusion.Grouping.ExpressionFieldDescriptor[] { exp1, exp2 });
```

**[VB .NET]**

```
' Define expression fields.  
Dim exp1 As ExpressionFieldDescriptor = New ExpressionFieldDescriptor("Winning %",  
"([wins] *100)/([wins]+[ties]+[losses])", "System.Double")  
Dim exp2 As ExpressionFieldDescriptor = New ExpressionFieldDescriptor("Loosing %",  
"([losses] *100)/([wins]+[ties]+[losses])", "System.Double")  
  
' Add the expression fields to the grid table.  
this.gridGroupingControl1.TableDescriptor.ExpressionFields.AddRange(New  
Syncfusion.Grouping.ExpressionFieldDescriptor() {exp1, exp2})
```

The screen shot given below highlights these expression fields.



ID	losses	School	Sport	ties	wins	year	Winning %	Lossing %
1	7	Duke	Basketball	0	26	2003	78.79	21.21
2	10	Maryland	Basketball	0	21	2003	67.74	32.26
3	6	Wake Forest	Basketball	0	25	2003	80.65	19.35
4	15	Georgia Tech	Basketball	0	16	2003	51.61	48.39
5	16	Virginia	Basketball	0	16	2003	50.00	50.00
6	13	NC State	Basketball	0	18	2003	58.06	41.94
7	16	North Carolina	Basketball	0	19	2003	54.29	45.71
8	13	Clemson	Basketball	0	15	2003	53.57	46.43
9	15	Florida State	Basketball	0	14	2003	48.28	51.72
10	7	Duke	Football	0	3	2003	30.00	70.00
11	3	Maryland	Football	0	6	2003	66.67	33.33
12	5	Wake Forest	Football	0	5	2003	50.00	50.00
13	4	Georgia Tech	Football	0	5	2003	55.56	44.44

Figure 307: Setting Expression Fields Through Code



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Grouping.Windows\Samples\2.0\Filters and Expressions\Expression Field Demo

## See Also

### 4.3.4.3.4.1.1 Nested Expression Fields

Expression fields can be **nested**, means that the formula expression of an expression fields can have reference to other fields. Given below are examples for nested expression fields.

- ExpressionField1.Expression = " [Col1] \* 100 "
- ExpressionField2.Expression = " [ExpressionField1] + 0.5 "
- ExpressionField3.Expression = " [ExpressionField1] + [ExpressionField2] "

## Sample Code

The following code examples are used to create nested expression fields.

**[C#]**

```
// Define expression fields that are nested.  
ExpressionFieldDescriptor expField1 = new  
ExpressionFieldDescriptor("ExpCol1", "[wins]+[ties]+[losses]",  
typeof(System.Double));  
ExpressionFieldDescriptor expField2 = new  
ExpressionFieldDescriptor("ExpCol2", "[ExpCol1]*100",  
typeof(System.Double));  
  
// Add these expression fields to the grid table.  
this.gridGroupingControl1.TableDescriptor.ExpressionFields.AddRange(new  
ExpressionFieldDescriptor[] { expField1, expField2 });  
  
// Appearance Settings.  
this.gridGroupingControl1.TableDescriptor.Columns["ExpCol1"].Appearance  
.AnyRecordFieldCell.BackColor = Color.Cornsilk;  
this.gridGroupingControl1.TableDescriptor.Columns["ExpCol2"].Appearance  
.AnyRecordFieldCell.BackColor = Color.Cornsilk;
```

**[VB .NET]**

```
' Define expression fields that are nested.  
Dim expField1 As ExpressionFieldDescriptor = New  
ExpressionFieldDescriptor("ExpCol1", "[wins]+[ties]+[losses]",  
GetType(System.Double))  
Dim expField2 As ExpressionFieldDescriptor = New  
ExpressionFieldDescriptor("ExpCol2", "[ExpCol1]*100",  
GetType(System.Double))  
  
' Add these expression fields to the grid table.  
Me.gridGroupingControl1.TableDescriptor.ExpressionFields.AddRange(New  
ExpressionFieldDescriptor() {expField1, expField2})  
  
' Appearance Settings.  
Me.gridGroupingControl1.TableDescriptor.Columns("ExpCol1").Appearance.A  
nyRecordFieldCell.BackColor = Color.Cornsilk  
Me.gridGroupingControl1.TableDescriptor.Columns("ExpCol2").Appearance.A  
nyRecordFieldCell.BackColor = Color.Cornsilk
```

Here is a sample screen shot showing two expression fields **ExpCol1** and **ExpCol2** where **ExpCol2** is referencing **ExpCol1**.

	wins	losses	ties	year	ExpCol1	ExpCol2
	26	7	0	2003	33	3300
	21	10	0	2003	31	3100
	25	6	0	2003	31	3100
	16	15	0	2003	31	3100
	16	16	0	2003	32	3200
	18	13	0	2003	31	3100
	19	16	0	2003	35	3500
	15	13	0	2003	28	2800
	14	15	0	2003	29	2900
	3	7	0	2003	10	1000
	6	3	0	2003	9	900
	5	5	0	2003	10	1000
	5	4	0	2003	9	900
	5	4	0	2003	9	900

Figure 308: Nested Expression Fields

#### 4.3.4.3.4.1.2 List of Expressions

##### A note on Valid Expression Syntax

Expressions may be any well-formed algebraic combination of column mapping names enclosed with brackets ([]), numerical constants and literals, and the algebraic and logical operators are listed below.

The computations are performed as listed, with level one operations done first. Alpha constants used with match and like should be enclosed in apostrophes (').

- \*, / : multiplication, division.
- +, - : addition, subtraction.
- <, >, =, <=, >=: less than, greater than, equal, less than or equal.
- match, like, in, between.
- or, and.

Below is the list of operators used and their descriptions.

Expression	Syntax	Description	Example Usage

Multiplication, Division	* , /	Multiplies/Divides first argument by second argument. Multiplies/Divides first argument by second argument.	[Wins] * [Losses] / 100
Addition, Subtraction	+,-	Adds first argument with second argument/ Subtracts second argument from the first one.	[Wins]+[Losses]
Or	OR	Returns 1 if either the first argument or the second one returns true.	[Val]=50 OR [Val]=100
And	AND	Returns 1 if both parameters return true.	[Val]< 50 AND [Val]>100
Less than	<	Returns true if first parameter is less than the second one.	[OrderID] < 2000
Greater than	>	Returns true if first parameter is greater than the second one.	[OrderID] > 2500
Less than Or Equal to	<=	Returns true if first parameter is less than or equal to the second one.	[OrderID] <= 2050
Greater than Or Equal to	>=	Returns true if first parameter is greater than or equal to the second one.	[OrderID] >= 2056
Equal	=	Returns true if both arguments have same value.	[CustomerID] = 90
Not Equal to	<>	Returns true if both arguments does not have same value.	[CustomerID] <> 95

Match	match	Returns 1 if there is any occurrence of the right-hand argument in the left-hand argument. For example, [CompanyName] match 'RTR' returns 0 for any record whose CompanyName field does not contain RTR anywhere in the string.	[Company] match 'Syncfusion'
Like	Like	Checks if the field starts exactly as specified in the right-hand argument. For example, [CompanyName] like 'RTR' returns 1 for any record whose CompanyName field is exactly RTR. You can use an asterisk as a wildcard. [CompanyName] like 'RTR*' returns 1 for any record whose CompanyName field starts with RTR. [CompanyName] like '*RTR' returns 1 for any record whose CompanyName field ends with RTR.	[Sport] like 'Basket*'
In	in	Checks if the field value is any of the values listed in the right-hand operand. The collection of items used as the right-hand should be separated by commas and enclosed with brackets({}). For example, [code] in	[Country] in {"USA", "UK"}

		{1,10,21} returns 1 for any record whose code field contains 1, 10 or 21. [CompanyName] in {RTR,MAS} returns 1 for any record whose CompanyName field is RTR or MAS.	
Between	between	Checks if a date field value between the two values is listed in the right-hand operand. For example, [date] between {2/25/2004, 3/2/2004} returns 1 for any record whose date field is greater or equal 2/25/2004 and less than 3/2/2004. To represent the current date, use the token TODAY. To represent DateTime.MinValue, leave the first argument empty. To represent DateTime.MaxValue, leave the second argument empty.	[OrderDate] between {2/25/2007, TODAY}
Between time	betweentime	Checks if a time in the date field value between the two values is listed in the right-hand operand. For example, [time] between {04:00:00 PM, 05:00:00 PM} returns 1 for any record whose date field is greater than or equal to 04:00 and less than 05:00. The time will be calculated along with the date for betweentime.	[OrderDate] between {"04/17/2008 9:00:00 PM", "04/21/2008 07:00:00 AM"}

#### [4.3.4.3.4.2 RecordFilters](#)

**RecordFilters** otherwise called as **RowFilters** will allow you to restrict displayed records to those that will satisfy the logical condition that you specify with a **FilterRowDescriptor**. You have the option of typing an expression (similar to an Expression Field) or entering a condition using an editor dialog.

#### **RecordFilters Collection**

The RecordFilters collection defines filter criteria for showing or hiding records. Each filter in this collection is internally maintained by a **RecordFilterDescriptor**. All the RecordFilterDescriptors for a given filter are managed by the **RecordFilterDescriptorCollection** which is returned by the RecordFilters property of the TableDescriptor. Filters can be specified through text formulas similar to expression fields or through the entries of **FilterConditionCollection**. FilterConditionCollection is a set of conditions each with a **CompareOperator** and a **CompareValue** to compare with the value retrieved from the record. A condition can also have a custom **ICustomFilter** object if you want to provide your own logic for evaluating filter criteria.

#### **Adding Filters Through Designer**

Record Filters can easily be added through the designer. Opening the TableDescriptor.RecordFilters property in the property window will display the RecordFilterDescriptor Collection Editor that allows you to define record filters. The designer settings shown in the below image will setup a record filter for the field '**wins**', to display only the records with **wins > 20**.

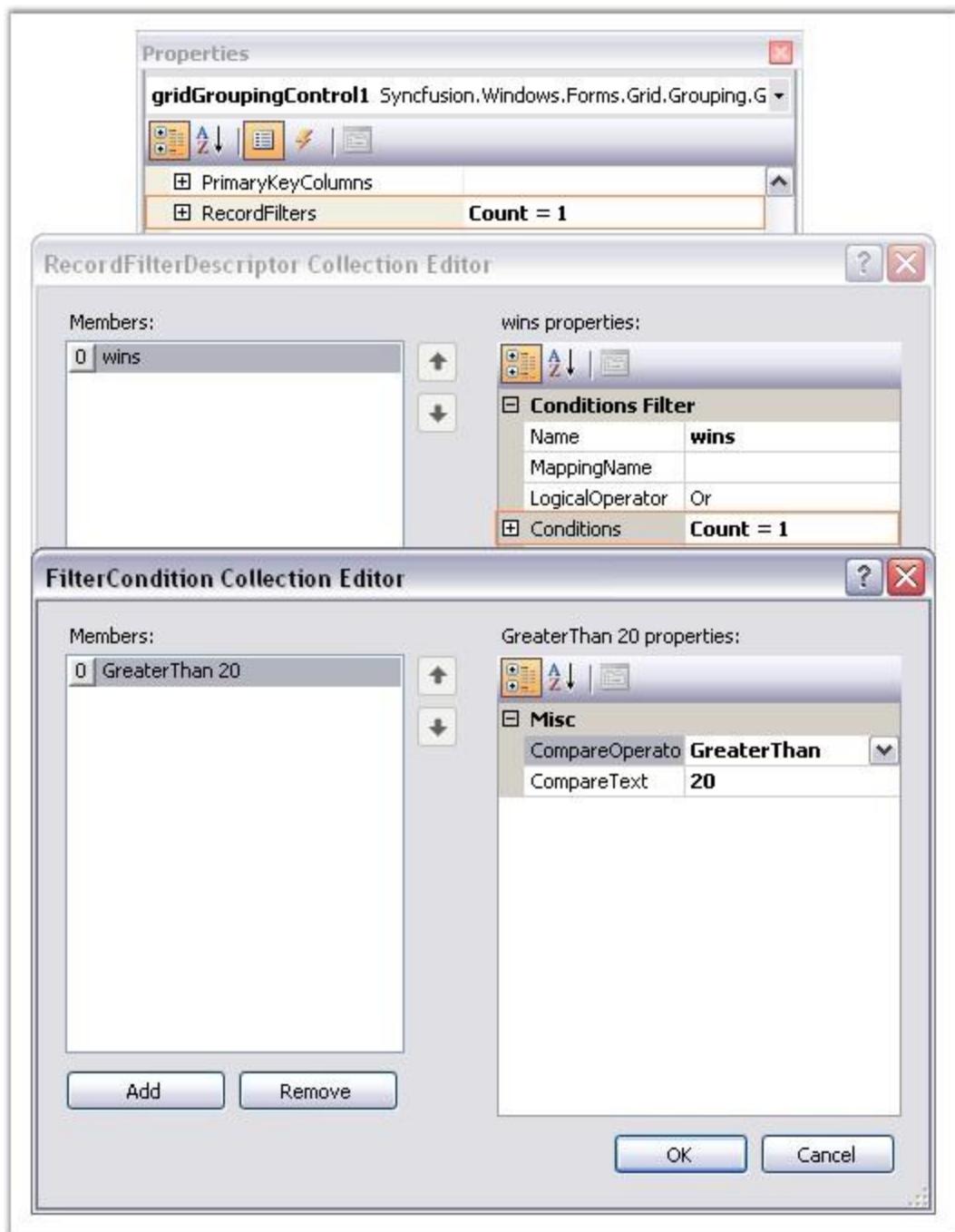


Figure 309: RecordFilterDescriptor Collection Editor

Here is the description for some important properties used to set a row filter.

Property	Description
Name	Specifies the name of the field with which the

	filter is compared.
Conditions	A collection of conditions each with a CompareOperator and a CompareValue.
Expression	A formula expression similar to expression fields.
LogicalOperator	Indicates the logical operator used if multiple conditions are given.

### Programmatically

To add a record filter, you must create a RecordFilterDescriptor by specifying the field name with which the filter should be compared and a filter condition that contains a CompareOperator and a CompareValue. The possible options for a CompareOperator are Equals, NotEquals, LessThan, LessThanOrEqualTo, GreaterThan, GreaterThanOrEqualTo, Like, Match and Custom ( for Custom Filter). A filter criteria can also be specified as an expression text similar to the one used in expression fields. A LogicalOperator will be used when you specify more than one condition for a given filter. Finally add the record filter descriptor to the RecordFilters collection of the Table Descriptor.

Following code example illustrates how to add a record filter for the column "wins" to display only the records with wins > 20.

#### [C#]

```
FilterCondition cond = new FilterCondition(FilterCompareOperator.GreaterThan,
20);
RecordFilterDescriptor filter = new RecordFilterDescriptor("wins", cond);
this.gridGroupingControll1.TableDescriptor.RecordFilters.Add(filter);
```

#### [VB .NET]

```
Dim cond As FilterCondition = New
FilterCondition(FilterCompareOperator.GreaterThan, 20)
Dim filter As RecordFilterDescriptor = New RecordFilterDescriptor("wins", cond)
Me.gridGroupingControll1.TableDescriptor.RecordFilters.Add(filter)
```

Given below is a sample screen shot showing the grid filtered with **wins > 20**.

The screenshot shows a Windows Form application window titled "Essential Grid for Windows Forms". Inside the window is a data grid control displaying a table of sports records. The columns are labeled: ID, losses, School, Sport, ties, wins, and year. The data includes records for Duke, Maryland, Wake Forest, and Georgia Tech in both Basketball and Baseball categories across the years 2002 and 2003. The grid has a standard Windows-style scroll bar on the right and navigation buttons at the bottom.

ID	losses	School	Sport	ties	wins	year
1	7	Duke	Basketball	0	26	2003
2	10	Maryland	Basketball	0	21	2003
3	6	Wake Forest	Basketball	0	25	2003
19	34	Duke	Baseball	0	24	2003
20	23	Maryland	Baseball	0	34	2003
21	13	Wake Forest	Baseball	0	47	2003
22	16	Georgia Tech	Baseball	0	52	2003
23	32	Virginia	Baseball	0	24	2003
24	26	NC State	Baseball	0	33	2003
25	21	North Carolina	Baseball	0	43	2003
26	17	Clemson	Baseball	0	54	2003
27	14	Florida State	Baseball	0	60	2003
28	4	Duke	Basketball	0	31	2002
29	4	Maryland	Basketball	0	32	2002
30	13	Wake Forest	Basketball	0	21	2002

Figure 310: Adding Record Filters Through Code



**Note:** Filter Expressions share the same format as in expression fields. For a list of valid expressions, refer [List of Filter Expressions](#).

## Nested Tables

Record Filters can also be set to the nested tables by accessing the RecordFilters collection of the ChildTableDescriptor.

### [C#]

```
FilterCondition cond = new
FilterCondition(FilterCompareOperator.GreaterThan, 20);
RecordFilterDescriptor filter = new RecordFilterDescriptor("OrderID",
cond);
this.gridGroupingControl1.GetTableDescriptor("Orders").RecordFilters.Add(filter);
```

### [VB.NET]

```
Dim cond As FilterCondition = New
FilterCondition(FilterCompareOperator.GreaterThan, 20)
Dim filter As RecordFilterDescriptor = New
```

```
RecordFilterDescriptor("OrderID", cond)
Me.gridGroupingControll1.GetTableDescriptor("Orders").RecordFilters.Add(
filter)
```

### Special Characters in Filter Values

To match the special characters like left bracket ([), question mark (?), number sign (#) and asterisk (\*), enclose them in square brackets (like [#] for # and [\*] for \* etc.,). The right bracket () can't be used within a group to match itself, but it can be used outside a group as an individual character.

This is illustrated in the below example with our Grid Grouping control.

[C#]

```
void Form1_Load(object sender, EventArgs e)
{
    ArrayList rank = new ArrayList();
    RankData rankData = new RankData("aaa");
    rank.Add(rankData);
    rankData = new RankData("bbb#");
    rank.Add(rankData);
    gridGroupingControll1.DataSource = rank;
    string filter = "";
    RecordFilterDescriptor rfd = null;
    Record r = null;

    foreach (RankData a in rank)
    {
        filter = "[WellName] like '" + ReplaceSpcChar(a.WellName) + "'";
        rfd = new RecordFilterDescriptor(filter);
        gridGroupingControll1.TableDescriptor.RecordFilters.Add(rfd);
        int cont = gridGroupingControll1.Table.FilteredRecords.Count;
        r = new Record(gridGroupingControll1.Table);

        // Exception will be thrown here if special characters are not
        // enclosed in square brackets.
        r = gridGroupingControll1.Table.FilteredRecords[0];
        rankData = r.GetData() as RankData;
        gridGroupingControll1.TableDescriptor.RecordFilters.Clear();
    }
}

private string ReplaceSpcChar(string pattern)
{
```

```
// Take caution while replacing the pattern and ensure that only the
// intended pattern is modified.
pattern = pattern.Replace("[", "[[]");
pattern = pattern.Replace("#", "[#]");
pattern = pattern.Replace("*", "[*]");
pattern = pattern.Replace("?", "[?]");
return pattern;
}
```

**[VB.NET]**

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs)
    Dim rank As New ArrayList()
    Dim rankData As New RankData("aaa")
    rank.Add(rankData)
    rankData = New RankData("bbb#")
    rank.Add(rankData)
    gridGroupingControll1.DataSource = rank
    Dim filter As String = ""
    Dim rfd As RecordFilterDescriptor = Nothing
    Dim r As Record = Nothing

    For Each a As RankData In rank
        filter = "[WellName] like '" & ReplaceSpcChar(a.WellName) & "'"
        rfd = New RecordFilterDescriptor(filter)
        gridGroupingControll1.TableDescriptor.RecordFilters.Add(rfd)
        Dim cont As Integer =
        gridGroupingControll1.Table.FilteredRecords.Count
        r = New Record(gridGroupingControll1.Table)

        ' Exception will be thrown here if special characters are not
        ' enclosed in square brackets.
        r = gridGroupingControll1.Table.FilteredRecords(0)
        rankData = TryCast(r.GetData(), RankData)
        gridGroupingControll1.TableDescriptor.RecordFilters.Clear()
    Next a
End Sub

Private Function ReplaceSpcChar(ByVal pattern As String) As String

    ' Take caution while replacing the pattern and ensure that only the
    ' intended pattern is modified.
    pattern = pattern.Replace("[", "[[]")
    pattern = pattern.Replace("#", "[#]")
    pattern = pattern.Replace("*", "[*]")
    pattern = pattern.Replace("?", "[?]")

```

```
    Return pattern  
End Function
```



**Note:** The 'Like' operator here is implemented similar to the 'Like' operator in VB.NET, where "#" character is considered as a character in patterns. Refer <http://msdn.microsoft.com/en-us/library/swf8kaxw.aspx> for detailed information.

### **Clearing Filters**

Row Filters that are added for a table can be cleared by calling the Clear() method of the RecordFilters property.

**[C#]**

```
this.gridGroupingControl1.TableDescriptor.RecordFilters.Clear();
```

**[VB .NET]**

```
Me.gridGroupingControl1.TableDescriptor.RecordFilters.Clear()
```



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Filters and Expressions\Filtering Tutorial

#### **4.3.4.3.4.2.1 Filter Bar**

Grouping Grid provides in-built support for displaying a **Filter Bar** across the columns. It can be used to filter and unfilter the records at run time. It is very user interactive and more advantageous than using the RecordFilters collection. The main reason for its wide usage is that it could display various filter options for the columns. You can be able to add your own filter criteria too.

When filter bar is applied, a new row (Filter Row) will be added at the top of the table displaying the filter options for the columns in a drop down. Each cell in the filter bar row is a simple ComboBox cell whose items are the filter options. The filter options for a given column includes one entry for each value in that column.

## Setting up a Filter Bar

The Filter Bar can be enabled by setting the ShowFilterBar and AllowFilter properties to true. The AllowFilter property can be set for the columns that require filter options.

Given below is the code to set the filter for all the columns in the main table.

### [C#]

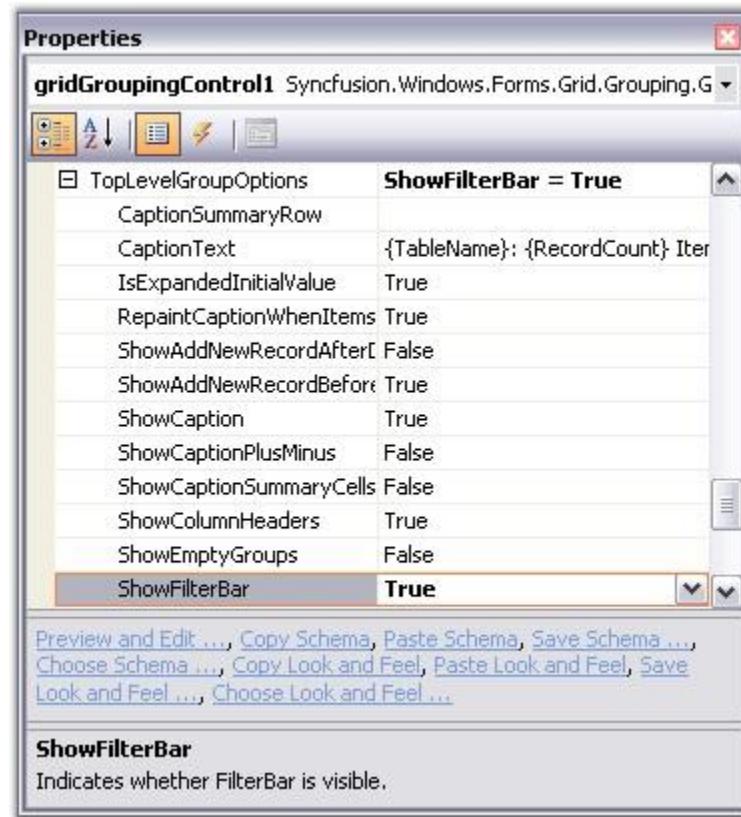
```
// Show Filter Bar for the main table.  
this.gridGroupingControl1.TopLevelGroupOptions.ShowFilterBar = true;  
  
// Change the appearance of the Filter Row.  
this.gridGroupingControl1.TableDescriptor.Appearance.FilterBarCell.BackColor = Color.CornSilk;  
  
// Enable filter for all columns.  
for (int i = 0; i < gridGroupingControl1.TableDescriptor.Columns.Count; i++)  
gridGroupingControl1.TableDescriptor.Columns[i].AllowFilter = true;
```

### [VB .NET]

```
' Show Filter Bar for the main table.  
Me.gridGroupingControl1.TopLevelGroupOptions.ShowFilterBar = True  
  
' Change the appearance of the Filter Row .  
  
Me.gridGroupingControl1.TableDescriptor.Appearance.FilterBarCell.BackColor = Color.AliceBlue  
  
' Enable the filter for all columns.  
Dim i As Integer = 0  
Do While i < gridGroupingControl1.TableDescriptor.Columns.Count  
gridGroupingControl1.TableDescriptor.Columns(i).AllowFilter = True  
i += 1  
Loop
```

## Through Designer

A filter bar can also be added at design time by setting the above properties through the property window of the grouping grid. The designer settings shown below adds the filter for the columns CompanyName and ContactTitle.



*Figure 311: Setting ShowFilterBar Property*

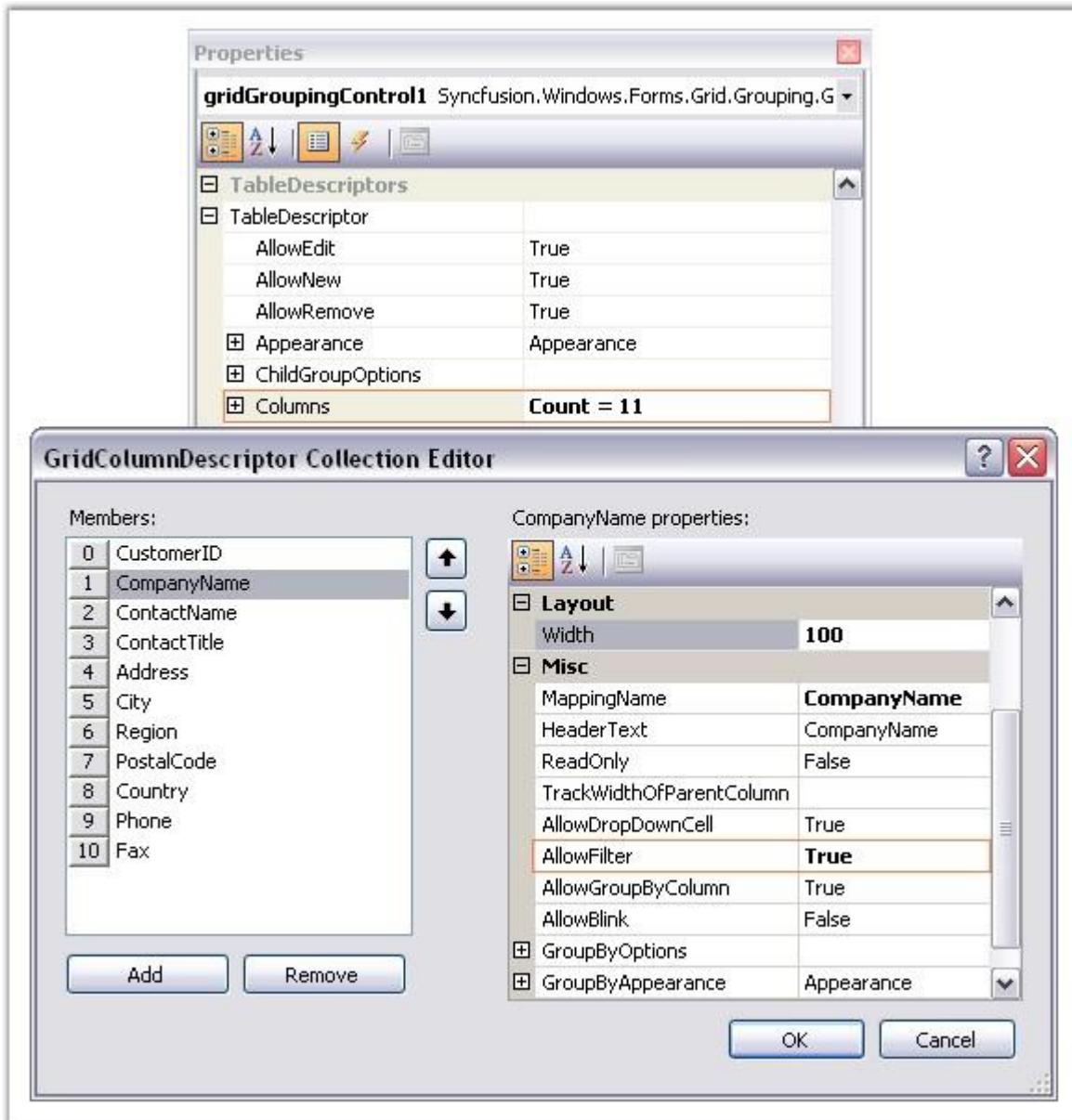


Figure 312: AllowFilter property enabled by using the GridColumnDescriptor Collection Editor

#### Setting AllowFilter property for the column CompanyName

Here is a sample screen shot that shows the filter bar with filters enabled for the columns CompanyName and ContactTitle. The records are filtered against the filter condition ContactTitle = 'Sales Representative'. The image also shows the filter options for the column CompanyName.

Customers: 17 Items

	CustomerID	CompanyName	ContactName	ContactTitle
		(All) (Custom...)		Sales Representative
	ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative
	AROUT	Around the Horn	Thomas Hardy	Sales Representative
	BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative
	BSBEV	B's Beverages	Victoria Ashworth	Sales Representative
	CONSH	Consolidated Holdings	Elizabeth Brown	Sales Representative
	FRANS	Die Wandernde Kuh	Paolo Accorti	Sales Representative
	HILAA	Franchi S.p.A.	Carlos Hernández	Sales Representative
	HUNGC	HILARIÓN-Abastos	Yoshi Latimer	Sales Representative
	LACOR	Hungry Coyote Import Store	Daniel Tonini	Sales Representative
	LEHMS	Lehmans Marktstand	Renate Messner	Sales Representative
	OLDWO	Old World Delicatessen	Rene Phillips	Sales Representative
	PFRNC	Pericles Comidas clásicas	Guillermo Fernández	Sales Representative

Figure 313: FilterBars added for "CompanyName" and "ContactTitle" Columns

While enabling the FilterBar, it automatically adds up an option for showing **All** records and a **Custom** option. Clicking the Custom option will open the RecordFilterDescriptor collection editor wherein you can edit the filter conditions or add any more filters you want. The following screen shot illustrates this process.

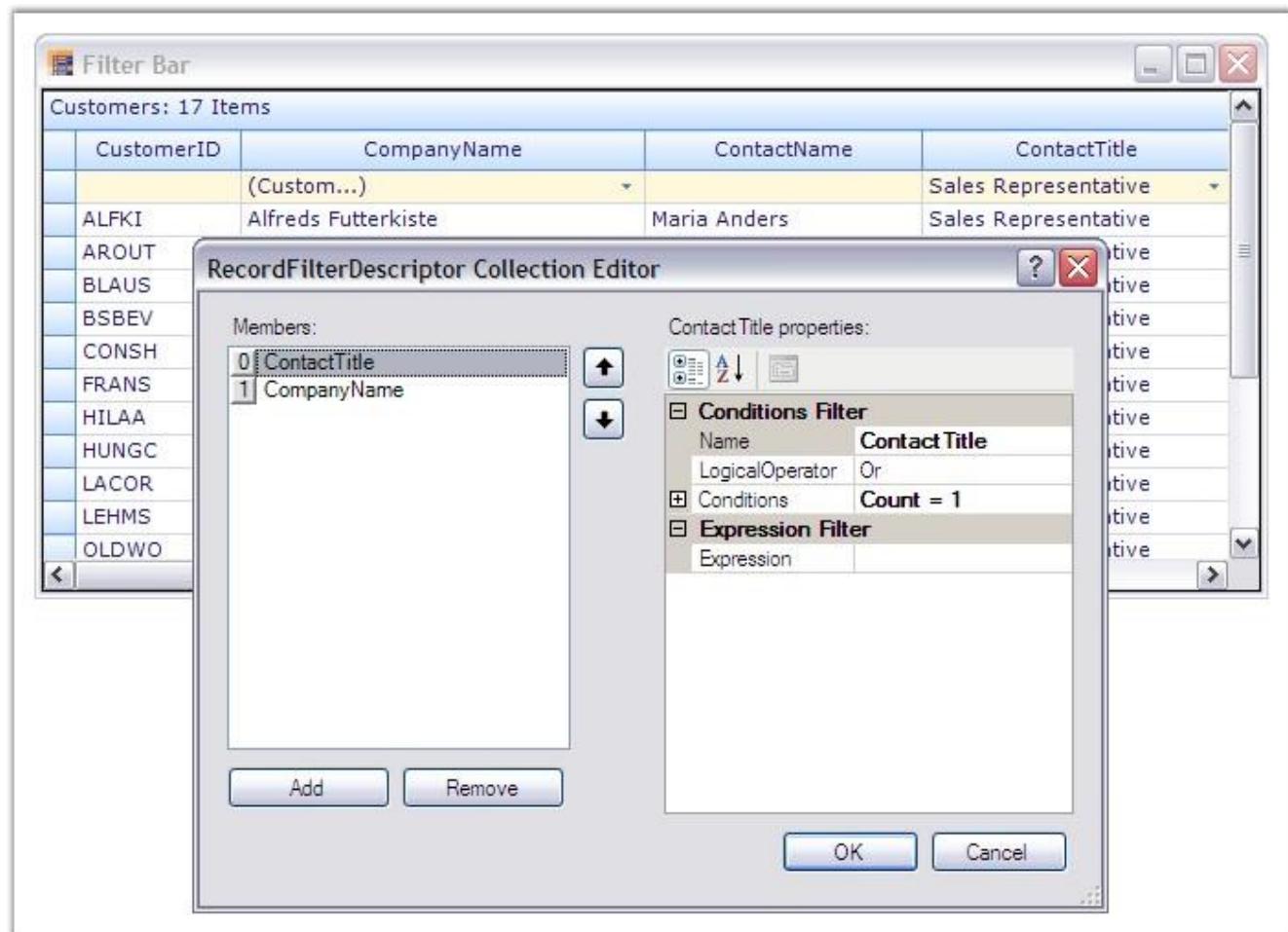


Figure 314: Editing Filter Conditions by using the RecordFilterDescriptor Collection Editor

### NestedTables and NestedGroups

AutoFilterRow can also be added to the nested tables and groups. To turn on the Filter Bar for the Nested Tables, set the property ShowFilterBar under NestedTableGroupOptions. For all the groups, ShowFilterBar under ChildGroupOptions need to be set to true.

[C#]

```
// Show Filter Bar for the child tables.
this.gridGroupingControl1.NestedTableGroupOptions.ShowFilterBar = true;

// Show Filter Bar for the groups.
this.gridGroupingControl1.ChildGroupOptions.ShowFilterBar = true;
```

[VB .NET]

```
' Show Filter Bar for the child tables.  
Me.gridGroupingControll1.NestedTableGroupOptions.ShowFilterBar = True  
  
' Show Filter Bar for the groups.  
Me.gridGroupingControll1.ChildGroupOptions.ShowFilterBar = True
```



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Filters and Expressions\Filter Bar Demo

#### 4.3.4.3.4.2.2 Dynamic Filter

Dynamic Filter serves as good replacement for the default Filter Bar, which provides advanced filtering capabilities. It is available as an add-on feature for Essential Grid, and is built on the foundation of the regular filter bar with added provisions to support dynamic filtering, user-friendliness, and so on.

The dynamic filter can be used with Nested Tables and Nested Groups too. To make it more interactive, it adds a Filter Button at the right-most corner of every Filter Bar Cell, on clicking, which drops down into a list showing the available comparative operators. On hovering any filter bar cell, a filter icon is displayed, indicating whether a filter is applied to that particular column or not. The key feature of the dynamic filtering mechanism is that it allows you to view the filter results as you type each and every character. It supports user-defined filter criteria as well.

#### Set up Dynamic Filter

The dynamic filter is defined in the GridDynamicFilter class, which exposes two public methods, **WireGrid** and **UnwireGrid**, in order to hook up and unhook the dynamic filter with the desired grid.

[C#]

```
GridDynamicFilter f = new GridDynamicFilter();  
if (showDynamicFilter)  
{  
    f.WireGrid(gridGroupingControll1);  
}
```

```

else
f.UnWireGrid(gridGroupingControl1);

```

**[VB .NET]**

```

Dim f As GridDynamicFilter = New GridDynamicFilter()
If showDynamicFilter Then
    f.WireGrid(gridGroupingControl1)
Else
    f.UnWireGrid(gridGroupingControl1)
End If

```

### Sample Output

Below image illustrates a sample output.

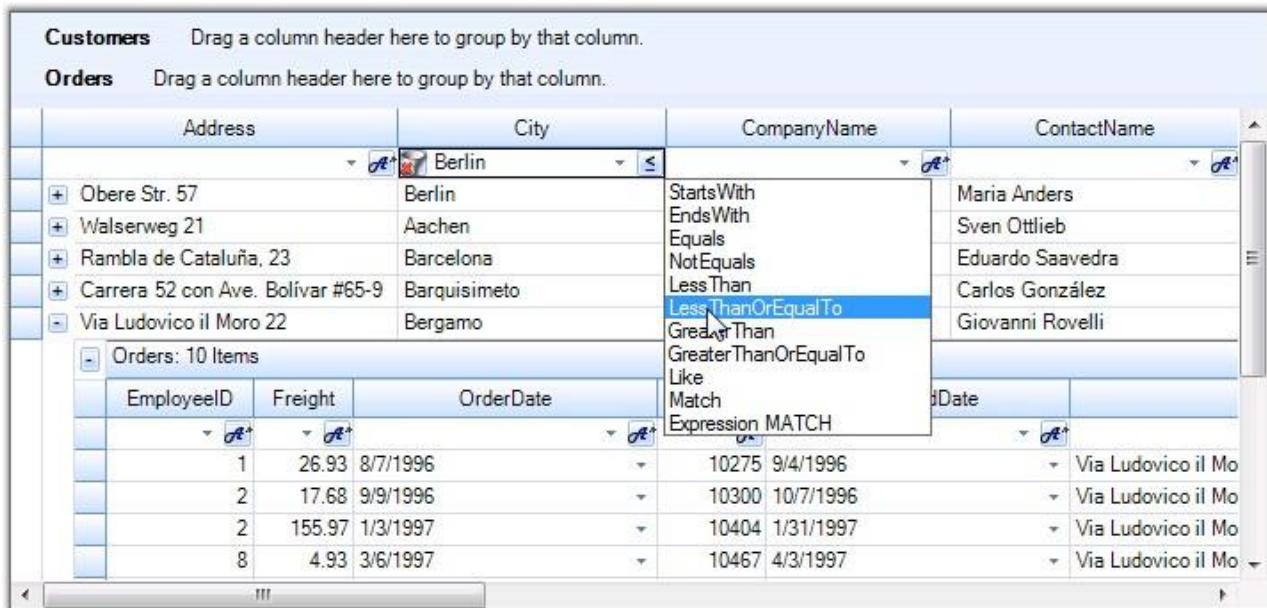


Figure 315: Dynamic Filter



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Filters and Expressions\Dynamic Filter Demo

#### 4.3.4.3.4.2.2.1 Localization Support for CompareOperatorListBox

The dynamic filter in the GridGrouping control provides support to customize the display content of the static element. Using this you can localize the static elements in the compare operator list box.

#### Use Case Scenarios

With this feature, you can localize the options in the compare operator list box to display the language specific to your locale.

#### Sample Link

A demo of this feature is available in the following location:

**{Installed Path}\Syncfusion\EssentialStudio\x.x.x\Windows\Grid.Grouping.Windows\Samples\2.0\Filters and Expressions\Dynamic Filter Demo**

#### Adding Localization Support for CompareOperatorListBox

To localize the content, create a class file and add an interface as *ILocalizationProvider*. Assign the required content to be displayed to the *DynamicFilterResourceIdentifiers* of the *GetLocalizedString* method as illustrated in the following code:

[C#]

```
public string GetLocalizedString(System.Globalization.CultureInfo culture,
string name)
{
    if (str == "True")
    {
        switch (name)
        {
            #region Menu Package
            case DynamicFilterResourceIdentifiers.StartsWith:
                return "empieza con";
            case DynamicFilterResourceIdentifiers.EndsWith:
                return "termina con";
            case DynamicFilterResourceIdentifiers.Equals:
                return "es igual a";
            case DynamicFilterResourceIdentifiers.GreaterThan:
                return "mayor que";
            case
DynamicFilterResourceIdentifiers.GreaterThanOrEqualTo:
                return "Mayor o igual a";
            case DynamicFilterResourceIdentifiers.LessThan:
                return "menos que";
            case DynamicFilterResourceIdentifiers.LessThanOrEqualTo:
                return "Menor o igual a";
        }
    }
}
```

```
        return "Menor o igual a";
    case DynamicFilterResourceIdentifiers.Like:
        return "como";
    case DynamicFilterResourceIdentifiers.Match:
        return "partido";
    case DynamicFilterResourceIdentifiers.NotEquals:
        return "no es igual";
    case DynamicFilterResourceIdentifiers.ExpressionMATCH:
        return "expresión de coincidencia";
#endregion

default:
    return string.Empty;
}
}
else
{
    switch (name)
    {
        #region Menu Package
        case DynamicFilterResourceIdentifiers.StartsWith:
            return "StartsWith";
        case DynamicFilterResourceIdentifiers.EndsWith:
            return "EndsWith";
        case DynamicFilterResourceIdentifiers.Equals:
            return "Equals";
        case DynamicFilterResourceIdentifiers.GreaterThan:
            return "GreaterThan";
        case
DynamicFilterResourceIdentifiers.GreaterThanOrEqualTo:
            return "GreaterThanOrEqualTo";
        case DynamicFilterResourceIdentifiers.LessThan:
            return "LessThan";
        case DynamicFilterResourceIdentifiers.LessThanOrEqualTo:
            return "LessThanOrEqualTo";
        case DynamicFilterResourceIdentifiers.Like:
            return "Like";
        case DynamicFilterResourceIdentifiers.Match:
            return "Match";
        case DynamicFilterResourceIdentifiers.NotEquals:
            return "NotEquals";
        case DynamicFilterResourceIdentifiers.ExpressionMATCH:
            return "ExpressionMATCH";
#endregion

default:
    return string.Empty;
}
}
}
```

**[VB]**

```

Public Function GetLocalizedString(ByVal culture As
System.Globalization.CultureInfo, ByVal name As String) As String
    If str = "True" Then
        Select Case name
            ' #Region "Menu Package"
            Case
DynamicFilterResourceIdentifiers.StartsWith
                Return "empieza con"
            Case
DynamicFilterResourceIdentifiers.EndsWith
                Return "termina con"
            Case
DynamicFilterResourceIdentifiers.Equals
                Return "es igual a"
            Case
DynamicFilterResourceIdentifiers.GreaterThan
                Return "mayor que"
            Case
DynamicFilterResourceIdentifiers.GreaterThanOrEqualTo
                Return "Mayor o igual a"
            Case
DynamicFilterResourceIdentifiers.LessThan
                Return "menos que"
            Case
DynamicFilterResourceIdentifiers.LessThanOrEqualTo
                Return "Menor o igual a"
            Case
DynamicFilterResourceIdentifiers.Like
                Return "como"
            Case
DynamicFilterResourceIdentifiers.Match
                Return "partido"
            Case
DynamicFilterResourceIdentifiers.NotEquals
                Return "no es igual"
            Case
DynamicFilterResourceIdentifiers.ExpressionMATCH
                Return "expresión de coincidencia"
        '
        #End Region

        Case Else
            Return String.Empty
        End Select
    Else
        Select Case name
            ' #Region "Menu Package"

```

```
Case
DynamicFilterResourceIdentifiers.StartsWith
    Return "StartsWith"
Case
DynamicFilterResourceIdentifiers.EndsWith
    Return "EndsWith"
Case
DynamicFilterResourceIdentifiers.Equals
    Return "Equals"
Case
DynamicFilterResourceIdentifiers.GreaterThan
    Return "GreaterThan"
Case
DynamicFilterResourceIdentifiers.GreaterThanOrEqualTo
    Return "GreaterThanOrEqualTo"
Case
DynamicFilterResourceIdentifiers.LessThan
    Return "LessThan"
Case
DynamicFilterResourceIdentifiers.LessThanOrEqualTo
    Return "LessThanOrEqualTo"
Case
DynamicFilterResourceIdentifiers.Like
    Return "Like"
Case
DynamicFilterResourceIdentifiers.Match
    Return "Match"
Case
DynamicFilterResourceIdentifiers.NotEquals
    Return "NotEquals"
Case
DynamicFilterResourceIdentifiers.ExpressionMATCH
    Return "ExpressionMATCH"
' #End Region

Case Else
    Return String.Empty
End Select
End If
End Function
```

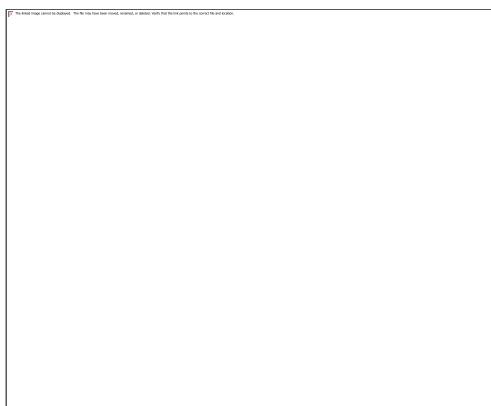


Figure 316: Localized CompareOperatorListBox

#### 4.3.4.3.4.2.3 Working with Filters

This section deals with different options to process the filter rows.

##### Accessing FilteredRecords

If you want to work with the subset of the records being filtered from a grid table, then you can use FilteredRecords collection for that table. This is a read only collection that manages the subset of records that has been filtered against a filter criteria.

###### [C#]

```
GridTable table = this.grid.Table;
foreach (Record freq in table.FilteredRecords)
{
    Console.WriteLine("Record Info : " + freq);
}
```

###### [VB.NET]

```
Private table As GridTable = Me.grid.Table
For Each freq As Record In table.FilteredRecords
    Console.WriteLine("Record Info : " & freq)
Next freq
```

##### Accessing the FilterBarString

To get access to the FilteredString, you can use the GetFilterBarText of the FilterBarCellRenderer. Following code example illustrates how to print the filter bar string for a given column.

[C#]

```
private void GetFilterBarString()
{
    int row = 0, col=0;
    string colName = null;
    GridTableCellStyleInfo style;

    // Ensure the filter bar is visible and RecordFilters collection is
    // not empty,
    // and get the filter bar row index and index of the field, by
    // using the value with which
    // the grid records are filtered.
    if(gridGroupingControl1.TableDescriptor.RecordFilters.Count > 0)
        colName =
gridGroupingControl1.TableDescriptor.RecordFilters[0].MappingName;

    foreach (Element el in
this.gridGroupingControl1.Table.DisplayElements)
    {
        if (el.IsFilterBar() && colName != null)
        {
            style = gridGroupingControl1.Table.GetTableCellStyle(el,
colName);
            row = style.TableCellIdentity.RowIndex;
            col = style.TableCellIdentity.ColIndex;
        }
    }

    // By using the calculated row and column indices, get the filter
    // bar string of the record filter.
    GridTableFilterBarCellRenderer cr =
this.gridGroupingControl1.TableControl.CellRenderers["FilterBarCell"]
as GridTableFilterBarCellRenderer;
    if (cr != null && row != 0)
    {

Console.WriteLine(cr.GetFilterBarText(this.gridGroupingControl1.TableMo
del[row, col]));
    }
}
```

[VB .NET]

```

Private Sub GetFilterBarString()
    Dim row As Integer = 0, col As Integer = 0
    Dim colName As String = Nothing
    Dim style As GridTableCellStyleInfo

    ' Ensure the filter bar is visible and RecordFilters collection is
    not empty,
    ' and get the filter bar row index and index of the field, by using
    the value with which
    ' the grid records are filtered.
    If gridGroupingControl1.TableDescriptor.RecordFilters.Count > 0 Then
        colName =
            gridGroupingControl1.TableDescriptor.RecordFilters(0).MappingName
    End If

    For Each el As Element In
        Me.gridGroupingControl1.Table.DisplayElements
        If el.IsFilterBar() AndAlso Not colName Is Nothing Then
            style = gridGroupingControl1.Table.GetTableCellStyle(el,
                colName)
            row = style.TableCellIdentity.RowIndex
            col = style.TableCellIdentity.ColIndex
        End If
    Next el

    ' By using the calculated row and column indices, get the filter bar
    string of the record filter.
    Dim cr As GridTableFilterBarCellRenderer =
        TryCast(Me.gridGroupingControl1.TableControl.CellRenderers("FilterBa
        rCell"), GridTableFilterBarCellRenderer)
    If Not cr Is Nothing AndAlso row <> 0 Then
        Console.WriteLine(cr.GetFilterBarText(Me.gridGroupingControl1.Tab
        leModel(row, col)))
    End If
End Sub

```

#### 4.3.4.3.4.2.4 Filter By DisplayMember

Grid Grouping control filters the data records by the value member of the columns by default. This behavior can be customized to get the filters work with display member of the columns. This is accomplished in the Filter By DisplayMember feature.

#### Implementation

The implementation includes the following classes.

- **GroupingGridFilterBarExt** - It derives a custom filter bar cell type named FilterByDisplayMemberCell and use this cell type in the place of default filter bar cell.
- **GridFilterByDisplayMemberCellModel** - This is the cell model class that loads the filter drop down with the display strings of the respective filter bar column and creates cell renderer.
- **GridFilterByDisplayMemberCellRenderer** - This is the cell renderer class that sets up the actual filter string by replacing the display string with the value string.

Following code example illustrates how to enable/disable this custom filter.

**[C#]**

```
// Enable filter.  
GroupingGridFilterBarExt gGCFilter = new GroupingGridFilterBarExt();  
gGCFilter.WireGrid(gridGroupingControll);  
  
// Disable filter.  
gGCFilter.UnwireGrid(gridGroupingControll);
```

**[VB .NET]**

```
' Enable filter.  
Dim gGCFilter As GroupingGridFilterBarExt = New  
GroupingGridFilterBarExt()  
gGCFilter.WireGrid(gridGroupingControll)  
  
' Disable filter.  
gGCFilter.UnwireGrid(gridGroupingControll)
```



Figure 317: Filtering By Display Member



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Filters and Expressions\Filter By DisplayMember Demo

#### 4.3.4.3.4.2.5 List of Filter Expressions

Filter expressions can be set and added to RecordFilters collection through a textual string. The string should abide by the syntax to be followed and should be valid.

The following table lists tokens used, (Expression Filters) and their descriptions.

Syntax	Expression	Description
*	[columnname] * 'anynumber'	Filters grid based on the multiplied value computed.

/	[columnname] / 'anynumber'	Filters grid based on the divided value computed.
+	[columnname] + 'anynumber'	Filters grid based on the result computed.
-	[columnname] - 'anynumber'	Filters grid based on computed value.
<	[columnname] < 'anynumber'	Filters grid displaying records whose specified column holds value lesser than the mentioned value.
>	[columnname] > 'anynumber'	Filters grid displaying records whose specified column holds value greater than the mentioned value.
=	[columnname] = 'value'	Filters grid displaying records whose specified column holds value equal to the mentioned value.
<=	[columnname] <= 'anynumber'	Filters grid displaying records whose specified column holds value lesser than or equal to the mentioned value.
>=	[columnname] >= 'anynumber'	Filters grid displaying records whose specified column holds value greater than or equal to the mentioned value.
<>	[columnname] <> 'value'	Filters grid displaying records whose specified column holds value not equal to the mentioned value.
AND	[expression1] AND [expression2] AND[expression]	Filters grid displaying records that meet criteria of all the expressions.
OR	[expression1] OR [expression2] OR[expression]	Filters grid displaying records that meet criteria of either or all the expressions.
XOR	[expression1] XOR [expression2] XOR [expression]	Filters grid displaying records that doesn't meet criteria of either or all the expressions.
LIKE	[columnname] LIKE 'value'	Filters grid displaying records whose specified column holds value equal to the mentioned value(irrespective of

		character casing).
MATCH	[columnname] MATCH 'value'	Filters grid displaying records whose specified column holds whole or a part of the mentioned value(irrespective of character casing).
BETWEEN	[columnname] BETWEEN {date1,date2}	Filters grid displaying records whose date lies between the two dates irrespective of the time.
BETWEENTIME	[columnname] BETWEENTIME {datetime1,datetime2}	Filters grid displaying records whose datetimevalue lies between the two dates and respective times.
IN	[columnname] IN 'val1,val2,..,valn'	Filters grid displaying records whose specified column hold the values mentioned.

#### 4.3.4.3.5 Relations and Hierarchy

Grid Grouping control can display nested tables in a **hierarchy** using a master-detail configuration. In an hierarchical view, all the tables in the data source are inter-connected via relations. Generally a relation between any two tables can take any of the following forms: **1:1**, **1:n**, **n:1** or **n:n**.

A grouping grid can automatically detect the data relations in a dataset for display. By default, a Relation is created for each such relation found in dataset. Hence the data relations defined in a dataset are sufficient enough for the grid to form the relations. No additional code is required in this case.

With nested tables, each record in the parent table will have an associated set of records in the child table. Every record in the relation is provided with a +/- button called **RecordPlusMinus** that can be expanded(as well as collapsed) to bring the underlying records in the child table into view. The number of tables that can be nested with relations using a Grid Grouping control is unlimited.

#### Relations Collection

The **TableDescriptor.Relations** collection defines relations for the table. By default a relation (**RelatedMasterDetails**) is created for each DataRelation found in a DataSet. Relations can either be related foreign key tables or nested child tables that can be expanded and collapsed. Each entry in this collection is owned by a RelationDescriptor that stores the details of a relation. All the RelationDescriptors for a given table is managed by the RelationDescriptor Collection which is returned by the TableDescriptor.Relations property.

The **TableDescriptor.RelationChildColumns** collection is internally initialized and contains the child key fields of the RelationDescriptor.RelationKeys collection of a RelationKind.RelatedMasterDetails relation. You should not modify this collection.

The **TableDescriptor.PrimaryKeyColumns** collection defines fields that form a unique primary key for the table. By default, the PrimaryKeyColumns collection is initialized from the child key fields of the RelationDescriptor.RelationKeys collection of a RelationKind.ForeignKeyReference relation. If the table is not a foreign table and a UniqueConstraint for a DataTable is present the collection is initialized with fields from that UniqueConstraint. Users can also manually modify the collection. If the table is the foreign table of a RelationKind.ForeignKeyReference relation, the parent table uses the fields that are defined in the PrimaryKeyColumns collection to lookup and identify records in the foreign table.

### **Setting Up Relations Through Designer**

After binding an hierarchical dataset to the grouping grid, you could find the TableDescriptor.Relations collection populated with values. These values represent the relationship between the parent and child tables.

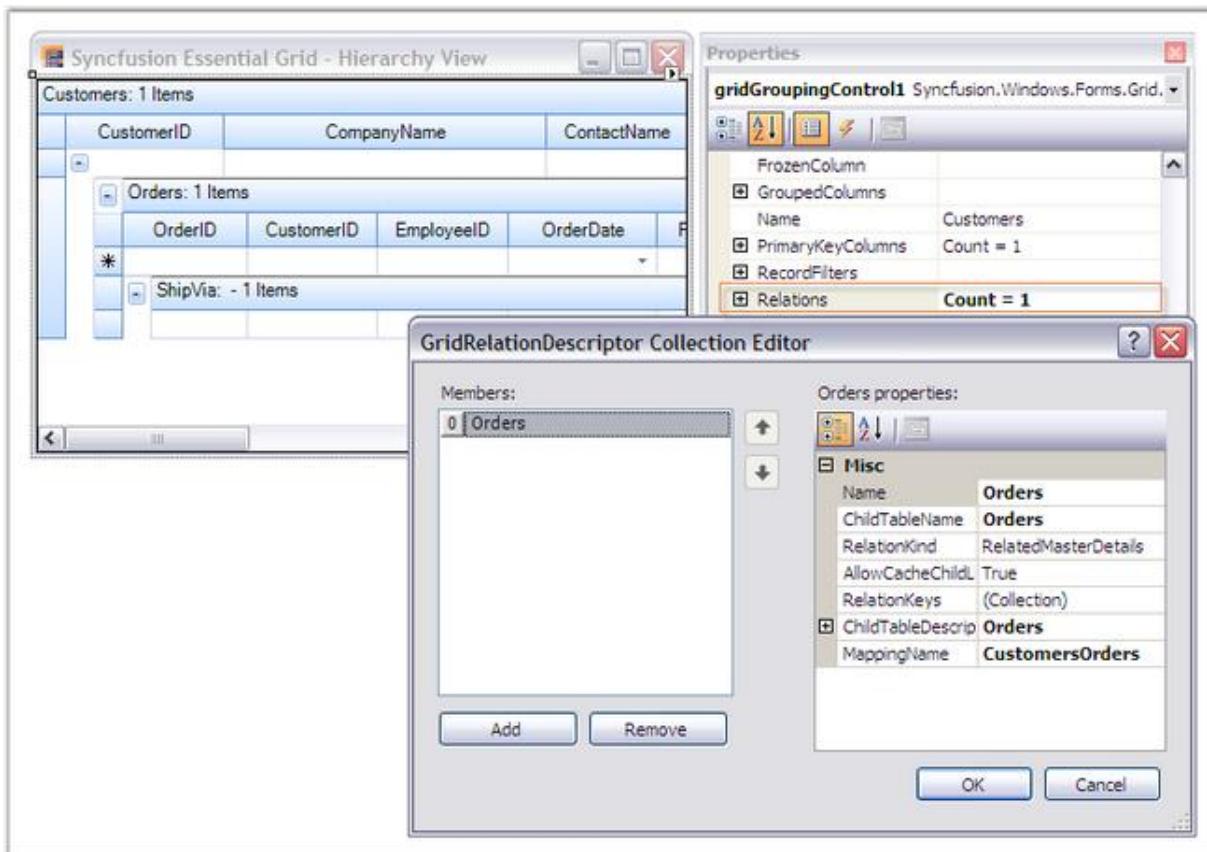


Figure 318: Setting Up Relations by using the GridRelationDescriptor Collection Editor

## Properties

Here is a brief description on the properties used to setup a relation.

GridRelationDescriptor Property	Description
Name	Specifies the relation name.
ChildTableName	Specifies the name of the ChildTable.
RelationKeys	Defines the mapping between the parent and child columns in a master-detail relation.
MappingName	Specifies the name of the PropertyDescriptor in the parent table that contains the details about the relation.

<b>RelationKind</b>	<p>Specifies the type of the relation.</p> <p>The options included are as follows.</p> <ul style="list-style-type: none"> <li>• <a href="#">RelatedMasterDetails</a></li> <li>• <a href="#">ForeignKeyReference</a></li> <li>• <a href="#">ForeignKey KeyWords</a></li> <li>• <a href="#">UniformChildList</a></li> <li>• <a href="#">ListItemReference</a></li> </ul>
<b>AllowCacheChildList</b>	<ul style="list-style-type: none"> <li>• Indicates whether the ChildList associated with a view can be cached.</li> <li>• Used with UniformChildList relation.</li> </ul>
<b>ChildTableDescriptor</b>	<p>Specifies the table schema of Child Table.</p>

### Different RelationKinds

Apart from the default Master-Detail type, Essential Grouping Grid supports a number of relations which could be enabled by specifying the relations manually in the grouping engine. In case of manual relations, the dataset does not need to have relations. This is the same approach that should be used if you want to setup relationships between independent IList collections.

### Supported Relations

- [RelatedMasterDetails](#)
- [ForeignKeyReference](#)
- [ForeignKey KeyWords](#)
- [ListItemReference](#)
- [UniformChildList](#)

### The ForeignKey DropDown

When a **ForeignKey** relation (a relation for looking up values into a related child table using a key) is used, the child records are displayed using a DropDownList. Using this foreign key drop down, you could be able to edit the record. Here are the screen shots of the foreign key drop down.

ForeignKey DropDown showing Edit Button (button with Pencil Icon) clicking which allows you to edit the list

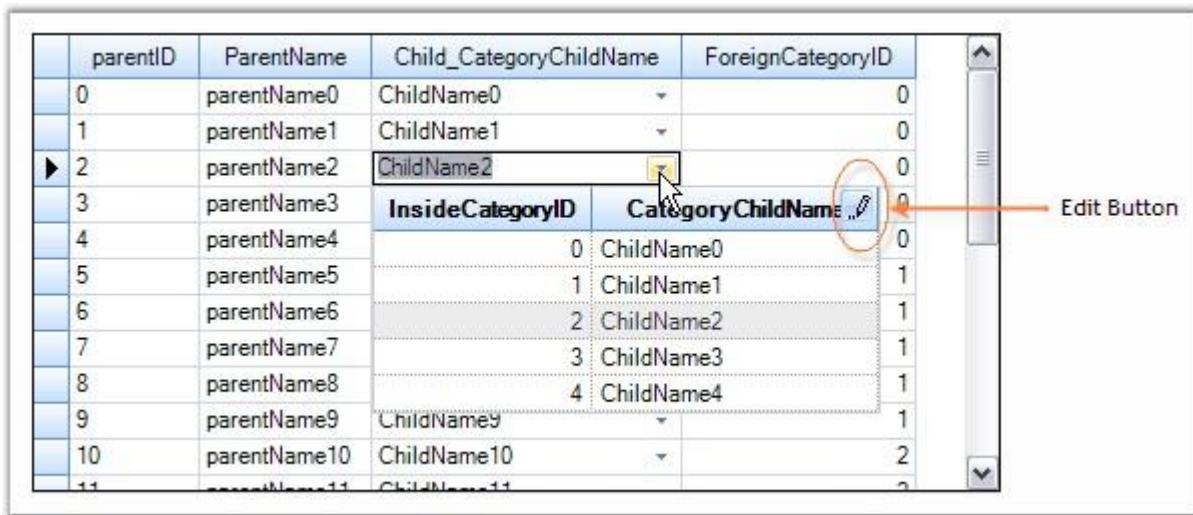


Figure 319: Edit button in the ForeignKey DropDownList

Below image shows the state after clicking the Edit button.

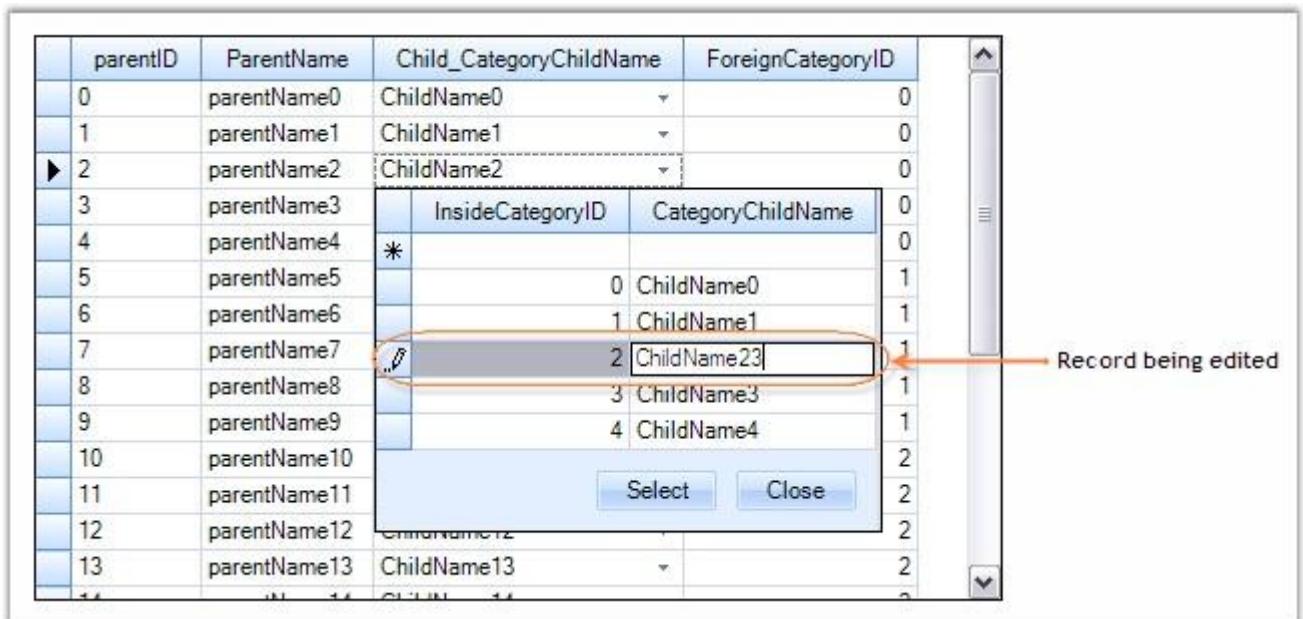


Figure 320: Editing Records by clicking the Edit Button in the ForeignKey DropDownList

#### 4.3.4.3.5.1 Related Master Details Relation

**RelatedMasterDetails** is a Master-Details relation where matching keys in columns in the parent and child tables, define a relationship between two tables. This a 1:n relation where each record in the child table can only belong to one parent record.

This section demonstrates how to manually specify the master-detail relations between three independent tables that have the primary key and foreign key column in common.

#### Steps to setup RelatedMasterDetails relation

1. Setup three datatables that have primary and foreign key columns in common.

[C#]

```
private int numberParentRows = 5;
private int numberChildRows = 20;
private int numberGrandChildRows = 50;

// Create Parent Table.
private DataTable GetParentTable()
{
    DataTable dt = new DataTable("ParentTable");

    dt.Columns.Add(new DataColumn("parentID"));
    dt.Columns.Add(new DataColumn("ParentName"));
    dt.Columns.Add(new DataColumn("ParentDec"));

    for(int i = 0; i < numberParentRows; i++)
    {
        DataRow dr = dt.NewRow();

        dr[0] = i;
        dr[1] = string.Format("parentName{0}", i);
        dr[2] = string.Format("parentName{0}", i);
        dt.Rows.Add(dr);
    }
    return dt;
}

// Create Child Table.
private DataTable GetChildTable()
{
    DataTable dt = new DataTable("ChildTable");
}
```

```
dt.Columns.Add(new DataColumn("childID"));
dt.Columns.Add(new DataColumn("Name"));
dt.Columns.Add(new DataColumn("ParentID"));

for(int i = 0; i < numberChildRows; i++)
{
    DataRow dr = dt.NewRow();
    dr[0] = i.ToString();
    dr[1] = string.Format("ChildName{0}",i);
    dr[2] = (i % numberParentRows).ToString();
    dt.Rows.Add(dr);
}
return dt;
}

// Create Grand Child Table.
private DataTable GetGrandChildTable()
{
    DataTable dt = new DataTable("GrandChildTable");

    dt.Columns.Add(new DataColumn("GrandChildID"));
    dt.Columns.Add(new DataColumn("Name"));
    dt.Columns.Add(new DataColumn("ChildID"));

    for(int i = 0; i < numberGrandChildRows; i++)
    {
        DataRow dr = dt.NewRow();
        dr[0] = i.ToString();
        dr[1] = string.Format("GrandChildName{0}",i);
        dr[2] = (i % numberChildRows).ToString();
        dt.Rows.Add(dr);
    }
    return dt;
}
```

**[VB.NET]**

```
Private numberParentRows As Integer = 5
Private numberChildRows As Integer = 20
Private numberGrandChildRows As Integer = 50

' Create Parent Table.
Private Function GetParentTable() As DataTable
    Dim dt As New DataTable("ParentTable")
```

```
dt.Columns.Add(New DataColumn("parentID"))
dt.Columns.Add(New DataColumn("ParentName"))
dt.Columns.Add(New DataColumn("ParentDec"))

Dim i As Integer
For i = 0 To numberParentRows - 1
    Dim dr As DataRow = dt.NewRow()
    dr(0) = i
    dr(1) = String.Format("parentName{0}", i)
    dr(1) = String.Format("parentName{0}", i)
    dt.Rows.Add(dr)
Next i

    Return dt
End Function

' Create Child Table.
Private Function GetChildTable() As DataTable
    Dim dt As New DataTable("ChildTable")

    dt.Columns.Add(New DataColumn("childID"))
    dt.Columns.Add(New DataColumn("Name"))
    dt.Columns.Add(New DataColumn("ParentID"))
    Dim i As Integer
    For i = 0 To numberChildRows - 1
        Dim dr As DataRow = dt.NewRow()
        dr(0) = i.ToString()
        dr(1) = String.Format("ChildName{0}", i)
        dr(2) = (i Mod numberParentRows).ToString()
        dt.Rows.Add(dr)
    Next i

    Return dt
End Function

' Create Grand Child Table.
Private Function GetGrandChildTable() As DataTable
    Dim dt As New DataTable("GrandChildTable")

    dt.Columns.Add(New DataColumn("GrandChildID"))
    dt.Columns.Add(New DataColumn("Name"))
    dt.Columns.Add(New DataColumn("ChildID"))
    Dim i As Integer
    For i = 0 To numberGrandChildRows - 1
        Dim dr As DataRow = dt.NewRow()
        dr(0) = i.ToString()
```

```
    dr(1) = String.Format("GrandChildName{0}", i)
    dr(2) = (i Mod numberChildRows).ToString()
    dt.Rows.Add(dr)
Next i

Return dt
End Function
```

2. Manually set up relationships between the tables and add the relation to the parent and child tables.

**[C#]**

```
GridRelationDescriptor parentToChildRelationDescriptor = new
GridRelationDescriptor();

// Same as SourceListSetEntry.Name for Child Table.
parentToChildRelationDescriptor.ChildTableName = "MyChildTable";
parentToChildRelationDescriptor.RelationKind =
RelationKind.RelatedMasterDetails;
parentToChildRelationDescriptor.RelationKeys.Add("parentID",
"ParentID");

// Add relation to Parent Table.
gridGroupingControl1.TableDescriptor.Relations.Add(parentToChildRelationDescriptor);

GridRelationDescriptor childToGrandChildRelationDescriptor = new
GridRelationDescriptor();

// Same as SourceListSetEntry.Name for Grand Child Table.
childToGrandChildRelationDescriptor.ChildTableName =
"MyGrandChildTable";
childToGrandChildRelationDescriptor.RelationKind =
RelationKind.RelatedMasterDetails;
childToGrandChildRelationDescriptor.RelationKeys.Add("childID",
"ChildID");

// Add relation to Child Table.
parentToChildRelationDescriptor.ChildTableDescriptor.Relations.Add(childToGrandChildRelationDescriptor);
```

**[VB .NET]**

```
Dim parentToChildRelationDescriptor As New GridRelationDescriptor()

' Same as SourceListSetEntry.Name for Child Table.
```

```
parentToChildRelationDescriptor.ChildTableName = "MyChildTable"
parentToChildRelationDescriptor.RelationKind =
RelationKind.RelatedMasterDetails
parentToChildRelationDescriptor.RelationKeys.Add("parentID",
"ParentID")

' Add relation to Parent Table.
gridGroupingControl1.TableDescriptor.Relations.Add(parentToChildRelationDescriptor)

Dim childToGrandChildRelationDescriptor As New GridRelationDescriptor()

' Same as SourceListSetEntry.Name for Grand Child Table.
childToGrandChildRelationDescriptor.ChildTableName =
"MyGrandChildTable"
childToGrandChildRelationDescriptor.RelationKind =
RelationKind.RelatedMasterDetails
childToGrandChildRelationDescriptor.RelationKeys.Add("childID",
"ChildID")

' Add relation to Child Table.
parentToChildRelationDescriptor.ChildTableDescriptor.Relations.Add(childToGrandChildRelationDescriptor)
```

3. Register the datatables with Engine.SourceListSet so that the RelationDescriptor can resolve the name.

**[C#]**

```
this.gridGroupingControl1.Engine.SourceListSet.Add("MyParentTable",
parentTable);
this.gridGroupingControl1.Engine.SourceListSet.Add("MyChildTable",
childTable);
this.gridGroupingControl1.Engine.SourceListSet.Add("MyGrandChildTable",
grandChildTable);
```

**[VB.NET]**

```
Me.gridGroupingControl1.Engine.SourceListSet.Add("MyParentTable",
parentTable)
Me.gridGroupingControl1.Engine.SourceListSet.Add("MyChildTable",
ChildTable)
Me.gridGroupingControl1.Engine.SourceListSet.Add("MyGrandChildTable",
grandChildTable)
```

4. Finally bind the hierarchical data source, which has been created through the above steps, to a grouping grid by assigning the parent table to the datasource.

[C#]

```
this.gridGroupingControl1.DataSource = parentTable;
```

[VB.NET]

```
Me.gridGroupingControl1.DataSource = parentTable
```

5. When you run the sample, you could find the tables connected with Master-Details relation.

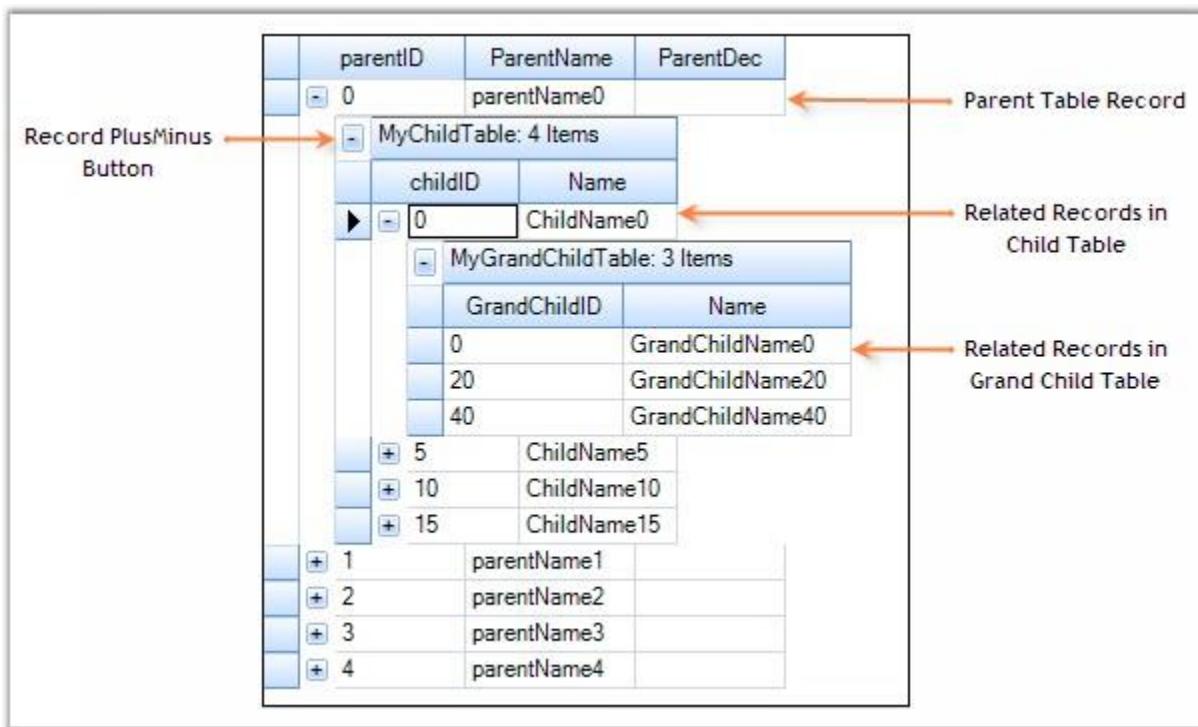


Figure 321: RelatedMasterDetails Relation



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\Version  
Number\Windows\Grid.Grouping.Windows\Samples\2.0\Relations And Hierarchy\Related Master  
Details Demo

#### 4.3.4.3.5.2 Foreign Key Reference Relation

**ForeignKeyReference** is a foreign-key relation for looking up values where an id column in the main table can be used to look up a record in a related table. This is an n:1 relation where multiple records in the parent table can reference the same record in the related table. Fields in the related table can be referenced using a '.' dot in the FieldDescriptor.MappingName of the main table.

#### Steps to setup ForeignKeyReference relation

This section sets up a foreignkeyreference relation between a data table and the collection **USStates**. The data table represents the Parent Table of the relation and the USStates collection serves as the related child list where in the values can be looked up using a key. The collection derives from ArrayList in which every item is an **USState** object having two properties named **Key** and **Name**. It also defines a method named **CreateDefaultCollection()** that returns an instance of itself populated with a set of values.

A foreignkeyreference relation can be set up between the lists by defining a relation descriptor with its attributes carrying the relation details and adding this descriptor to the Relations collection of the main table.

The following steps demonstrate this process.

1. Create a collection named **USStates** in which each entry stores a **USState** object.

[C#]

```
// US States Collection.
[Serializable]
public class USStatesCollection : ArrayList
{
    public new USState this[int index]
    {
        get
        {
            return (USState) base[index];
        }
        set
        {
            base[index] = value;
        }
    }

    public static USStatesCollection CreateDefaultCollection()
```

```
{  
    USStatesCollection states = new USStatesCollection();  
    states.Add(new USState("AL", "Alabama"));  
    states.Add(new USState("AK", "Alaska"));  
    states.Add(new USState("CA", "California"));  
    states.Add(new USState("FL", "Florida"));  
    states.Add(new USState("GA", "Georgia"));  
    states.Add(new USState("IN", "Indiana"));  
    states.Add(new USState("MS", "Mississippi"));  
    states.Add(new USState("NJ", "New Jersey"));  
    states.Add(new USState("NM", "New Mexico"));  
    states.Add(new USState("NY", "New York"));  
    states.Add(new USState("TX", "Texas"));  
    states.Add(new USState("WA", "Washington"));  
    states.Add(new USState("PE", "Prince Edward Island"));  
    states.Add(new USState("YT", "Yukon Territories"));  
    return states;  
}  
  
public override bool IsReadOnly  
{  
    get  
    {  
        return true;  
    }  
}  
  
public override bool IsFixedSize  
{  
    get  
    {  
        return true;  
    }  
}  
}  
  
}  
  
// US State Class.  
[Serializable]  
public class USState  
{  
    private string _code;  
    private string _name;  
  
    public USState()  
    {  
    }
```

```
}

public USState(string key, string name)
{
    this._code = key;
    this._name = name;
}

[Browsable(true)]
public string Key
{
    get
    {
        return _code;
    }
    set
    {
        _code = value;
    }
}

[Browsable(true)]
public string Name
{
    get
    {
        return _name ;
    }
    set
    {
        _name = value;
    }
}

public override string ToString()
{
    return this._name + "(" + this._code + ")";
}
```

**[VB.NET]**

```
' US States Collection.
<Serializable()> _
Public Class USStatesCollection
```

```
Inherits ArrayList

    Default Public Shadows Property Item(index As Integer) As USState
    Get
        Return CType(MyBase.Item(index), USState)
    End Get
    Set
        MyBase.Item(index) = Value
    End Set
End Property

Public Shared Function CreateDefaultCollection() As
USStatesCollection
    Dim states As New USStatesCollection()
    states.Add(New USState("AL", "Alabama"))
    states.Add(New USState("AK", "Alaska"))
    states.Add(New USState("CA", "California"))
    states.Add(New USState("FL", "Florida"))
    states.Add(New USState("GA", "Georgia"))
    states.Add(New USState("IN", "Indiana"))
    states.Add(New USState("MS", "Mississippi"))
    states.Add(New USState("NJ", "New Jersey"))
    states.Add(New USState("NM", "New Mexico"))
    states.Add(New USState("NY", "New York"))
    states.Add(New USState("TX", "Texas"))
    states.Add(New USState("WA", "Washington"))
    states.Add(New USState("PE", "Prince Edward Island"))
    states.Add(New USState("YT", "Yukon Territories"))
    Return states
End Function

Public Overrides ReadOnly Property IsReadOnly() As Boolean
Get
    Return True
End Get
End Property

Public Overrides ReadOnly Property IsFixedSize() As Boolean
Get
    Return True
End Get
End Property
End Class

' US State Class.
<Serializable()>
```

```
Public Class USState
    Private _code As String
    Private _name As String

    Public Sub New()
        End Sub

    Public Sub New(key As String, name As String)
        Me._code = key
        Me._name = name
        End Sub

    <Browsable(True)> _
    Public Property Key() As String
        Get
            Return _code
        End Get
        Set
            _code = value
        End Set
    End Property

    <Browsable(True)> _
    Public Property Name() As String
        Get
            Return _name
        End Get
        Set
            _name = value
        End Set
    End Property

    Public Overrides Function ToString() As String
        Return Me._name + "(" + Me._code + ")"
    End Function
End Class
```

2. Create an object of USStates and add this object into the SourceListSet with a lookup name.

[C#]

```
USStatesCollection usStates =
USStatesCollection.CreateDefaultCollection();
this.gridGroupingControl1.Engine.SourceListSet.Add("USStates",
usStates);
```

**[VB.NET]**

```
Dim usStates As USStatesCollection =  
    USStatesCollection.CreateDefaultCollection()  
Me.gridGroupingControll1.Engine.SourceListSet.Add("USStates", usStates)
```

3. Creates a datatable with the **Key** from **USState** as one of the columns.

**[C#]**

```
DataTable table = new DataTable();  
table.Columns.Add("Id", typeof(string));  
table.Columns.Add("State", typeof(string));  
  
// Adding rows.  
for (int i = 0; i < 25; i++)  
{  
    table.Rows.Add(table.NewRow());  
    table.Rows[i][0] = i;  
    table.Rows[i][1] = usStates[i % 8].Key;  
}
```

**[VB.NET]**

```
Dim table As New DataTable()  
table.Columns.Add("Id", GetType(String))  
table.Columns.Add("State", GetType(String))  
  
' Adding rows.  
Dim i As Integer  
For i = 0 To 24  
    table.Rows.Add(table.NewRow())  
    table.Rows(i)(0) = i  
    table.Rows(i)(1) = usStates((i Mod 8)).Key  
Next i
```

4. Establish the ForeignKeyReference relationship.

**[C#]**

```
GridTableDescriptor mainTd = this.gridGroupingControll1.TableDescriptor;  
  
GridRelationDescriptor usStatesRd = new GridRelationDescriptor();  
usStatesRd.Name = "State";
```

```
usStatesRd.RelationKind = RelationKind.ForeignKeyReference;

// SourceListSet name for look up.
usStatesRd.ChildTableName = "USStates";
usStatesRd.RelationKeys.Add("State", "Key");

// Format ChildList.
usStatesRd.ChildTableDescriptor.Appearance.AlternateRecordFieldCell.BackColor = Color.FromArgb(255, 245, 227);

// To hide the Key column.
usStatesRd.ChildTableDescriptor.VisibleColumns.Add("Name");
usStatesRd.ChildTableDescriptor.SortedColumns.Add("Name");
usStatesRd.ChildTableDescriptor.AllowEdit = false;

// Disallow users to modify states.
usStatesRd.ChildTableDescriptor.AllowNew = false;

mainTd.Relations.Add(usStatesRd);

// Assign data source.
this.gridGroupingControll1.DataSource = table;
mainTd.Name = "ForeignKeyReference";
```

**[VB.NET]**

```
Dim mainTd As GridTableDescriptor =
Me.gridGroupingControll1.TableDescriptor

Dim usStatesRd As New GridRelationDescriptor()
usStatesRd.Name = "State"
usStatesRd.RelationKind = RelationKind.ForeignKeyReference

' SourceListSet name for look up.
usStatesRd.ChildTableName = "USStates"
usStatesRd.RelationKeys.Add("State", "Key")

' Format ChildList.
usStatesRd.ChildTableDescriptor.Appearance.AlternateRecordFieldCell.BackColor = Color.FromArgb(255, 245, 227)

' To hide the Key Column.
usStatesRd.ChildTableDescriptor.VisibleColumns.Add("Name")
usStatesRd.ChildTableDescriptor.SortedColumns.Add("Name")
usStatesRd.ChildTableDescriptor.AllowEdit = False
```

```
' Disallow users to modify states.  
usStatesRd.ChildTableDescriptor.AllowNew = False  
  
mainTd.Relations.Add(usStatesRd)  
  
' Assign data source.  
Me.gridGroupingControll1.DataSource = table  
mainTd.Name = "ForeignKeyReference"
```

5. Here is a sample output that displays a look up child list for the data column **State** with value **Georgia**.



Figure 322: ForeignKeyRelation Relation

 **Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Relations And Hierarchy\Foreign-Key Reference Demo

### GridForeignKeyHelper

GridForeignKeyHelper is a helper class that makes it easy for the users to use foreign key relations to do foreign key look ups. With this class available, the users can easily hook up a foreign table by a single method call instead of going through all the steps described above.

The GridForeignKeyHelper class exposes a static method called SetupForeignTableLookUp that accepts grouping grid, main table, foreign table, main table column, foreign table value column and foreign table display column and sets up the Foreign Key relation using these parameter values.

#### [C#]

```
string valueColInMainTable = "Country", valueColInForeignTable =
"CountryCode", displayColInForeignTable = "CountryName";
GridForeignKeyHelper.SetupForeignTableLookUp(gridGroupingControl1,
valueColInMainTable, countries, valueColInForeignTable,
displayColInForeignTable);
```

#### [VB .NET]

```
Dim valueColInMainTable As String = "Country", valueColInForeignTable
As String = "CountryCode",
displayColInForeignTable As String = "CountryName"
GridForeignKeyHelper.SetupForeignTableLookUp(gridGroupingControl1,
valueColInMainTable, countries, valueColInForeignTable,
displayColInForeignTable)
```

#### 4.3.4.3.5.3 Foreign Key KeyWords Relation

**ForeignKeyKeyWords** is a unique relation kind which, is offered by the grouping engine. It is a foreign key relation where matching keys in the columns of the parent and child table define a relationship between the two tables. This is an m:n relation. Field summaries of the related child table can be referenced using a '.' dot in the FieldDescriptor.MappingName of the main table. This relation kind allows you to have multi-valued columns in the grid.

## Example

Say you have a Customers table. Each customer can have a list of purchased items. With MasterDetails, for a given customer, the underlying child list (the list of items purchased by that customer) will be displayed in a separate table once the RecordPlusMinus button is clicked. Instead if you want to view the entire record along with the related child records in a single row, then ForeignKeyKeyWords would be the right choice to use.

The following example illustrates creation of ForeignKeyKeyWords relation.

1. Create two data tables Customers and Items and add a list of records into them.

[C#]

```
private int numberParentRows = 6;
private int numberChildRows = 20;

private DataTable GetParentTable()
{
    DataTable dt = new DataTable("Customers");

    dt.Columns.Add(new DataColumn("customerID"));
    dt.Columns.Add(new DataColumn("CustomerName"));
    dt.Columns.Add(new DataColumn("Address"));

    for(int i = 0; i < numberParentRows; ++i)
    {
        DataRow dr = dt.NewRow();
        dr[0] = i;
        dr[1] = string.Format("CustomerName{0}", i);
        dr[2] = string.Format("Address{0}", i);
        dt.Rows.Add(dr);
    }

    return dt;
}

private DataTable GetChildTable()
{
    DataTable dt = new DataTable("Items");

    dt.Columns.Add(new DataColumn("ItemID"));
    dt.Columns.Add(new DataColumn("ItemName"));
```

```
dt.Columns.Add(new DataColumn("CustomerID"));
dt.Columns.Add(new DataColumn("Price"));
Random rand = new Random();
for(int i = 0; i < numberChildRows; ++i)
{
    DataRow dr = dt.NewRow();
    dr[0] = i.ToString();
    dr[1] = string.Format("ItemName{0}",i);
    dr[2] = (i % numberParentRows).ToString();
    dr[3] = rand.Next(500).ToString();
    dt.Rows.Add(dr);
}

return dt;
}
```

**[VB.NET]**

```
Private numberParentRows As Integer = 6
Private numberChildRows As Integer = 20

Private Function GetParentTable() As DataTable
    Dim dt As DataTable = New DataTable("Customers")

    dt.Columns.Add(New DataColumn("customerID"))
    dt.Columns.Add(New DataColumn("CustomerName"))
    dt.Columns.Add(New DataColumn("Address"))

    Dim i As Integer = 0

    Do While i < numberParentRows
        Dim dr As DataRow = dt.NewRow()
        dr(0) = i
        dr(1) = String.Format("CustomerName{0}", i)
        dr(2) = String.Format("Address{0}", i)
        dt.Rows.Add(dr)
        i += 1
    Loop
    Return dt
End Function

Private Function GetChildTable() As DataTable
    Dim dt As DataTable = New DataTable("Items")

    dt.Columns.Add(New DataColumn("ItemID"))
```

```
dt.Columns.Add(New DataColumn("ItemName"))
dt.Columns.Add(New DataColumn("CustomerID"))
dt.Columns.Add(New DataColumn("Price"))
Dim rand As Random = New Random()
Dim i As Integer = 0
Do While i < numberChildRows
    Dim dr As DataRow = dt.NewRow()
    dr(0) = i.ToString()
    dr(1) = String.Format("ItemName{0}", i)
    dr(2) = (i Mod numberParentRows).ToString()
    dr(3) = rand.Next(500).ToString()
    dt.Rows.Add(dr)
    i += 1
Loop
Return dt
End Function
```

2. Register the child table (Items) into the SourceListSet of the grouping engine.

**[C#]**

```
DataTable parentTable = GetParentTable();
DataTable childTable = GetChildTable();

this.gridGroupingControl1.Engine.SourceListSet.Add("Items",
childTable);
```

**[VB .NET]**

```
Dim parentTable As DataTable = GetParentTable()
Dim childTable As DataTable = GetChildTable()

Me.gridGroupingControl1.Engine.SourceListSet.Add("Items", childTable)
```

3. Assign the datasource to the grid.

**[C#]**

```
this.gridGroupingControl1.DataSource = parentTable;
```

**[VB .NET]**

```
Me.gridGroupingControl1.DataSource = parentTable
```

4. Establish ForeignKeyWords relationship between the tables.

**[C#]**

```
GridRelationDescriptor childRelation = new GridRelationDescriptor();
childRelation.RelationKind = RelationKind.ForeignKeyKeyWords;

// SourceListSet name for look up.
childRelation.ChildTableName = "Items";
childRelation.RelationKeys.Add("customerID", "CustomerID");
childRelation.ChildTableDescriptor.AllowEdit = true;
childRelation.ChildTableDescriptor.AllowNew = true;
this.gridGroupingControl1.TableDescriptor.Relations.Add(childRelation);
```

**[VB .NET]**

```
Dim childRelation As GridRelationDescriptor = New
GridRelationDescriptor()
childRelation.RelationKind = RelationKind.ForeignKeyKeyWords

' SourceListSet name for look up.
childRelation.ChildTableName = "Items"
childRelation.RelationKeys.Add("customerID", "CustomerID")
childRelation.ChildTableDescriptor.AllowEdit = True;
childRelation.ChildTableDescriptor.AllowNew = True;
Me.gridGroupingControl1.TableDescriptor.Relations.Add(childRelation)
```

5. Here is a sample output.

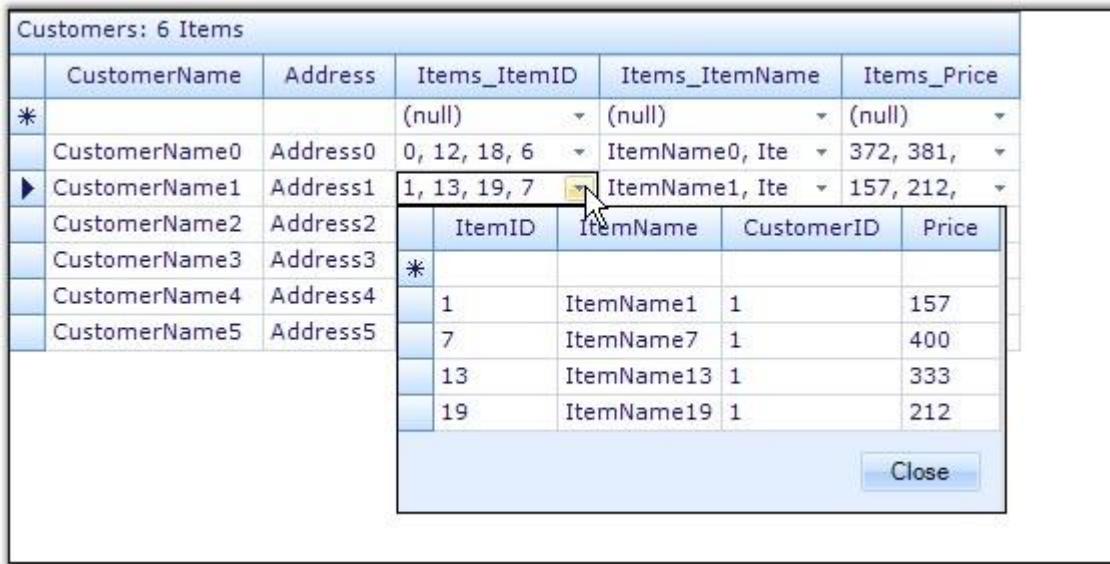


Figure 323: ForeignKeyKeyWords Relation



**Note:** For more details, refer the following browser sample:

```
<Install Location>\Syncfusion\EssentialStudio\Version
Number]\Windows\Grid.Grouping.Windows\Samples\2.0\Relations And Hierarchy\Employee Territory
Order Demo
```

#### 4.3.4.3.5.4 ListItem Reference Relation

**ListItemReference** is an object reference relation for looking up values from a strong typed collection. Like ForeignKeyReference, it is also an n:1 relation where multiple records in the parent table can reference the same record in the related table. One difference between the ForeignKeyReference and ListItemReference is that the former uses a key to look up the values whereas the later uses an object to look up the values in a nested collection.

#### Steps to setup ListItemReference relation

This section sets up a ListItemReference relation between a data table and the collection **Countries**. The data table represents the Parent Table of the relation and the Countries collection serves as the related child list where in the values can be looked up using an object of the child list. The collection derives from ArrayList in which every item is a **Country** object having two properties, **CountryCode** and **Name**. It also defines a method named **CreateDefaultCollection()** that returns an instance of itself populated with a set of values.

This relation kind can be set up by defining a relation descriptor with its attributes carrying the relation details and adding this descriptor to the Relations collection of the main table.

The following steps demonstrate this process.

1. Create a collection named **Countries** in which each entry stores a **Country** object.

```
[C#]

// Countries Collection.
[Serializable]
public class CountriesCollection : ArrayList
{
    public new Country this[int index]
    {
```

```
get
{
    return (Country) base[index];
}
set
{
    base[index] = value;
}
}

public static CountriesCollection CreateDefaultCollection()
{
    CountriesCollection countries = new CountriesCollection();
    countries.Add(new Country("US", "United States"));
    countries.Add(new Country("CA", "Canada"));
    countries.Add(new Country("AU", "Australia"));
    countries.Add(new Country("BR", "Brazil"));
    countries.Add(new Country("IO", "British Indian Ocean
Territory"));
    countries.Add(new Country("CN", "China"));
    countries.Add(new Country("FI", "Finland"));
    countries.Add(new Country("FR", "France"));
    countries.Add(new Country("DE", "Germany"));
    countries.Add(new Country("HK", "Hong Kong"));
    countries.Add(new Country("HU", "Hungary"));
    countries.Add(new Country("IS", "Iceland"));
    countries.Add(new Country("IN", "India"));
    countries.Add(new Country("JP", "Japan"));
    countries.Add(new Country("MY", "Malaysia"));
    countries.Add(new Country("SG", "Singapore"));
    countries.Add(new Country("CH", "Switzerland"));
    return countries;
}

public override bool IsReadOnly
{
    get
    {
        return true;
    }
}

public override bool IsFixedSize
{
    get
    {
```

```
        return true;
    }
}

// Country Class.
[Serializable]
public class Country
{
    private string _code;
    private string _name;

    public Country()
    {

    }

    public Country(string strCode, string strName)
    {
        this._code = strCode;
        this._name = strName;
    }

    [Browsable(true)]
    public string CountryCode
    {
        get
        {
            return _code;
        }
        set
        {
            _code = value;
        }
    }

    [Browsable(true)]
    public string Name
    {
        get
        {
            return _name ;
        }
        set
        {
            _name = value;
        }
    }
}
```

```
        }

    }

    public override string ToString()
    {
        return this._name + "(" + this._code + ")";
    }
}
```

**[VB.NET]**

```
'Countries Collection
<Serializable()> _
Public Class CountriesCollection
Inherits ArrayList

    Default Public Shadows Property Item(index As Integer) As Country
    Get
        Return CType(MyBase.Item(index), Country)
    End Get
    Set
        MyBase.Item(index) = Value
    End Set
    End Property

    Public Shared Function CreateDefaultCollection() As CountriesCollection
        Dim countries As New CountriesCollection()
        countries.Add(New Country("US", "United States"))
        countries.Add(New Country("CA", "Canada"))
        countries.Add(New Country("AU", "Australia"))
        countries.Add(New Country("BR", "Brazil"))
        countries.Add(New Country("IO", "British Indian Ocean Territory"))
        countries.Add(New Country("CN", "China"))
        countries.Add(New Country("FI", "Finland"))
        countries.Add(New Country("FR", "France"))
        countries.Add(New Country("DE", "Germany"))
        countries.Add(New Country("HK", "Hong Kong"))
        countries.Add(New Country("HU", "Hungary"))
        countries.Add(New Country("IS", "Iceland"))
        countries.Add(New Country("IN", "India"))
        countries.Add(New Country("JP", "Japan"))
        countries.Add(New Country("MY", "Malaysia"))
        countries.Add(New Country("SG", "Singapore"))
        countries.Add(New Country("CH", "Switzerland"))
    End Function
}
```

```
    Return countries
End Function

Public Overrides ReadOnly Property IsReadOnly() As Boolean
Get
    Return True
End Get
End Property

Public Overrides ReadOnly Property IsFixedSize() As Boolean
Get
    Return True
End Get
End Property
End Class

' Country Class.
<Serializable()>
Public Class Country
    Private _code As String
    Private _name As String

    Public Sub New()
    End Sub

    Public Sub New(strCode As String, strName As String)
        Me._code = strCode
        Me._name = strName
    End Sub

    <Browsable(True)>
    Public Property CountryCode() As String
        Get
            Return _code
        End Get
        Set
            _code = value
        End Set
    End Property

    <Browsable(True)>
    Public Property Name() As String
        Get
            Return _name
        End Get
    End Property

```

```
Set
    _name = value
End Set
End Property

Public Overrides Function ToString() As String
    Return Me._name + "(" + Me._code + ")"
End Function
End Class
```

2. Create an object of USStates and add this object into the SourceListSet with a lookup name.

**[C#]**

```
CountriesCollection countries =
CountriesCollection.CreateDefaultCollection();
this.gridGroupingControll.Engine.SourceListSet.Add("Countries",
countries);
```

**[VB .NET]**

```
Dim countries As CountriesCollection =
CountriesCollection.CreateDefaultCollection()
Me.gridGroupingControll.Engine.SourceListSet.Add("Countries",
countries)
```

3. Create a datatable with one of the columns is of type **Country**.

**[C#]**

```
DataTable table = new DataTable();
table.Columns.Add("Id", typeof(string));
table.Columns.Add("Country", typeof(Country));

// Adding Rows.
```

```
for (int i = 0; i < 25; i++)
{
    table.Rows.Add(table.NewRow());
    table.Rows[i][0] = i;
    table.Rows[i][1] = countries[i % 8];
}
```

**[VB.NET]**

```
Dim table As New DataTable()
table.Columns.Add("Id", GetType(String))
table.Columns.Add("Country", GetType(Country))

' Adding Rows.
Dim i As Integer
For i = 0 To 24
    table.Rows.Add(table.NewRow())
    table.Rows(i)(0) = i
    table.Rows(i)(1) = countries((i Mod 8))
Next i
```

4. Establish the ForeignKeyReference relationship.

**[C#]**

```
GridTableDescriptor mainTd = this.gridGroupingControl1.TableDescriptor;

GridRelationDescriptor countriesRd = new GridRelationDescriptor();
countriesRd.Name = "Country";
countriesRd.MappingName = "Country";
countriesRd.RelationKind = RelationKind.ListItemReference;

// SourceListSet name for look up.
countriesRd.ChildTableName = "Countries";

// Format ChildList.
countriesRd.ChildTableDescriptor.Appearance.AlternateRecordFieldCell.Ba
ckColor = Color.FromArgb(255, 245, 227);

// To hide the Key column.
countriesRd.ChildTableDescriptor.VisibleColumns.Add("Name");
countriesRd.ChildTableDescriptor.SortedColumns.Add("Name");
countriesRd.ChildTableDescriptor.AllowEdit = true;

// Disallow users to modify states.
```

```
countriesRd.ChildTableDescriptor.AllowNew = true;

mainTd.Relations.Add(countriesRd);

// Assign data source.
this.gridGroupingControll.DataSource = table;
mainTd.Name = "ListItemReference";
```

**[VB.NET]**

```
Dim mainTd As GridTableDescriptor =
Me.gridGroupingControll.TableDescriptor

Dim countriesRd As New GridRelationDescriptor()
countriesRd.Name = "Country"
countriesRd.MappingName = "Country"
countriesRd.RelationKind = RelationKind.ListItemReference

' SourceListSet name for look up.
countriesRd.ChildTableName = "Countries"

' Format ChildList.
countriesRd.ChildTableDescriptor.Appearance.AlternateRecordFieldCell.Ba-
ckColor = Color.FromArgb(255, 245, 227)

' To hide the Key Column.
countriesRd.ChildTableDescriptor.VisibleColumns.Add("Name")
countriesRd.ChildTableDescriptor.SortedColumns.Add("Name")
countriesRd.ChildTableDescriptor.AllowEdit = True

' Disallow users to modify states.
countriesRd.ChildTableDescriptor.AllowNew = True

mainTd.Relations.Add(countriesRd)

' Assign data source.
Me.gridGroupingControll.DataSource = Table
mainTd.Name = "ListItemReference"
```

5. Here is a sample output that displays a look up child list for the data column **Country** with value **Brazil**.

The screenshot shows a Windows Form application window titled 'Essential Grid for Windows Forms'. Inside, there is a large grid control and a smaller grid control embedded within it. The main grid has columns for 'Id', 'Country\_CountryCode', and 'Country\_Name'. A row for 'BR' is selected, and a dropdown menu is open over it, displaying a detailed view of country codes and names. The dropdown grid has columns for 'CountryCode' and 'Name'. The data shown in the dropdown is as follows:

CountryCode	Name
BR	Brazil
IO	British Indian Ocean Territory
CN	China
FI	Finland
FR	France
US	Germany
CA	Hong Kong
AU	Hungary
BR	Iceland
IO	India
CN	Japan
FI	Malaysia
FR	Singapore
US	Switzerland
CA	
AU	
BR	
IO	

Figure 324: ListItemReference Relation



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Relations And Hierarchy\List Item Reference Demo

#### 4.3.4.3.5.5 Uniform Child List Relation

**UniformChildList** relation can be used to map nested strong typed collection inside a parent collection. If a public property is an object, then it will be displayed in a Nested Table. The collection in the below example consists of two kinds of objects, **ParentObj** and **ChildObj**, where every ParentObj is associated with a collection of ChildObjs and it is represented by the public property named '**Child**'. Hence a nested table is always created to display the associated children for a given parent.

#### Example

1. Create a class(ChildObj) whose instances form the child table records.

[C#]

```
public class ChildObj : INotifyPropertyChanged
{
    private string f1, f2;
    private int f3;

    public ChildObj(string f1, string f2, int f3) {
        this.f1 = f1;
        this.f2 = f2;
        this.f3 = f3;
    }

    public string Field1
    {
        get { return f1; }
        set
        {
            if (f1 != value)
            {
                f1 = value;
                RaisePropertyChanged("Field1");
            }
        }
    }

    public string Field2
    {
        get { return f2; }
        set
        {
            if (f2 != value)
            {
                f2 = value;
                RaisePropertyChanged("Field2");
            }
        }
    }

    public int Field3
    {
        get { return f3; }
        set
```

```
{  
    if (f3 != value)  
    {  
        f3 = value;  
        RaisePropertyChanged("Field3");  
    }  
}  
  
void RaisePropertyChanged(string name)  
{  
    if (PropertyChanged != null)  
        PropertyChanged(this, new PropertyChangedEventArgs(name));  
}  
  
public event PropertyChangedEventHandler PropertyChanged;  
}
```

**[VB.NET]**

```
Public Class ChildObj : Implements INotifyPropertyChanged

    Private f1, f2 As String
    Private f3 As Integer

    Public Sub New(ByVal f1 As String, ByVal f2 As String, ByVal f3 As Integer)
        Me.f1 = f1
        Me.f2 = f2
        Me.f3 = f3
    End Sub

    Public Property Field1() As String
        Get
            Return f1
        End Get
        Set(ByVal value As String)
            If f1 <> value Then
                f1 = value
                RaisePropertyChanged("Field1")
            End If
        End Set
    End Property

    Public Property Field2() As String
        Get
            Return f2
        End Get
        Set(ByVal value As String)
            If f2 <> value Then
                f2 = value
                RaisePropertyChanged("Field2")
            End If
        End Set
    End Property

    Public Property Field3() As Integer
        Get
            Return f3
        End Get
        Set(ByVal value As Integer)
            If f3 <> value Then
                f3 = value
                RaisePropertyChanged("Field3")
            End If
        End Set
    End Property

```

```
f3 = value
RaisePropertyChanged("Field3")
End If
End Set
End Property

Sub RaisePropertyChanged(ByVal name As String)
    RaiseEvent PropertyChanged(Me, New
PropertyChangedEventArgs(name))
End Sub

Public Event PropertyChanged As PropertyChangedEventHandler
Implements INotifyPropertyChanged.PropertyChanged

End Class
```

2. Create another class(ParentObj) that contains a reference to the above class (ChildObj). The instances of this class make the parent records. Both the classes implement the INotifyPropertyChanged interface in order to get notified of any property changes.

[C#]

```
public class ParentObj : INotifyPropertyChanged
{
    private string f1, f2;
    private int f3;
    private BindingList<ChildObj> childObj = new
BindingList<ChildObj>();

    public ParentObj(string f1, string f2, int f3, params ChildObj[] c)
    {
        this.f1 = f1;
        this.f2 = f2;
        this.f3 = f3;
        foreach(ChildObj i in c)
            childObj.Add(i);
    }

    public string Field1
    {
        get { return f1; }
        set
        {
            if (f1 != value)
            {
                f1 = value;
                RaisePropertyChanged("Field1");
            }
        }
    }

    public string Field2
    {
        get { return f2; }
        set
        {
            if (f2 != value)
            {
                f2 = value;
                RaisePropertyChanged("Field2");
            }
        }
    }
}
```

```
        }

    }

    public int Field3 {
        get { return f3; }
        set
        {
            if (f3 != value)
            {
                f3 = value;
                RaisePropertyChanged("Field3");
            }
        }
    }

    public BindingList<ChildObj> Child {
        get { return childObj; }
    }

    void RaisePropertyChanged(string name)
    {
        if (PropertyChanged != null)
            PropertyChanged(this, new PropertyChangedEventArgs(name));
    }

    public event PropertyChangedEventHandler PropertyChanged;
}
```

**[VB.NET]**

```
Public Class ParentObj : Implements INotifyPropertyChanged

    Private f1, f2 As String
    Private f3 As Integer
    Private childObj As BindingList(Of ChildObj) = New BindingList(Of
ChildObj)()

    Public Sub New(ByVal f1 As String, ByVal f2 As String, ByVal f3 As
Integer, ByVal ParamArray c As ChildObj())
        Me.f1 = f1
        Me.f2 = f2
        Me.f3 = f3
        For Each i As ChildObj In c
            childObj.Add(i)
        Next i
    End Sub
```

```
Public Property Field1() As String
    Get
        Return f1
    End Get
    Set(ByVal value As String)
        If f1 <> value Then
            f1 = value
            RaisePropertyChanged("Field1")
        End If
    End Set
End Property

Public Property Field2() As String
    Get
        Return f2
    End Get
    Set(ByVal value As String)
        If f2 <> value Then
            f2 = value
            RaisePropertyChanged("Field2")
        End If
    End Set
End Property

Public Property Field3() As Integer
    Get
        Return f3
    End Get
    Set(ByVal value As Integer)
        If f3 <> value Then
            f3 = value
            RaisePropertyChanged("Field3")
        End If
    End Set
End Property

Public ReadOnly Property Child() As BindingList(Of ChildObj)
    Get
        Return childObj
    End Get
End Property

Sub RaisePropertyChanged(ByVal name As String)
    RaiseEvent PropertyChanged(Me, New
PropertyChangedEventArgs(name))
```

```
End Sub

Public Event PropertyChanged As PropertyChangedEventHandler
Implements INotifyPropertyChanged.PropertyChanged
End Class
```

3. Generate the collection using BindingList class which implements the ListChanged events in itself so that the grid can listen to those events when the list is changed. Add few items into the collection.

**[C#]**

```
BindingList<ParentObj> topList = new BindingList<ParentObj>();
BindingList<ChildObj> childList = new BindingList<ChildObj>();

Random r = new Random();
for (int i = 0; i < 30; i++)
    childList.Add(new ChildObj(string.Format("Name{0}", r.Next(10)),
        string.Format("Desc{0}", r.Next(20)), r.Next(30)));

for (int i = 0; i < 5; i++)
{
    topList.Add(new ParentObj(string.Format("Name{0}", r.Next(5)),
        string.Format("Desc{0}", r.Next(15)), r.Next(20)));
    for (int j = i * 5; j < (i * 5) + 5; j++)
        topList[i].Child.Add(childList[j]);
}
```

**[VB .NET]**

```
Private topList As BindingList(Of UniformChildList_2005.ParentObj) =
New BindingList(Of UniformChildList_2005.ParentObj)()
Private childList As BindingList(Of UniformChildList_2005.ChildObj) =
New BindingList(Of UniformChildList_2005.ChildObj)()

Private r As Random = New Random()
For i As Integer = 0 To 29
    childList.Add(New
        UniformChildList_2005.ChildObj(String.Format("Name{0}", r.Next(10)),
            String.Format("Desc{0}", r.Next(20)), r.Next(30)))
Next i

For i As Integer = 0 To 4
    topList.Add(New
        UniformChildList_2005.ParentObj(String.Format("Name{0}", r.Next(5)),
            String.Format("Desc{0}", r.Next(15)), r.Next(20)))
Dim j As Integer = i * 5
```

```
Do While j < (i * 5) + 5
    topList(i).Child.Add(childList(j))
    j += 1
Loop
Next i
```

4. Assign the above collection to the datasource of grouping grid.

**[C#]**

```
gridGroupingControl1.DataSource = topList;
```

**[VB .NET]**

```
GridGroupingControl1.DataSource = topList
```

5. Establish UniformChildList relation kind.

**[C#]**

```
GridRelationDescriptor relation = new GridRelationDescriptor();
relation.RelationKind = RelationKind.UniformChildList;
relation.MappingName = "Child";
relation.Name = "Child";
relation.ChildTableName = "ChildTable";
gridGroupingControl1.TableDescriptor.Relations.Add(relation);

this.gridGroupingControl1.ShowGroupDropArea = true;
GridTable chiltTable = gridGroupingControl1.GetTable("ChildTable");
this.gridGroupingControl1.AddGroupDropArea(chiltTable);
chiltTable.TableDescriptor.GroupedColumns.Add("Field1");
```

**[VB .NET]**

```
Dim relation As GridRelationDescriptor = New GridRelationDescriptor()
relation.RelationKind = RelationKind.UniformChildList
relation.MappingName = "Child"
relation.Name = "Child"
relation.ChildTableName = "ChildTable"
gridGroupingControl1.TableDescriptor.Relations.Add(relation)

Me.gridGroupingControl1.ShowGroupDropArea = True
Dim chiltTable As GridTable =
gridGroupingControl1.GetTable("ChildTable")
Me.gridGroupingControl1.AddGroupDropArea(chiltTable)
```

```
chiltTable.TableDescriptor.GroupedColumns.Add("Field1")
```

6. Here is a sample output.

BindingList`1			Drag a column header here to group by that column.
Child	Field1		
	Field1	Field2	Field3
+	Name4	Desc10	10
+	Name4	Desc11	6
+	Name4	Desc2	17
+	Name1	Desc7	8
+	Name0	Desc4	6

Child: 5 Items			
	Field1	Field2	Field3
-	Field1: Name0 - 2 Items		
	Name0	Desc16	12
	Name0	Desc13	17
-	Field1: Name1 - 1 Items		
	Name1	Desc1	21
+	Field1: Name6 - 1 Items		
-	Field1: Name7 - 1 Items		
	Name7	Desc6	16

Figure 325: UniformChildList Relation



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Relations And Hierarchy\Uniform Child List Demo

#### 4.3.4.3.5.6 Working with Relations

This section discusses how to handle the related tables in certain scenarios. It also describes the properties and events used for this purpose.

## AutoPopulateRelations Property

It specifies if the relations should be automatically generated when you assign DataSource, with a DataTable with constraints or a DataSet with relations defined. It is true by default.

### [C#]

```
this.gridGroupingControl1.AutoPopulateRelations = false;
```

### [VB .NET]

```
Me.gridGroupingControl1.AutoPopulateRelations = False
```

## ShowRelationFields Property

When a foreign key relation (or related collection) is used, this property controls the display of the dependent fields from a related table in the main table.

### Possible Options

Property	Description
ShowRelationFields	Displays the dependent fields from a related table in the main table. It includes the following options. <ul style="list-style-type: none"><li>• <b>Hide</b>-Hides all the fields.</li><li>• <b>ShowAllRelatedFields</b>-Shows all related fields including Primary and Foreign Keys.</li><li>• <b>ShowDisplayFieldsOnly</b>-Shows only dependent fields; Hides Primary and Foreign Keys.</li></ul>

**Default value is ShowDisplayFieldsOnly.**

### [C#]

```
this.gridGroupingControl1.ShowRelationFields =
ShowRelationFields.ShowAllRelatedFields;
```

### [VB .NET]

```
Me.gridGroupingControl1.ShowRelationFields =  
ShowRelationFields.ShowAllRelatedFields;
```

### **MaxNestedCollectionRecursionLevel Property**

When nested collection is used, this property specifies the number of levels of recursion allowed when self-relations are detected. Below settings lets the grid to loop through up to four recursion levels.

**[C#]**

```
this.gridGroupingControl1.Engine.MaxNestedCollectionRecurseLevel = 4;
```

**[VB .NET]**

```
Me.gridGroupingControl1.Engine.MaxNestedCollectionRecurseLevel = 4
```

### **QueryShowRelationDisplayFields Event**

It is raised for every foreign-key relation. It allows you to control at run-time related fields of the child table should be added to the FieldDescriptorCollection. Inside this event, you can check for specific fields and set e.Cancel to true to avoid adding those fields.

**[C#]**

```
this.gridGroupingControl1.Engine.QueryShowRelationDisplayFields += new  
QueryShowRelationFieldsEventHandler(Engine_QueryShowRelationDisplayFields);  
  
void Engine_QueryShowRelationDisplayFields(object sender,  
QueryShowRelationFieldsEventArgs e)  
{  
    if (e.Relation.ChildTableName == "Countries")  
    {  
        e.ShowRelationFields = ShowRelationFields.Hide;  
    }  
}
```

**[VB .NET]**

```
AddHandler gridGroupingControl1.Engine.QueryShowRelationDisplayFields,  
AddressOf Engine_QueryShowRelationDisplayFields
```

```
Private Sub Engine_QueryShowRelationDisplayFields(ByVal sender As Object, ByVal e As QueryShowRelationFieldsEventArgs)
    If e.Relation.ChildTableName = "Countries" Then
        e.ShowRelationFields = ShowRelationFields.Hide
    End If
End Sub
```

### QueryShowField Event

It gets fired for every field in the Field Descriptor Collection of the individual tables in the dataset. It lets you control over the population of field descriptors. Using this event, you can check for specific fields and cancel the population of desired fields at runtime.

#### [C#]

```
this.gridGroupingControll.Engine.QueryShowField += new
QueryShowFieldEventHandler(Engine_QueryShowField);

void Engine_QueryShowField(object sender, QueryShowFieldEventArgs e)
{
    if (e.Field.Name == "GrandChildID")
        e.Cancel = true;
}
```

#### [VB .NET]

```
AddHandler gridGroupingControll.Engine.QueryShowField, AddressOf
Engine_QueryShowField

Private Sub Engine_QueryShowField(ByVal sender As Object, ByVal e As
QueryShowFieldEventArgs)
    If e.Field.Name = "GrandChildID" Then
        e.Cancel = True
    End If
End Sub
```

### QueryAddRelation Event

It is invoked for every relation that is being added to the RelationDescriptorCollection. By setting e.Cancel to true, you can avoid specific relations being added.

#### [C#]

```
this.gridGroupingControll.Engine.QueryAddRelation += new
```

```
QueryAddRelationEventHandler(Engine_QueryAddRelation);

void Engine_QueryAddRelation(object sender, QueryAddRelationEventArgs e)
{
    Console.WriteLine(e.Relation.Name);
}
```

**[VB .NET]**

```
AddHandler gridGroupingControl1.Engine.QueryAddRelation, AddressOf
Engine_QueryAddRelation

Private Sub Engine_QueryAddRelation(ByVal sender As Object, ByVal e As
QueryAddRelationEventArgs)
    Console.WriteLine(e.Relation.Name)
End Sub
```

### **QueryShowNestedPropertiesFields Event**

It is called when there exists nested properties in the bound datasource. With the help of this event, you can determine if the individual fields in the nested property should be displayed.

**[C#]**

```
this.gridGroupingControl1.Engine.QueryShowNestedPropertiesFields += new
QueryShowNestedPropertiesFieldsEventHandler(Engine_QueryShowNestedPropertiesFields);

void Engine_QueryShowNestedPropertiesFields(object sender,
QueryShowNestedPropertiesFieldsEventArgs e)
{
    if (e.PropertyDescriptor.PropertyType == typeof(BaseClass))
        e.Cancel = true;
}
```

**[VB .NET]**

```
AddHandler gridGroupingControl1.Engine.QueryShowNestedPropertiesFields,
AddressOf Engine_QueryShowNestedPropertiesFields

Private Sub Engine_QueryShowNestedPropertiesFields(ByVal sender As
Object, ByVal e As QueryShowNestedPropertiesFieldsEventArgs)
    If e.PropertyDescriptor.PropertyType Is GetType(BaseClass) Then
        e.Cancel = True

```

```
End If  
End Sub
```

#### 4.3.4.4 Paging Support in GridGrouping Control

Paging support has been provided in Windows Grid to improve the performance of the GridGrouping control. This feature enables you to load data in an efficient way by storing and retrieving records in Pages.

Each page consisting of not more than hundred records, thereby decreasing the load time; previously all the records were displayed on a single page. Filters have also been customized to perform filtering only on the hundred items displayed anytime on the current page.

Navigation through the retrieved records is accomplished by using the Arrow buttons of the newly rendered Record Navigation control, displayed at the bottom of the page.

GridGrouping control also provides Paging support for various Data Source Items. This is achieved by using a Pager which extracts a specific page from the bound data source, and binds it to the GridGrouping control.

Certain restrictions may occur on Filtering while using the Paging functionality. This is because a grid can control only the records in the page bound to it, and not all the records in the data source. However, this restriction can be overcome by wiring the view to a temporary table.

The following code illustrates wiring the GridGrouping control to a Pager:

[C#]

```
var pager = new Pager {PageSize = 1000};  
pager.Wire(gridGroupingControl1, dt); // dt is a DataTable object
```

[VB]

```
Dim pager = New Pager With {PageSize = 1000}  
pager.Wire(gridGroupingControl1, dt) ' dt is a DataTable object
```

The following code illustrates Filtering along with Paging functionality in the GridGrouping control:

[C#]

```
gridGroupingControl1.TopLevelGroupOptions.ShowFilterBar = true;  
foreach (var col in _gridGroupingControl2.TableDescriptor.Columns)  
    col.AllowFilter = true;
```

[VB]

```
gridGroupingControl1.TopLevelGroupOptions.ShowFilterBar = True  
For Each col In _gridGroupingControl2.TableDescriptor.Columns  
    col.AllowFilter = True  
Next col
```

Screen shot:

SNo	StudentID	Department	Year	Location
01	S01	MECHANICAL	III	TRICHUR
02	S02	MARINE	II	BILASPUR
03	S03	BIO-TECH	II	HUBLI
04	S04	AERO	II	TRICHUR
05	S05	EEE	I	CHITHOOR
06	S06	CIVIL	I	CUTTACK
07	S07	CIVIL	I	BILASPUR
08	S08	ECE	III	CHENNAI
09	S09	AERO	III	GWALIOR
10	S10	MARINE	II	CHITHOOR
11	S11	BIO-TECH	II	GWALIOR
12	S12	AERO	III	HUBLI
13	S13	EEE	III	HUBLI

Figure Figure 326: Paging Support in Windows Grid

The following sample illustrates Paging support in the GridGrouping control for a data table populated with 100,000 records.



**Note:** For more details, refer the following sample link:

<http://www.syncfusion.com/downloads/Support/DirectTrac/101296/Paging-1289568956.zip>

#### 4.3.4.4.1 Paging support with a different data source

Paging support has been provided in the Grid control for Windows Forms to improve the performance of the GridGroupingControl. This feature enables you to load data in an efficient way by storing and retrieving records in pages. Paging support is provided for different data sources. We have provided paging support with IEnumerable, ArrayList, Generic Collection, IBindingList, and DataTable.

The following code sample illustrates wiring the GridGrouping control to a Pager:

[C#]

```
var pager = new Pager {PageSize = 1000};  
pager.Wire(gridGroupingControl1); // Data sources assigned to the pager  
from gridGroupingControl
```

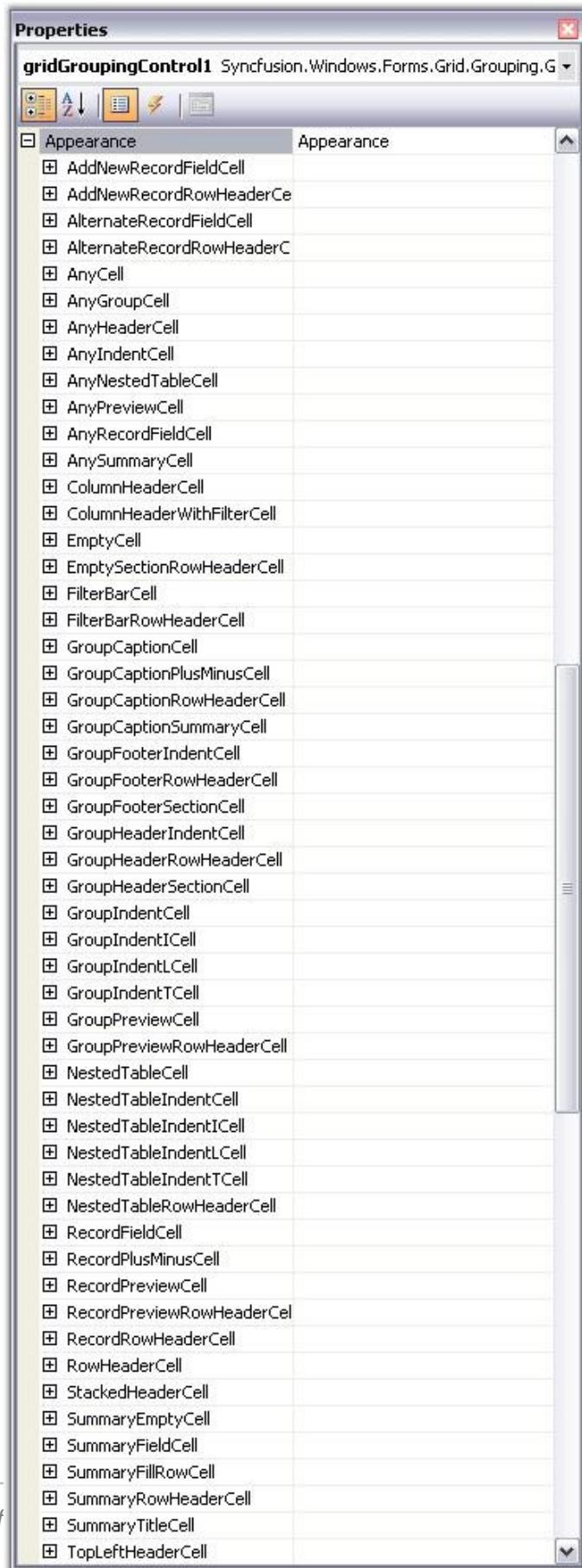
[VB]

```
Dim pager = New Pager With {PageSize = 1000}  
pager.Wire(gridGroupingControl1) 'Data sources assigned to the pager  
from gridGroupingControl
```

#### 4.3.4.5 Appearance

This property allows you to control the appearance of the grouping grid at design time as well as at run time. You can change the overall appearance of the grid and also the appearance of each element in the grid by setting this property.

**Appearance** contains a list of **GridStyleInfo** properties as seen in the following graphic. A **GridStyleInfo** object contains many properties such as **BackColor**, **Font** and **CellType** which defines the look and behavior of a grid cell. Each of these properties identifies a particular set of cells that make up a Grid Grouping control.



*Figure 327: Appearance Properties*

To understand exactly what is going on here, let's consider three of these `GridStyleInfo` properties: `AnyCell`, `AnyRecordFieldCell` and `AnyAlternateRecordFieldCell`. Say we set `AnyCell.BackColor = Color.LightBlue`. This will color any grid cell light blue.



**Note:** If you are using a Themed Operating system, like Windows XP, turn the `GridGroupingControl.ThemesEnabled` property off so that the theme coloring does not affect things like header cell buttons. Otherwise, this will interfere with illustrating the concepts we are trying to communicate in this section.

Next if we set `AnyRecordFieldCell.BackColor = Color.Azure`, we will see the color of any record field cell change to azure. If we then set `AnyAlternateRecordFieldCell.BackColor = Color.LightGreen`, we will see alternate records being displayed with a green background. Below is a picture illustrating the look of the grid after setting each property in order.

There is an inheritance hierarchy that is associated with the `Appearance` properties. The general rule is that, if present, the more specific property takes precedence over the less specific property. This means that `AnyCell.BackColor` is overridden by setting the `AnyRecordFieldCell.BackColor` which, is again overridden by setting even more specific `AnyAlternatingRecordFieldCell.BackColor`.

**Statistics: 225 Items**

	ID	School	Sport	wins	losses	ties
	1	Duke	Basketball	26	7	0
	2	Maryland	Basketball	21	10	0
	3	Wake Forest	Basketball	25	6	0
	4	Georgia Tech	Basketball	16	15	0
	5	Virginia	Basketball	16	16	0
	6	NC State	Basketball	18	13	0
	7	North Carolina	Basketball	19	16	0
	8	Clemson	Basketball	15	13	0
	9	Florida State	Basketball	14	15	0
	10	Duke	Football	3	7	0
	11	Maryland	Football	6	3	0

AnyCell.BackColor set to "LightBlue"

**Statistics: 225 Items**

	ID	School	Sport	wins	losses	ties
	1	Duke	Basketball	26	7	0
	2	Maryland	Basketball	21	10	0
	3	Wake Forest	Basketball	25	6	0
	4	Georgia Tech	Basketball	16	15	0
	5	Virginia	Basketball	16	16	0
	6	NC State	Basketball	18	13	0
	7	North Carolina	Basketball	19	16	0
	8	Clemson	Basketball	15	13	0
	9	Florida State	Basketball	14	15	0
	10	Duke	Football	3	7	0
	11	Maryland	Football	6	3	0

AnyRecordFieldCell.BackColor set to "Azure"

**Statistics: 225 Items**

	ID	School	Sport	wins	losses	ties
	1	Duke	Basketball	26	7	0
	2	Maryland	Basketball	21	10	0
	3	Wake Forest	Basketball	25	6	0
	4	Georgia Tech	Basketball	16	15	0
	5	Virginia	Basketball	16	16	0
	6	NC State	Basketball	18	13	0
	7	North Carolina	Basketball	19	16	0
	8	Clemson	Basketball	15	13	0
	9	Florida State	Basketball	14	15	0
	10	Duke	Football	3	7	0
	11	Maryland	Football	6	3	0

AnyAlternateRecordFieldCell.BackColor set to "LightGreen"

Figure 328: Property Inheritance At Work

#### 4.3.4.5.1 Appearance Options

The simplest way to check exactly which cells are affected by setting one of these properties is to use the Preview and Edit verb to display a Grid Grouping control and then set the property to view the effect.

When using the Preview tool at design-time, there is a cell tip that is displayed over each cell which, gives information regarding that cell. In particular, the first line of the tip will give the exact **Appearance** property that this cell is based on. In addition, it will also list the Appearance properties that the cell inherits. Here is a graphic showing some cell tip samples.

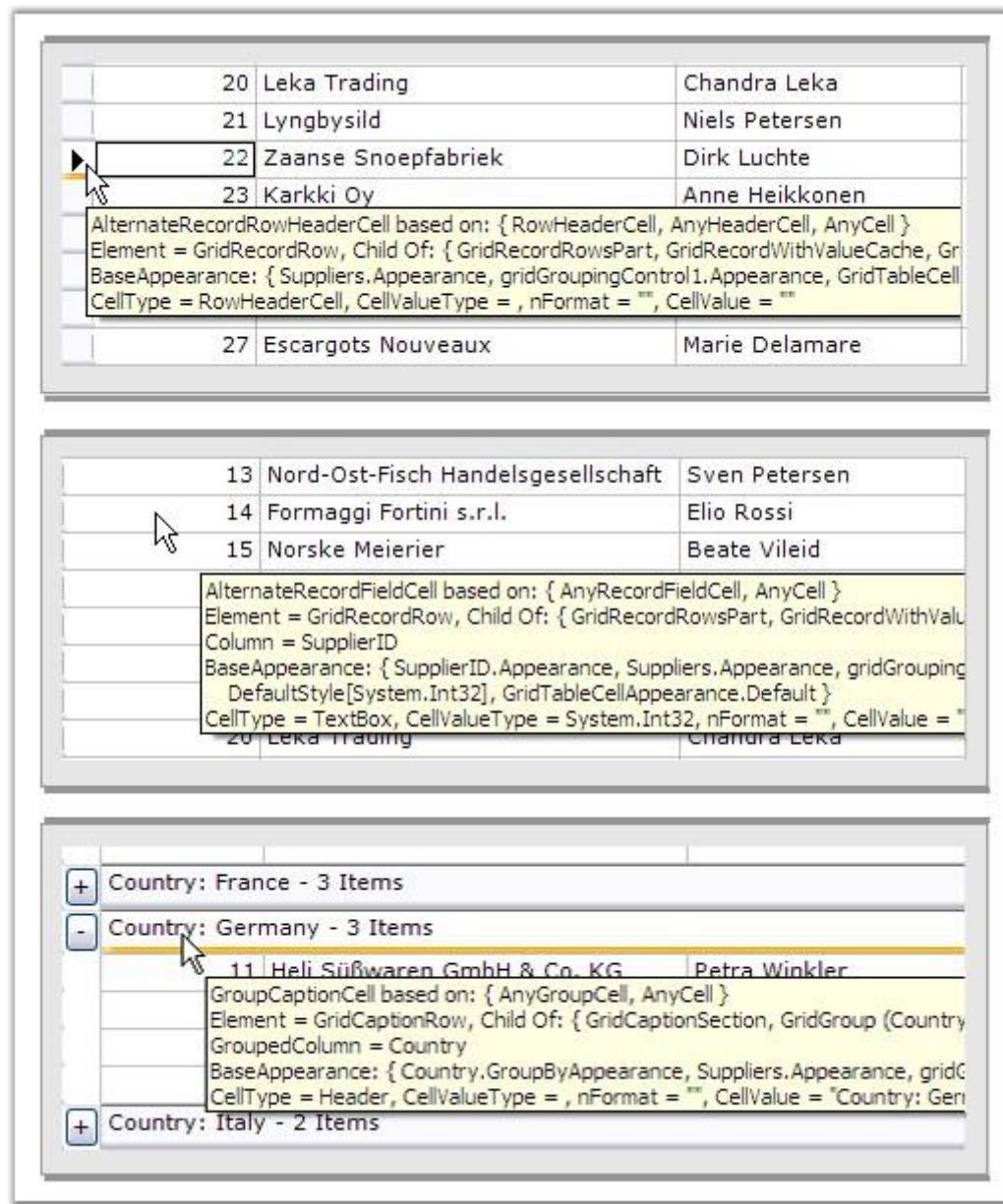


Figure 329: Cell Tips Listing Appearance Inheritance Using Preview at Design-Time

**Note:**

**When using Preview, make sure that you set ThemesEnabled to False, if you want to see the effect of setting the property on a header cell or a button type cell. If you do not, then the theme appearance will supersede the appearance properties you set here.**

**Some of the properties are not applicable unless the item they affect is used in the grid. For example, properties that affect nested tables or summaries will not change the appearance of a Grid Grouping control that does not have either of these items. In later tutorials, you will be able to test such properties.**

### List of Appearance Properties

Appearance Property	Description
AddNewRecordFieldCell	Style information for any cell in a new record row.
AddNewRecordRowHeaderCell	Style information for header cell for any new record.
AlternateRecordFieldCell	Style information for any cell in alternate record rows.
AlternateRecordRowHeaderCell	Style information for any header cell in alternate record rows.
AnyCell	Style information for any cell in the Grid.
AnyGroupCell	Style information for any cell in a Group item.
AnyHeaderCell	Style information for any header cell.
AnyIndentCell	Style information for any indent cell.
AnyNestedTableCell	Style information for any nested table cell.
AnyPreviewCell	Style information for any preview cell.
AnyRecordFieldCell	Style information for any record cell.
AnySummaryCell	Style information for any summary cell.
ColumnHeaderCell	Style information for any column header cell.
ColumnHeaderWithFilterCell	Style information for any column header cell with

	filter.
EmptyCell	Style information for any empty cell.
EmptySectionRowHeaderCell	Style information for any row header cell in an empty section.
FilterBarCell	Style information for any filter bar cell.
FilterBarRowHeaderCell	Style information for any filter bar rowheader cell.
GroupCaptionCell	Style information for any cell in a group caption.
GroupCaptionPlusMinusCell	Style information for any plus-minus cell in a group caption.
GroupCaptionRowHeaderCell	Style information for any row header cell in a group caption.
GroupCaptionSummaryCell	Style information for any summary cell in a group caption.
GroupFooterIndentCell	Style information for any indent cell in a group footer.
GroupFooterRowHeaderCell	Style information for any header cell in a group footer.
GroupFooterSectionCell	Style information for any section cell in a group footer.
GroupHeaderIndentCell	Style information for any indent cell in a group.
GroupHeaderRowHeaderCell	Style information for any cell in a group header.
GroupHeaderSectionCell	Style information for any section cell in a group header.
GroupIndentCell	Style information for any indent cell in a group.
GroupIndentICell	Style information for any indent cell with no connected item.
GroupIndentLCell	Style information for any indent cell with a bottom connected item.
GroupIndentTCell	Style information for any indent cell with a middle

	connected item.
GroupPreviewCell	Style information for any preview cell.
GroupPreviewRowHeaderCell	Style information for any header cell in a preview row.
NestedTableCell	Style information for any cell in a nested table.
NestedTableIndentCell	Style information for any indent cell in a nested table.
NestedTableIndentICell	Style information for any nested table indent cell with a bottom connected item.
NestedTableIndentLCell	Style information for any nested table indent cell with a middle connected item.
NestedTableIndentTCell	Style information for any nested table indent cell with no connected item.
NestedTableRowHeaderCell	Style information for any row header cell in a nested table.
RecordFieldCell	Style information for any field cell in a record row.
RecordPlusMinusCell	Style information for any plus-minus cell in a record row.
RecordPreviewCell	Style information for any preview cell in a record row.
RecordPreviewRowHeaderCell	Style information for any header cell in a record preview row
RecordRowHeaderCell	Style information for any header cell in a record row.
RowHeaderCell	Style information for any row header cell.
SummaryEmptyCell	Style information for any empty cell in a summary row.
SummaryFieldCell	Style information for any field cell in a summary row.

SummaryFillRowCell	Style information for any fill cell in a summary row.
SummaryRowHeaderCell	Style information for any header cell in a summary row.
SummaryTitleCell	Style information for any title cell in a summary row.
TopLeftHeaderCell	Style information for any top left header cell.

#### 4.3.4.5.1.1 Styles At Table Level

This section demonstrates how to provide different appearances to the tables at different levels. Properties set through grid.**TableDescriptor.Appearance** property will be applied to the top level table(parent). To control the appearance of individual child tables, you need to first get the TableDescriptor of the desired Child Table. You can then use **ChildTableDescriptor.Appearance** property to provide the appearances to the respective child table.

#### Example

This implementation applies unique styles to the tables at different levels (Parent and Child). The grouping grid is bound to an hierarchical dataset with two tables. Below is the code to customize the appearance of these tables.

1. Set styles to the Parent Table.

```
[C#]

// Column Header Cell styles.
this.gridGroupingControl1.Appearance.ColumnHeaderCell.CellTipText =
"ColumnHeader";
this.gridGroupingControl1.Appearance.ColumnHeaderCell.Interior = new
BrushInfo(GradientStyle.Vertical, Color.FromArgb(214, 220, 232),
Color.FromArgb(106, 111, 151));
this.gridGroupingControl1.Appearance.ColumnHeaderCell.TextColor =
System.Drawing.Color.White;

// Record Field Cell style.
this.gridGroupingControl1.Appearance.RecordFieldCell.Interior = new
BrushInfo(Color.Lavender);
```

```
// Row Header Cell styles.  
this.gridGroupingControl1.Appearance.RowHeaderCell.Interior = new  
Syncfusion.Drawing.BrushInfo(Syncfusion.Drawing.GradientStyle.Horizontal  
1, SystemColors.Window, Color.FromArgb(206, 213, 231));  
this.gridGroupingControl1.Appearance.RowHeaderCell.Themed = false;  
  
// Top Left Header Cell style.  
this.gridGroupingControl1.Appearance.TopLeftHeaderCell.Interior = new  
BrushInfo(GradientStyle.PathRectangle, SystemColors.Window,  
Color.FromArgb(255, 228, 221));
```

**[VB.NET]**

```
' Column Header Cell styles.  
Me.gridGroupingControl1.Appearance.ColumnHeaderCell.CellTipText =  
"ColumnHeader"  
Me.gridGroupingControl1.Appearance.ColumnHeaderCell.Interior = new  
BrushInfo(GradientStyle.Vertical, Color.FromArgb(214, 220,  
232), Color.FromArgb(106, 111, 151))  
Me.gridGroupingControl1.Appearance.ColumnHeaderCell.TextColor =  
Color.White  
  
' Record Field Cell style.  
Me.gridGroupingControl1.Appearance.RecordFieldCell.Interior = new  
BrushInfo(Color.Lavender)  
  
' Row Header Cell styles.  
Me.gridGroupingControl1.Appearance.RowHeadersCell.Interior = new  
BrushInfo(GradientStyle.Horizontal, SystemColors.Window,  
Color.FromArgb(206, 213, 231))  
Me.gridGroupingControl1.Appearance.RowHeadersCell.Themed = false  
  
' Top Left Header Cell style.  
Me.gridGroupingControl1.Appearance.TopLeftHeaderCell.Interior = new  
BrushInfo(GradientStyle.PathRectangle, SystemColors.Window,  
Color.FromArgb(255, 228, 221))
```

2. Apply styles to the Child Table.

**[C#]**

```
GridTableDescriptor gtd =  
this.gridGroupingControl1.GetTableDescriptor("Orders");  
  
// Record Field Cell styles.  
gtd.Appearance.AnyRecordFieldCell.BackColor = Color.FromArgb(223, 247,  
252);
```

```
gtd.Appearance.AlternateRecordFieldCell.BackColor = Color.FromArgb(255,  
229, 201);  
  
// Column Header Cell styles.  
gtd.Appearance.ColumnHeaderCell.Interior = new  
BrushInfo(GradientStyle.Vertical, Color.FromArgb(203, 201, 202),  
Color.FromArgb(253, 247, 215));  
gtd.Appearance.ColumnHeaderCell.TextColor = Color.Black;  
  
// Group Caption Cell styles.  
gtd.Appearance.GroupCaptionCell.Interior = new  
BrushInfo(Color.FromArgb(255, 238, 220));  
gtd.Appearance.GroupCaptionCell.Borders.Bottom = new  
GridBorder(GridBorderStyle.Solid, Color.FromArgb(242, 158, 32),  
GridBorderWeight.Medium);
```

**[VB.NET]**

```
Dim gtd As GridTableDescriptor =  
Me.gridGroupingControll1.GetTableDescriptor("Orders")  
  
' Record Field Cell styles.  
gtd.Appearance.AnyRecordFieldCell.BackColor = Color.FromArgb(223, 247,  
252)  
gtd.Appearance.AlternateRecordFieldCell.BackColor = Color.FromArgb(255,  
229, 201)  
  
' Column Header Cell styles.  
gtd.Appearance.ColumnHeaderCell.Interior = New  
BrushInfo(GradientStyle.Vertical, Color.FromArgb(203, 201, 202),  
Color.FromArgb(253, 247, 215))  
gtd.Appearance.ColumnHeaderCell.TextColor = Color.Black  
  
' Group Caption Cell styles.  
gtd.Appearance.GroupCaptionCell.Interior = New  
BrushInfo(Color.FromArgb(255, 238, 220))  
gtd.Appearance.GroupCaptionCell.Borders.Bottom = New  
GridBorder(GridBorderStyle.Solid, Color.FromArgb(242, 158, 32),  
GridBorderWeight.Medium)
```

Here is a sample screenshot.

The screenshot shows a Windows application window titled "Customers: 91 Items". Inside, there are two tables. The first table, titled "Customers", has columns for CustomerID, CompanyName, and ContactName. It lists four groups: ALFKI, ANATR, ANTON, and AROUT. The second table, titled "Orders: 13 Items", has columns for OrderID, EmployeeID, OrderDate, and RequiredDate. It lists six orders. The application includes standard Windows-style scroll bars and a toolbar at the bottom.

Figure 330: Customized Appearance of Tables at Different Levels



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Grouping.Windows\Samples\2.0\Appearance\Table Style Demo

#### 4.3.4.5.1.2 Styles At Group Level

This section lets you customize the appearances of different group elements. You can provide unique appearances to every element of a group such as GroupCaptionCell and Group Header / Footer Cells by setting the following properties under Appearance section: **GroupCaptionCell**, **GroupCaptionPlusMinusCell**, **GroupHeaderSectionCell**, **GroupIndentCell**, **GroupFooterSectionCell**, **GroupPreviewCell** and the like.

#### Example

Below is the code to apply different styles to various group members.

[C#]

```
this.gridGroupingControl1.Appearance.AnyGroupCell.Interior = new
```

```
BrushInfo(Color.White);
this.gridGroupingControl1.Appearance.AnyGroupCell.Themed = false;
this.gridGroupingControl1.Appearance.GroupCaptionCell.Borders.Bottom =
GridBorder(GridBorderStyle.Solid, Color.FromArgb(157, 179, 200));
this.gridGroupingControl1.Appearance.GroupCaptionRowHeaderCell.Interior =
= new BrushInfo(GradientStyle.BackwardDiagonal, SystemColors.Window,
Color.DarkOrange);
this.gridGroupingControl1.Appearance.GroupFooterSectionCell.Interior =
new BrushInfo(GradientStyle.Horizontal, Color.White,
Color.FromArgb(192, 255, 192));
this.gridGroupingControl1.Appearance.GroupHeaderRowHeaderCell.Interior =
= new BrushInfo(GradientStyle.Vertical, SystemColors.Window,
Color.LightPink);
this.gridGroupingControl1.Appearance.GroupHeaderSectionCell.Interior =
new BrushInfo(GradientStyle.Horizontal, Color.White,
Color.FromArgb(255, 199, 190));
```

**[VB .NET]**

```
gridGroupingControl1.Appearance.AnyGroupCell.Interior = New
BrushInfo(Color.White)
gridGroupingControl1.Appearance.AnyGroupCell.Themed = False
gridGroupingControl1.Appearance.GroupCaptionCell.Borders.Bottom =
GridBorder(GridBorderStyle.Solid, Color.FromArgb(157, 179, 200))
gridGroupingControl1.Appearance.GroupCaptionRowHeaderCell.Interior =
New BrushInfo(GradientStyle.BackwardDiagonal, SystemColors.Window,
Color.DarkOrange)
gridGroupingControl1.Appearance.GroupFooterSectionCell.Interior = New
BrushInfo(GradientStyle.Horizontal, Color.White, Color.FromArgb(192,
255, 192))
gridGroupingControl1.Appearance.GroupHeaderRowHeaderCell.Interior = New
BrushInfo(GradientStyle.Vertical, SystemColors.Window, Color.LightPink)
gridGroupingControl1.Appearance.GroupHeaderSectionCell.Interior = New
BrushInfo(GradientStyle.Horizontal, Color.White, Color.FromArgb(255,
199, 190))
```

Here is a sample screen shot.



Figure 331: Customized Appearance of Groups at Different Levels



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Appearance\Group Style Demo

#### 4.3.4.5.1.3 ColumnStyles

Grid Grouping control allows you to do **Column-Based formatting**. With this feature, you can provide an unique appearance to different grid columns. Grid columns can be customized by setting the **GridColumnDescriptor.Appearance** property.

ColumnFormatting can be done at design time. Once the data source is set, select TableDescriptor.Columns property in the property window of the grid grouping control. This will open the GridColumnDescriptor collection editor that is populated with the columns in the datasource. You can modify the appearance of the desired column by setting the Appearance property of that column in this editor. The following picture shows this process.

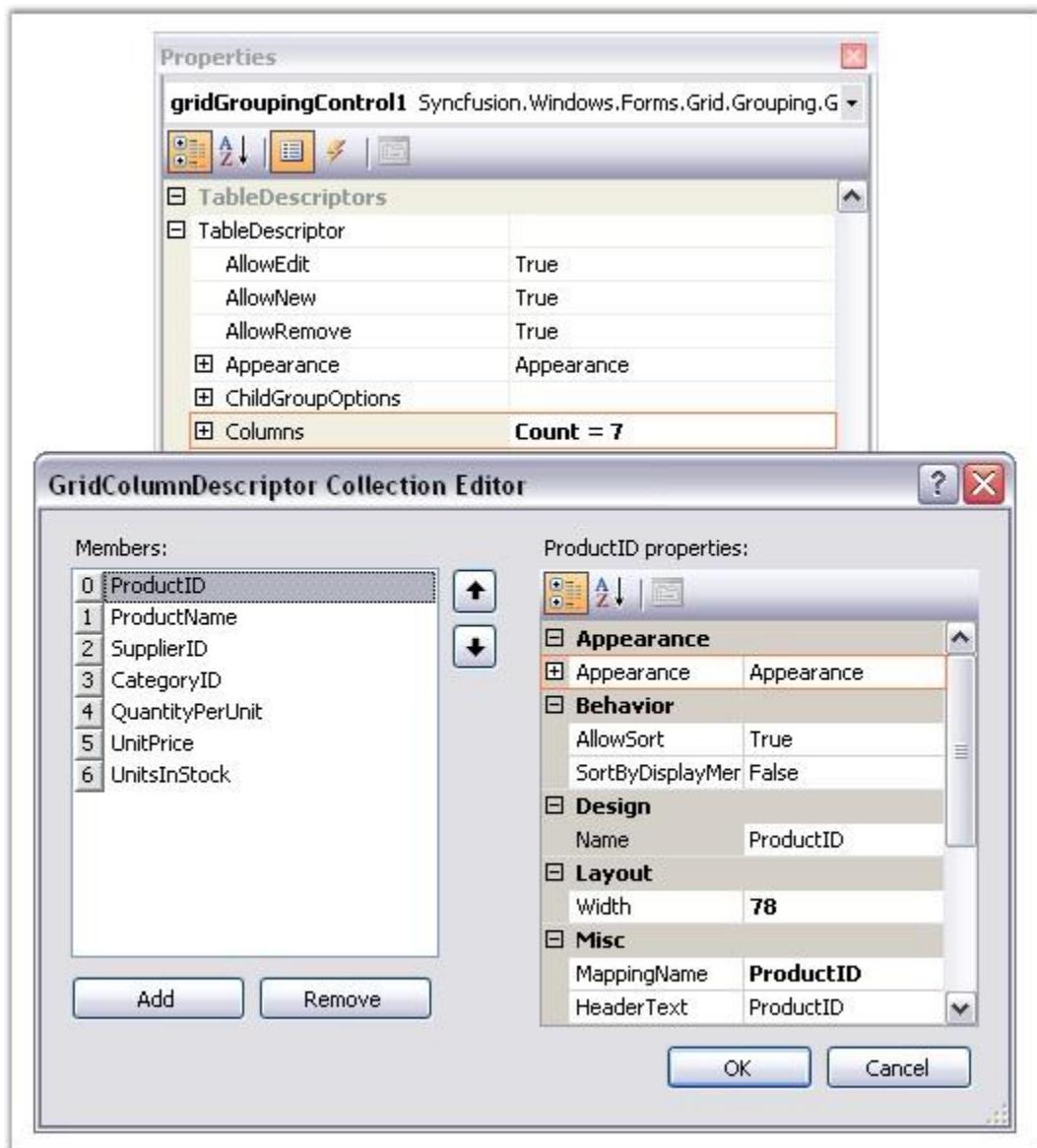


Figure 332: GridColumnDescriptor Collection Editor

## Programmatically

You can control the appearance of the columns through code also. Below is the code that applies different styles to the various columns in the grid.

[C#]

```
GridColumnDescriptor desc1 = new GridColumnDescriptor();
desc1.MappingName = "ProductName";
```

```
desc1.Appearance.RecordFieldCell.Interior = new  
BrushInfo(Color.FromArgb(237, 240, 246));  
  
GridColumnDescriptor desc2 = new GridColumnDescriptor();  
desc2.MappingName = "SupplierID";  
desc2.Appearance.RecordFieldCell.Interior = new  
BrushInfo(Color.FromArgb(218, 229, 245));  
  
GridColumnDescriptor desc3 = new GridColumnDescriptor();  
desc3.MappingName = "CategoryID";  
desc3.Appearance.RecordFieldCell.Interior = new  
BrushInfo(Color.FromArgb(102, 110, 152));  
  
GridColumnDescriptor desc4 = new GridColumnDescriptor();  
desc4.MappingName = "QuantityPerUnit";  
desc4.Appearance.RecordFieldCell.Interior = new  
BrushInfo(Color.FromArgb(252, 172, 38));
```

**[VB.NET]**

```
Dim desc1 As GridColumnDescriptor = New GridColumnDescriptor  
desc1.MappingName = "ProductName"  
desc1.Appearance.RecordFieldCell.Interior = New  
BrushInfo(Color.FromArgb(237, 240, 246))  
  
Dim desc1 As GridColumnDescriptor = New GridColumnDescriptor  
desc1.MappingName = "ProductName"  
desc1.Appearance.RecordFieldCell.Interior = New  
BrushInfo(Color.FromArgb(218, 229, 245))  
  
Dim desc1 As GridColumnDescriptor = New GridColumnDescriptor  
desc1.MappingName = "SupplierID"  
desc1.Appearance.RecordFieldCell.Interior = New  
BrushInfo(Color.FromArgb(102, 110, 152))  
  
Dim desc1 As GridColumnDescriptor = New GridColumnDescriptor  
desc1.MappingName = "QuantityPerUnit"  
desc1.Appearance.RecordFieldCell.Interior = New  
BrushInfo(Color.FromArgb(252, 172, 38))
```

Given below is a sample screen shot.

	ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit
	1	Chai	1	1	10 boxes x 20 bags
	2	Chang	1	1	24 - 12 oz bottles
	3	Aniseed Syrup	1	2	12 - 550 ml bottles
	4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars
	5	Chef Anton's Gumbo Mix	2	2	36 boxes
	6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars
	7	Uncle Bob's Organic Dried Pears	3	7	12 - 1 lb pkgs.
	8	Northwoods Cranberry Sauce	3	2	12 - 12 oz jars
	9	Mishi Kobe Niku	4	6	18 - 500 g pkgs.
	10	Ikura	4	8	12 - 200 ml jars
	11	Queso Cabrales	5	4	1 kg pkg.
	12	Queso Manchego La Pastora	5	4	10 - 500 g pkgs.
	13	Konbu	6	8	2 kg box

Figure 333: Customized Appearance of Grid Columns



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Grouping.Windows\Samples\2.0\Appearance\Column Style Demo

#### 4.3.4.5.1.4 Render Images in Header Cells

Header cells are enhanced to render images along with their header text to represent the data more clearly. The user can align the images as desired.

##### Use Case Scenarios

In a payroll application, the images in the header cells help the user understand the nature of the data type in which the column is supposed to be bound.

Employees Drag a column header here to group by that column.				
EmployeeID	LastName	FirstName	Title	BirthDate
1	Davolio	Nancy	Sales Representative	12/8/1968 12:00:00 AM
2	Fuller	Andrew	Vice President, Sales	2/19/1952 12:00:00 AM
3	Leverling	Janet	Sales Representative	8/30/1963 12:00:00 AM
4	Peacock	Margaret	Sales Representative	9/19/1958 12:00:00 AM
5	Buchanan	Steven	Sales Manager	3/4/1955 12:00:00 AM

Figure 334: Header Cells with Images

## Properties

Table 10: Properties Table

Property	Description	Type	Data Type
HeaderImage	Gets or sets images to display in the header cells.	Image	Image
HeaderAlignment	Get or sets the alignment of the image in the header.	Enumeration	Enumeration

## Sample Link

Find a sample in the following location:

<Install Location>\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Grouping.Windows\Samples\2.0\Grouping\Grouping Demo

## Adding Images at Header Cells to an Application

To display the images in the header cells, pass the image through the **GridColumnDescriptor**. The following code helps you to set the image for a specific column header.

### [C#]

```
this.gridGroupingControl1.TableDescriptor.Columns[Name/Index].HeaderImage = Image.FromFile(@"..\..\image.PNG");
```

### [VB.NET]

```
Me.gridGroupingControl1.TableDescriptor.Columns(Name/Index).HeaderImage = Image.FromFile(@"..\..\image.PNG")
```

To switch the alignment of the image between right and left of the header cell, the enumeration property **HeaderImageAlignment** is used. The code helps you to set the alignment.

### [C#]

```
this.gridGroupingControl1.TableDescriptor.Columns[Name/Index].HeaderImageAlignment =
Syncfusion.Windows.Forms.Grid.Grouping.HeaderImageAlignment.Right;
```

### [VB.NET]

```
Me.gridGroupingControl1.TableDescriptor.Columns(Name/Index).HeaderImageAlignment =
Syncfusion.Windows.Forms.Grid.Grouping.HeaderImageAlignment.Right
```



Employees Drag a column header here to group by that column.				
EmployeeID	LastName	FirstName	Title	BirthDate
1	Davolio	Nancy	Sales Representative	12/8/1968 12:00:00 AM
2	Fuller	Andrew	Vice President, Sales	2/19/1952 12:00:00 AM
3	Leverling	Janet	Sales Representative	8/30/1963 12:00:00 AM
4	Peacock	Margaret	Sales Representative	9/19/1958 12:00:00 AM
5	Buchanan	Steven	Sales Manager	3/4/1955 12:00:00 AM
6	Suyama	Michael	Sales Representative	7/2/1963 12:00:00 AM
7	King	Robert	Sales Representative	5/29/1960 12:00:00 AM
8	Callahan	Laura	Inside Sales Coordinator	1/9/1958 12:00:00 AM
9	Dodsworth	Anne	Sales Representative	7/2/1969 12:00:00 AM

Figure 335: Header Cells with Aligned Images

#### 4.3.4.5.2 Format Cells Dialog Support

The GridGrouping control provides support to apply styles using a Format Cell dialog as found in Microsoft Excel.

#### Use Case Scenarios

This feature enables you to dynamically format such cell attributes as cell background color, font, text color, alignment, and more.

#### Methods

Table 11: Method Table

Method	Description	Parameters	Type	Return Type	Reference links
<code>GroupingGridFormatCellDialog ()</code>	Uses to wire the Grid with FormatCell Dialog	GridGoupingControl	Method (Constructor)	N/A.	N/A.

#### Sample Link

A demo of this feature is available in the following location:

..\\AppData\\Local\\Syncfusion\\EssentialStudio\\{Version}\\Windows\\Grid.Grouping.Window\\Samples\\2.0\\Appearance\\FormatCells Dialog Demo

### **Adding GridFormatCellDialog To The GridGroupingControl**

You can add the cell formatting dialog using the *GridFormatCellDialog* class. To add the *GridFormatCellDialog*, pass the *GridGroupingControl* as a parameter of the *GroupingGridFormatCellDialog()* method.

The following code illustrates this:

[C#]

```
GroupingGridFormatCellDialog Dialog = new  
GroupingGridFormatCellDialog(this.gridGroupingControl1);  
Dialog.ShowDialog();
```

[VB]

```
Dim Dialog As GroupingGridFormatCellDialog = New  
GroupingGridFormatCellDialog(Me.gridGroupingControl1)  
Dialog.ShowDialog()
```

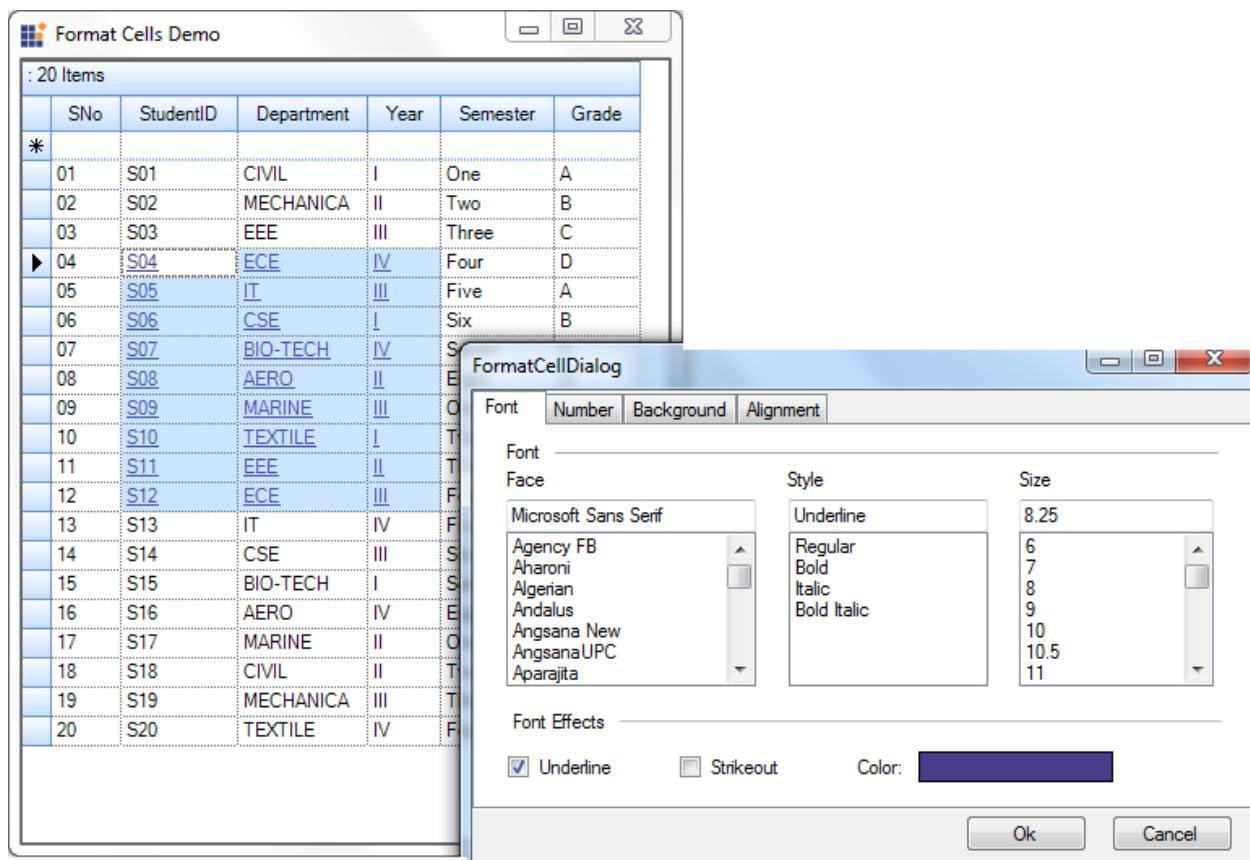


Figure 336: Format Cell Dialog

#### 4.3.4.5.3 Conditional Formatting

Grid Grouping control has in-built support for Conditional Formatting. This feature allows you to format grid cells based on a certain condition. This can be achieved by defining a **GridConditionalFormatDescriptor** for the grid. Using this descriptor, you can specify the filter criteria for the cells and the style to be applied for the filtered cells. Once these specifications are defined, then the given styles are applied to only those cells that satisfy the condition specified.

Conditional Formatting can be specified through the designer itself by accessing the **TableDescriptor.ConditionalFormats** property. This will open the **GridConditionalFormatDescriptor** editor wherein you can add as many formatters as you want. For each such formatter, you need to specify the filter criteria either by adding **RecordFilters** or by an **Expression**. The below property editor illustrates this process.

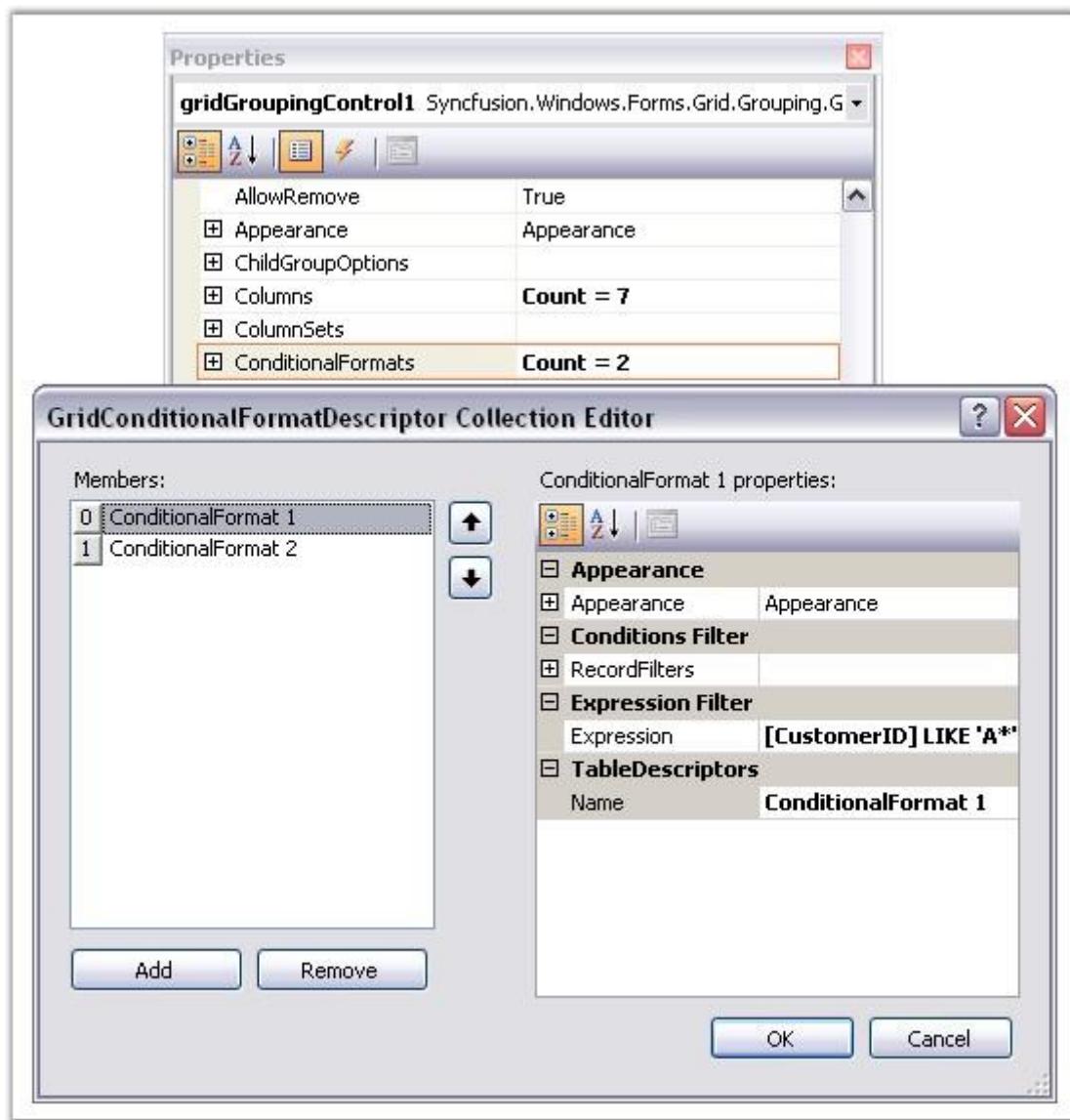


Figure 337: GridConditionalFormatDescriptor Collection Editor

### Programmatically

Following code example illustrates how to apply conditional formatting to the grouping grid.

1. Define a Conditional Format Descriptor and specify a filter criteria and style to be applied.

[C#]

```
// Apply the following style to the records whose CustomerID starts
// with 'A'.
```

```
GridConditionalFormatDescriptor format1 = new
GridConditionalFormatDescriptor();
format1.Appearance.AnyRecordFieldCell.Interior = new
BrushInfo(Color.FromArgb(255, 191, 52));
format1.Appearance.AnyRecordFieldCell.TextColor = Color.White;
format1.Expression = "[CustomerID] LIKE '\A*\'";
format1.Name = "ConditionalFormat 1";

// Apply the following style to the records whose ContactTitle = 'Sales
Representative'.
GridConditionalFormatDescriptor format2 = new
GridConditionalFormatDescriptor();
format2.Appearance.AnyRecordFieldCell.Font.Bold = true;
format2.Appearance.AnyRecordFieldCell.Interior = new
BrushInfo(Color.FromArgb(102, 110, 152));
format2.Appearance.AnyRecordFieldCell.TextColor = Color.White;
format2.Expression = "[ContactTitle] LIKE '\Sales Representative\'";
format2.Name = "ConditionalFormat 2";
```

**[VB.NET]**

```
'Apply the following style to the records whose CustomerID starts with
'A'
Dim format1 As GridConditionalFormatDescriptor = New
GridConditionalFormatDescriptor()
format1.Appearance.AnyRecordFieldCell.Interior = New
BrushInfo(Color.FromArgb(255, 191, 52))
format1.Appearance.AnyRecordFieldCell.TextColor = Color.White
format1.Expression = "[CustomerID] LIKE '\A*\'"
format1.Name = "ConditionalFormat 1"

'Apply the following style to the records whose ContactTitle = 'Sales
Representative'
Dim format2 As GridConditionalFormatDescriptor = New
GridConditionalFormatDescriptor()
format2.Appearance.AnyRecordFieldCell.Font.Bold = True
format2.Appearance.AnyRecordFieldCell.Interior = New
BrushInfo(Color.FromArgb(102, 110, 152))
format2.Appearance.AnyRecordFieldCell.TextColor = Color.White
format2.Expression = "[ContactTitle] LIKE '\Sales Representative\'"
format2.Name = "ConditionalFormat 2"
```

2. Add the descriptor to the TableDescriptor.ConditionalFormats property.

**[C#]**

```
this.gridGroupingControll1.TableDescriptor.ConditionalFormats.Add(format)
```

```
1);
this.gridGroupingControl1.TableDescriptor.ConditionalFormats.Add(format2);
```

**[VB.NET]**

```
Me.gridGroupingControl1.TableDescriptor.ConditionalFormats.Add(format1)
Me.gridGroupingControl1.TableDescriptor.ConditionalFormats.Add(format2)
```

Given below is a sample screenshot.

CustomerID	CompanyName	ContactName
ALFKI	Alfreds Futterkiste	Maria Anders
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo
ANTON	Antonio Moreno Taqueria	Antonio Moreno
AROUT	Around the Horn	Thomas Hardy
BERGS	Berglunds snabbköp	Christina Berglund
BLAUS	Blauer See Delikatessen	Hanna Moos
BLONP	Blondel père et fils	Frédérique Citeaux
BOLID	Bólido Comidas preparadas	Martín Sommer
BONAP	Bon app'	Laurence Lebihan
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln
BSBEV	B's Beverages	Victoria Ashworth

Figure 338: Conditional Formatting applied to the Grid Grouping Control



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Appearance\Conditional Formatting Demo

#### 4.3.4.5.4 Dynamic Formatting

Style Settings can be applied to different grid elements dynamically at run time. This can be achieved by proper handling of the **QueryCellStyleInfo** event. It provides the GridStyleInfo object for a cell on demand.

**QueryCellStyleInfo** is raised every time a request is made to access the style information for a cell. You can do any type of formatting cells with this event. It accepts **GridTableCellStyleEventArgs** as one of its parameters which can be used to customize the cells of the grouping grid control. For instance, you can apply style settings for a given CellType by using the **TableCellIdentity.TableCellType** property on the instances of **GridTableCellStyleEventArgs**. **style** for a given cell.

Here is an example code that applies different styles to different cells in the grouping grid.

```
[C#]

// Hook up the event.
this.gridGroupingControl1.QueryCellStyleInfo += new
GridTableCellStyleInfoEventHandler(gridGroupingControl1_QueryCellStyleInfo);

// QueryCellStyleInfo event : To Format Cell by Cell Basis on demand.
private void gridGroupingControl1_QueryCellStyleInfo(object sender,
GridTableCellStyleInfoEventArgs e)
{
    if(e.TableCellIdentity.TableCellType ==
GridTableCellType.RecordFieldCell)
    {
        if(e.TableCellIdentity.ColumnIndex %2 == 0)
        {
            e.Style.BackColor = Color.FromArgb(255, 187, 111);
            e.Style.Font.FontStyle = FontStyle.Bold;
        }
        else
        {
            e.Style.TextColor = Color.White;
            e.Style.BackColor = Color.FromArgb(55, 91, 114);
        }
    }
    else if( e.TableCellIdentity.TableCellType ==
GridTableCellType.AlternateRecordFieldCell)
    {
        if(e.TableCellIdentity.ColumnIndex%2==0)
        {
            e.Style.Font.FontStyle = FontStyle.Underline;
            e.Style.BackColor = Color.FromArgb(0, 21, 84);
            e.Style.TextColor = Color.White;
        }
        else
        {
            e.Style.BackColor = Color.FromArgb(255, 188, 112);
        }
    }
}
```

```
        e.Style.Font.FontStyle = FontStyle.Italic;
    }
}
}
```

**[VB.NET]**

```
' Hook up the event.
AddHandler gridGroupingControl1.QueryCellStyleInfo, AddressOf
gridGroupingControl1_QueryCellStyleInfo

' QueryCellStyleInfo event : To Format Cell by Cell Basis on demand.
Private Sub gridGroupingControl1_QueryCellStyleInfo(ByVal sender As
Object, ByVal e As GridTableCellStyleEventArgs)
If e.TableCellIdentity.TableCellType =
GridTableCellType.RecordFieldCell Then
    If e.TableCellIdentity.ColumnIndex Mod 2 = 0 Then
        e.Style.BackColor = Color.FromArgb(255, 187, 111)
        e.Style.Font.FontStyle = FontStyle.Bold
    Else
        e.Style.TextColor = Color.White
        e.Style.BackColor = Color.FromArgb(55, 91, 114)
    End If
ElseIf e.TableCellIdentity.TableCellType =
GridTableCellType.AlternateRecordFieldCell Then
    If e.TableCellIdentity.ColumnIndex Mod 2 = 0 Then
        e.Style.BackColor = Color.FromArgb(0, 21, 84)
        e.Style.Font.FontStyle = FontStyle.Underline
        e.Style.TextColor = Color.White
    Else
        e.Style.BackColor = Color.FromArgb(255, 188, 112)
        e.Style.Font.FontStyle = FontStyle.Italic
    End If
End If
End Sub
```

Given below is a sample screen shot.

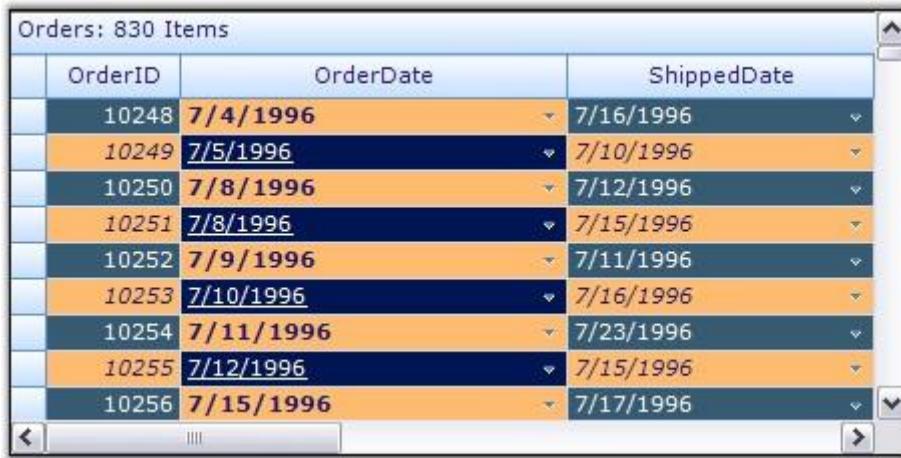


Figure 339: Dynamic Formatting applied to the Grid Grouping Control



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Grouping.Windows\Samples\2.0\Appearance\Dynamic Formatting Demo

#### 4.3.4.5.5 Merge Cells Dynamically

Merging can be applied to the cells dynamically based on the cell values. The merged state will be preserved after any dynamic change such as sorting or grouping.

By default, when merging is applied in the cells, the Grid's bounds will be drawn with respect to the MergeCellsMode applied in the Grid. In the case of the GridGrouping control, when the Grid's view layout is changed, the merged state becomes invalid as the Grid still has its initial state bounds.

To preserve this merging after the dynamic change in the grid's layout, the existing MergeCellsMode is modified to update the grid bounds after they change, and provides additional support to query the user whether to always update the range of cells to merge.

#### Properties

Property	Description	Type	Data Type
MergeCellsLayout	Sets the range of cells to merge in the grid.	Enum	enumeration

#### Sample Link

{Installed  
Drive}\AppData\Local\Syncfusion\EssentialStudio\{version}\Windows\Grid.Grouping.Windows\Samples\2.0\Grouping\Grouping Demo

#### *4.3.4.5.1 Merging Cells Dynamically in a Grid*

To apply dynamic merging in the GridGrouping control, the MergeCellsLayout needs to be applied along with the existing code to merge the cells in the grid.

**[C#]**

```
//Existing code to set merge cells.  
  
this.gridGroupingControl1.TableDescriptor.Columns[colName].Appearance.AnyRecordFieldCell.MergeCell = GridMergeCellDirection.Both;  
  
this.gridGroupingControl1.TableModel.Options.MergeCellsMode = GridMergeCellsMode.OnDemandCalculation ;  
  
// To set the range of cells.  
  
this.gridGroupingControl1.TableModel.Options.MergeCellsLayout = GridMergeCellsLayout.Grid;
```

**[VB]**

```
'Existing code to set merge cells.  
  
Me.gridGroupingControl1.TableDescriptor.Columns(colName).Appearance.AnyRecordFieldCell.MergeCell = GridMergeCellDirection.Both  
  
Me.gridGroupingControl1.TableModel.Options.MergeCellsMode = GridMergeCellsMode.OnDemandCalculation  
  
'To set the range of cells  
  
Me.gridGroupingControl1.TableModel.Options.MergeCellsLayout = GridMergeCellsLayout.Grid
```

#### *4.3.4.5.6 BaseStyles*

In addition to the parent styles discussed in the previous topics, Essential Grid supports one other parent-type style which, can contribute to a cell's appearance, they are BaseStyles of GridStyleInfo objects which, can be associated with an arbitrary collection of cells.

BaseStyles provide the way to create StyleTemplates that can be applied to the cells. It allows you to apply styles with ease and in faster manner. For example, in a word processing software, there is the common task of defining a particular style (such as style Header1 representing a bold, 20-point Helvetica font) and then using it repeatedly in your document whenever you need a 'Header1' type.

BaseStyles play the same role within Essential Grid. You can define a BaseStyle named Header1 as having certain properties and then you can place these properties onto any cell just by applying this BaseStyle Header1 to the cell. More importantly is that if later on you want to change what Header1 means (for example, changing its BackColor property from white to red), you can make the change one time by just changing the Header1 BaseStyle and not having to relabel every other cell assigned to this BaseStyle.

BaseStyles are stored in the `GridGroupingControl.TableModel.BaseStylesMap` class. In addition to the standardstyle, other BaseStyles used by all Essential Grids include Row Header, Header and Column Header. You can define and apply your own BaseStyles as well.

Users can add base styles to the engine and inherit the style settings through `GridStyleInfo.BaseStyle` property. You can create any number of style templates through BaseStyles.

#### Applying BaseStyles

1. To add style templates at design time, you need to access the `BaseStyles` property in the property editor. This will open the `GridTableStyle` Collection Editor that lists the `StyleInfo` properties that can be associated to a grid cell. Here is a property editor that shows the creation of two style templates named `BaseStyle1` and `BaseStyle2`.

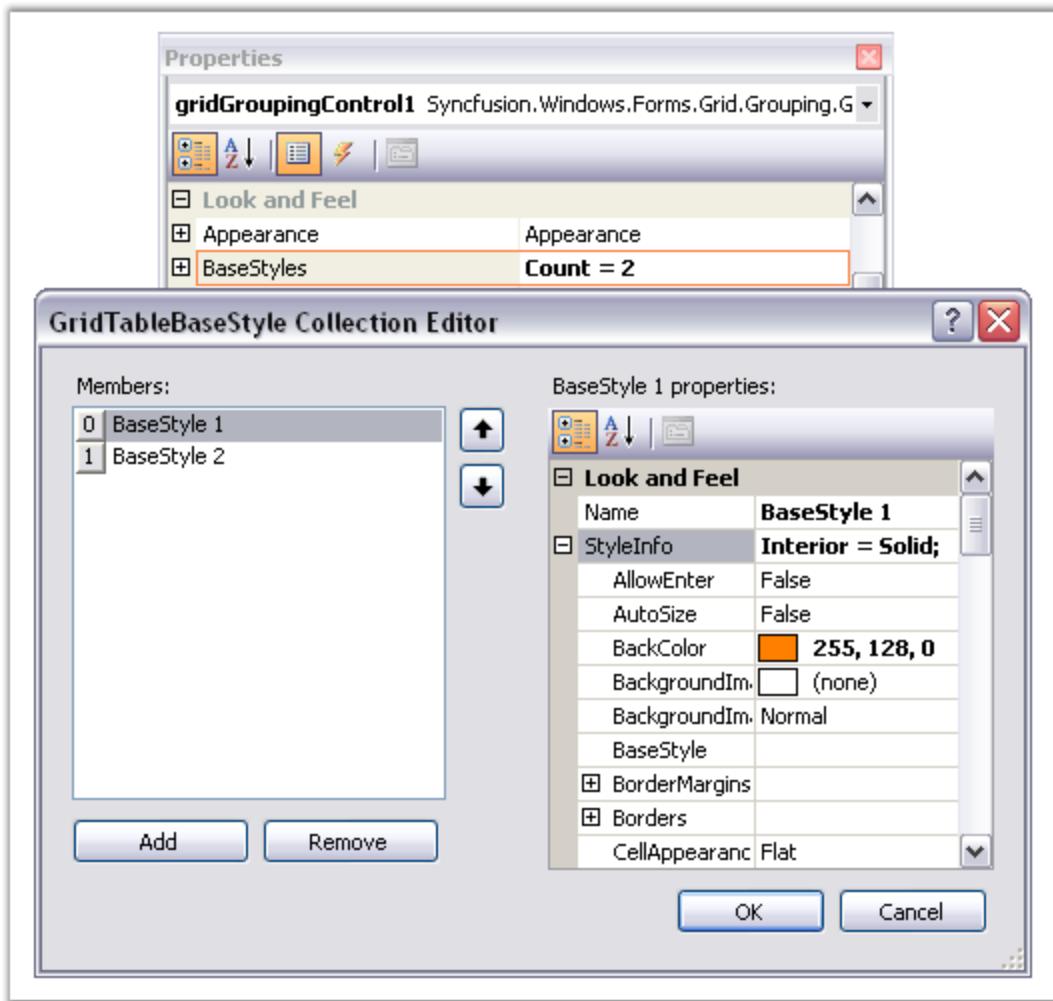


Figure 340: GridTableBaseStyle Collection Editor

2. Your next step is to set the base styles created above, to the grid cells as required. Suppose if you want to set BaseStyle1 for alternate record field cells and BaseStyle2 for the remaining cells, then this can be specified by setting Appearance.AlternateRecordFieldCell.BaseStyle property to BaseStyle1 and Appearance.AnyCell.BaseStyle property to BaseStyle2 as shown in the image below.

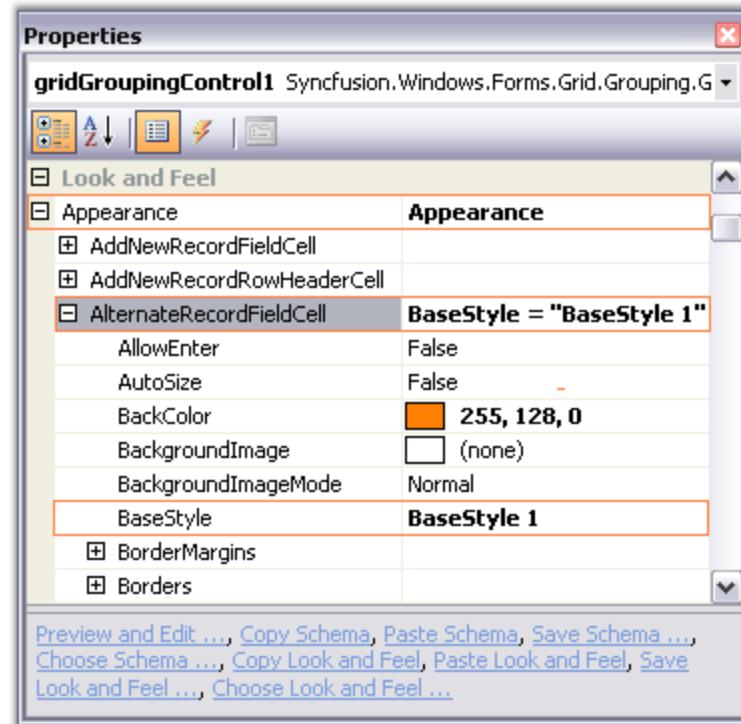


Figure 341: Setting the Base Styles to the respective Grid Cells

3. Here is a sample screenshot.

	CustomerID	CompanyName	ContactName
	ALFKI	Alfreds Futterkiste	Maria Anders
	ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo
	ANTON	Antonio Moreno Taquería	Antonio Moreno
	AROUT	Around the Horn	Thomas Hardy
	BERGS	Berglunds snabbköp	Christina Berglund
	BLAUS	Blauer See Delikatessen	Hanna Moos
	BLONP	Blondel père et fils	Frédérique Citeaux
	BOLID	Bólido Comidas preparadas	Martín Sommer
	BONAP	Bon app'	Laurence Lebihan
	BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln
	BSBEV	B's Beverages	Victoria Ashworth

Figure 342: Base Styles applied to the Grid Grouping Control

## Programmatically

Base styles can also be set through code. The following code example illustrates how to create and apply the above styles to the grouping grid.

**[C#]**

```
GridTableBaseStyle style1 = new GridTableBaseStyle("BaseStyle 1");
style1.Name = "BaseStyle 1";
style1.StyleInfo.Font.Facename = "Verdana";
style1.StyleInfo.Interior = new BrushInfo(Color.FromArgb(255, 128, 0));

GridTableBaseStyle style2 = new GridTableBaseStyle("BaseStyle 2");
style2.Name = "BaseStyle 2";
style2.StyleInfo.Font.Facename = "Arial";
style2.StyleInfo.Interior = new BrushInfo(Color.FromArgb(192, 192, 255));

gridGroupingControl1.BaseStyles.AddRange(new GridTableBaseStyle[] { style1,
style2 });

gridGroupingControl1.Appearance.AlternateRecordFieldCell.BaseStyle =
"BaseStyle 1";
gridGroupingControl1.Appearance.AnyCell.BaseStyle = "BaseStyle 2";
```

**[VB.NET]**

```
Dim style1 As GridTableBaseStyle = New GridTableBaseStyle("BaseStyle 1")
style1.Name = "BaseStyle 1"
style1.StyleInfo.Font.Facename = "Verdana"
style1.StyleInfo.Interior = New BrushInfo(Color.FromArgb(255, 128, 0))

Dim style2 As GridTableBaseStyle = New GridTableBaseStyle("BaseStyle 2")
style2.Name = "BaseStyle 2"
style2.StyleInfo.Font.Facename = "Arial"
style2.StyleInfo.Interior = New BrushInfo(Color.FromArgb(192, 192, 255))
```

```

gridGroupingControl1.BaseStyles.AddRange(New GridTableBaseStyle() { style1,
style2 });

gridGroupingControl1.Appearance.AlternateRecordFieldCell.BaseStyle =
"BaseStyle 1";
gridGroupingControl1.Appearance.AnyCell.BaseStyle = "BaseStyle 2";

```

#### 4.3.4.5.7 Get Cell Styles

This topic elaborates the way of retrieving the style information of grid cells. On a mouse hit, when you want to retrieve the content of underlying cells and also its style information, it is good to use the **PointToTableCellStyle** method on the instances of the Grid Table control.

##### PointToTableCellStyleInfo Method

For any given point on the grid, this method will return the style information of the underlying cell that is displayed under that point. If the underlying cell belongs to a nested table, then style information is returned for the cell inside the nested table. The details of this method are given below.

Method Name	Parameter	Return Value
PointToTableCellStyle	ptClient: A type of System.Drawing Point object that represents the mouse position in client coordinates.	A GridTableCellStyleInfo object that stores the style information of the underlying grid cell.

##### Implementation

The implementation of this method is a two-step process.

- As a first step, it gets the corresponding nested display element that is displayed at the given mouse position. This can be performed easily by employing the **PointToNestedDisplayElement** method. This method is explained later in this chapter.
-

- Once the display element is retrieved, the style information of the corresponding cell can be got by using the **Table.GetTableCellStyleInfo** method which will return a cell style information given its row and column indices.

### PointToNestedDisplayElement Method

This method returns the nested display element that is displayed at the given mouse position. The details are given below.

Method Name	Parameter	Return Value
PointToNestedDisplayElement	ptClient: A type of System.Drawing Point object that represents the mouse position in client coordinates.	An Element object that represents the underlying display element.

### Example

Below is an example that demonstrates how to use PointToTableCellStyle method to retrieve the cell style information. This example handles a MouseMove handler of the Grid Table Control, retrieves the cell content using the above given method and then writes the content to a listbox control.

- Setup a Grouping Grid and bind it to a dataset. Handle TableControl.MouseMove event to let the user get the cell style information printed while hovering the mouse over the grid cells. Once you have the style, you can check Style.TableCellIdentity for information about the cell such as its column, underlying record, parent table, and so on.

[C#]

```
private void TableControl_MouseMove(object sender, MouseEventArgs e)
{
    Point ptClient = new Point(e.X, e.Y);
    GridTableControl tableControl = this.groupingGrid1.TableControl;

    GridTableCellStyleInfo style =
tableControl.PointToTableCellStyle(ptClient);
    Element displayElement = style.TableCellIdentity.DisplayElement;
    string info = "";

    if (style != null)
    {
        if (style.TableCellIdentity.Column != null)
            info = "Field Name - "+style.TableCellIdentity.Column.Name
+ ", Field Value - \" + style.CellValue.ToString() + "\", Field Type -
```

```
" + style.CellType.ToString();
    else
        info = style.TableCellIdentity.ToString();
    }

listBox1.Items.Clear();
listBox1.Items.Add("MousePosition: " + ptClient.ToString());
listBox1.Items.Add("Category Keys: " +
displayElement.ParentChildTable.CategoriesToString());
listBox1.Items.Add("Display Element Type: " +
displayElement.GetType().Name);
listBox1.Items.Add("Cell Information: " + info);
}
```

**[VB.NET]**

```
Private Sub TableControl_MouseMove(ByVal sender As Object, ByVal e As
MouseEventArgs)
    Dim ptClient As New Point(e.X, e.Y)
    Dim tableControl As GridTableControl = Me.groupingGrid1.TableControl

    Dim style As GridTableCellStyleInfo =
    tableControl.PointToTableCellStyle(ptClient)
    Dim displayElement As Element =
    style.TableCellIdentity.DisplayElement
    Dim info As String = ""

    If Not (style Is Nothing) Then
        If Not (style.TableCellIdentity.Column Is Nothing) Then
            info = "Field Name - " & style.TableCellIdentity.Column.Name &
", Field Value - """ & styleCellValue.ToString() & """", Field
Type - " & style.CellType.ToString()
        Else
            info = style.TableCellIdentity.ToString()
        End If
    End If
    listBox1.Items.Clear()
    listBox1.Items.Add("MousePosition: " & ptClient.ToString())
    listBox1.Items.Add("Category Keys: " &
displayElement.ParentChildTable.CategoriesToString())
    listBox1.Items.Add("Display Element Type: " &
displayElement.GetType().Name)
    listBox1.Items.Add("Cell Information: " & info)
End Sub
```

2. Here is a sample output.

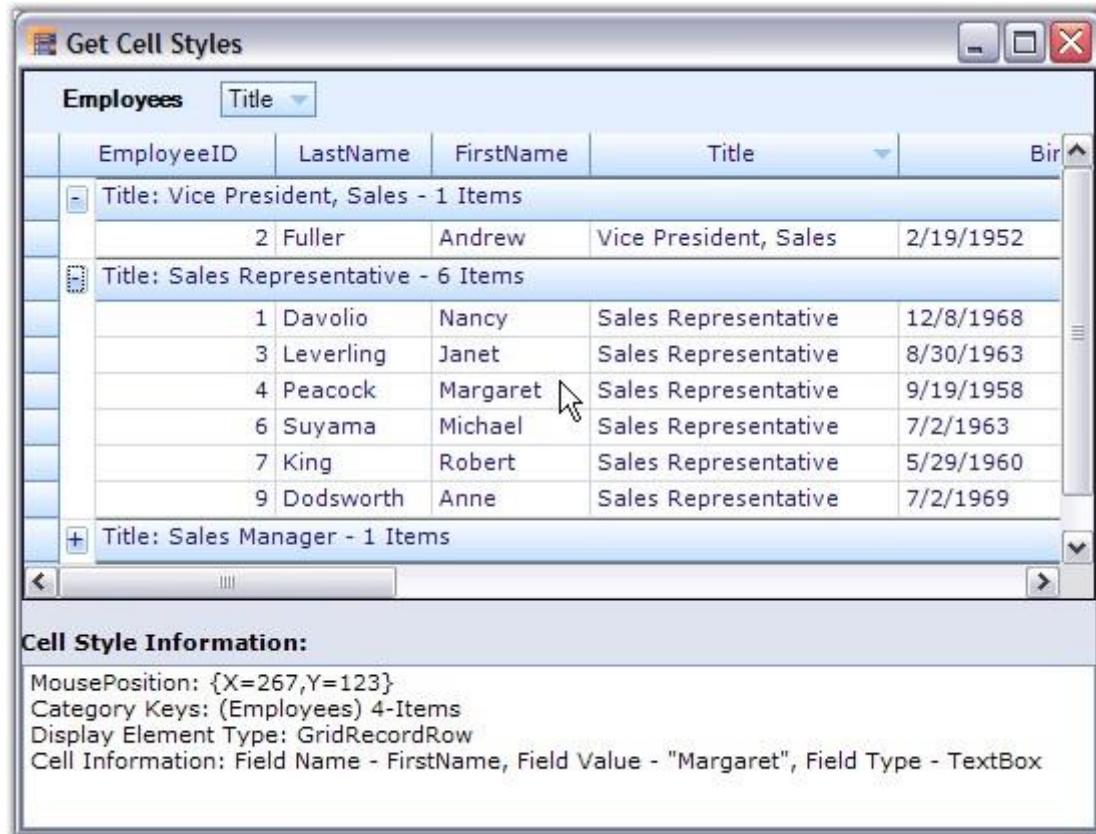


Figure 343: Retrieving Cell Style Information by using the PointToTableCellStyle Method

#### 4.3.4.5.8 Look and Feel

GridGroupingControl implements **Themes** and **VisualStyles** that set up a common Look and Feel to all the components in the grid. The term 'Look and Feel' refers not only the way the grid elements appear but also the way they behave in response to the user interactions like hovering mouse over them, clicking, and so on. Grid has in-built support for the following skins: **WindowsXP**, **Office2007 (Blue/Black/Silver)** and **Office2003**.

#### ThemesEnabled

Grouping Grid enables as well as disables XP themes via **ThemesEnabled** property. When it is set to true, XP themes are enabled.

```
[C#]
```

```
this.gridGroupingControl1.ThemesEnabled = true;
```

[VB.NET]

```
Me.gridGroupingControl1.ThemesEnabled = True
```



Figure 344: XP Themes for Grid Grouping Control

### GridVisualStyles

GridVisualStyles property is used to set different VisualStyles (skins) for grid like Office2007 and Office2003. Every component that is incorporated into the grid will be affected by these visual styles.

GridVisualStyles enumeration defines the skins exposed by the grouping grid. They are Office2003, Office2007Blue,

Office2007Black, Office2007Silver and SystemTheme. Default is SystemTheme.

Visual Styles can be set by assigning a GridVisualStyles enumeration value to the TableOptions.GridVisualStyles property.

[C#]

```
this.gridGroupingControl1.TableOptions.GridVisualStyles =
GridVisualStyles.Office2007Blue;
```

[VB.NET]

```
Me.gridGroupingControll.TableOptions.GridVisualStyles =  
GridVisualStyles.Office2007Blue
```

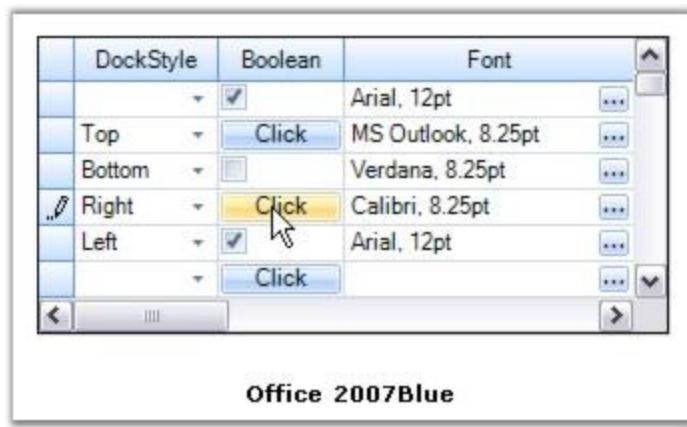


Figure 345: Grid Grouping Control with Office 2007 Blue Visual Style

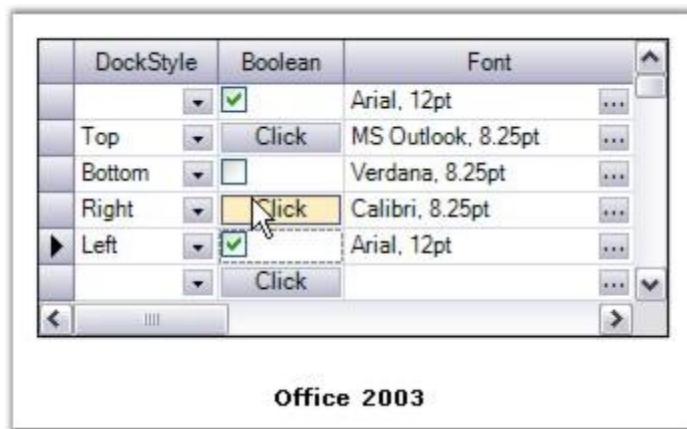


Figure 346: Grid Grouping Control with Office 2003 Visual Style

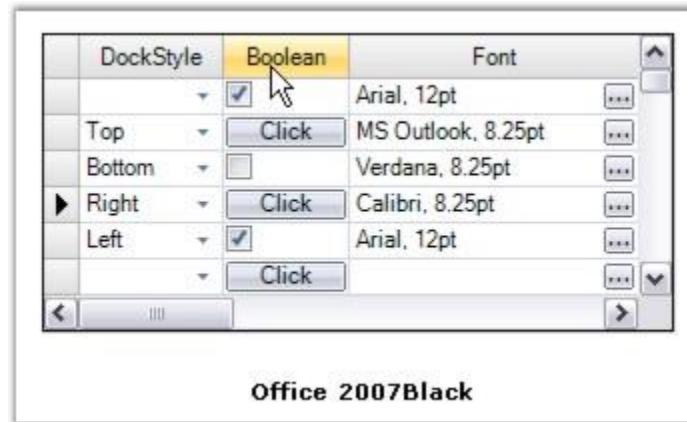


Figure 347: Grid Grouping Control with Office 2007 Black Visual Style



Figure 348: Grid Grouping Control with Office 2007 Silver Visual Style

Since visual styles also affect how the cells behave, the cell controls are painted with a different gradient when users interact with them either by clicking or by hovering the mouse over them. Below is the image that exposes these cases.

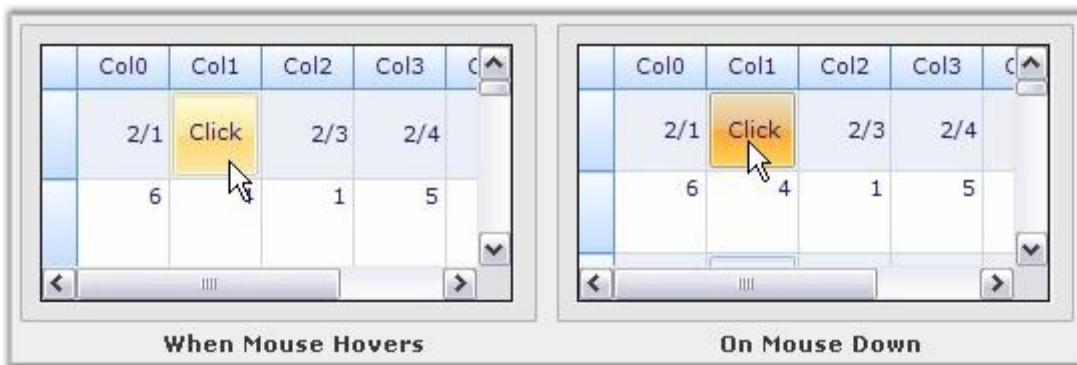


Figure 349: Highlighting Cells in the Grid when the Mouse Pointer moves or rests over the Cells

## Grid Skins

GridSkins, an extension of GridVisualStyles, is built on the idea of providing more advanced themes for your grid, along with the basic themes defined by GridVisualStyles. It is available as an add-on feature in the GridHelperClasses library.

GridSkins depicts the custom skin of GridVisualStyles. Currently it comes with new Vista skin that makes your grid components appear in vista-like look and feel.

Grid Skins can be set by invoking the static method ApplySkin of the GridSkins helper class. This method accepts two parameters, a grid that needs to be styled and a Skins enumeration value that specifies the desired skin, and applies this desired skin to all the grid components.

### [C#]

```
GridSkins.ApplySkin(this.gridControl1.Model, Skins.Vista);
```

### [VB .NET]

```
GridSkins.ApplySkin(Me.gridControl1.Model, Skins.Vista)
```

Below image illustrates a sample output.



Figure 350: Grid Grouping Control with Vista Skin

### 4.3.4.5.9 Table Options

**TableOptions** lets you set various properties that will affect the look and behavior of a grouping grid across all groups and child groups. Properties such as the default height of a Caption Row, GroupHeader / Footer, PreviewRow or whether TreeLines are visible between the PlusMinus cells are controlled by this property. Here is a screen shot that shows the list of properties you can set under TableOptions.

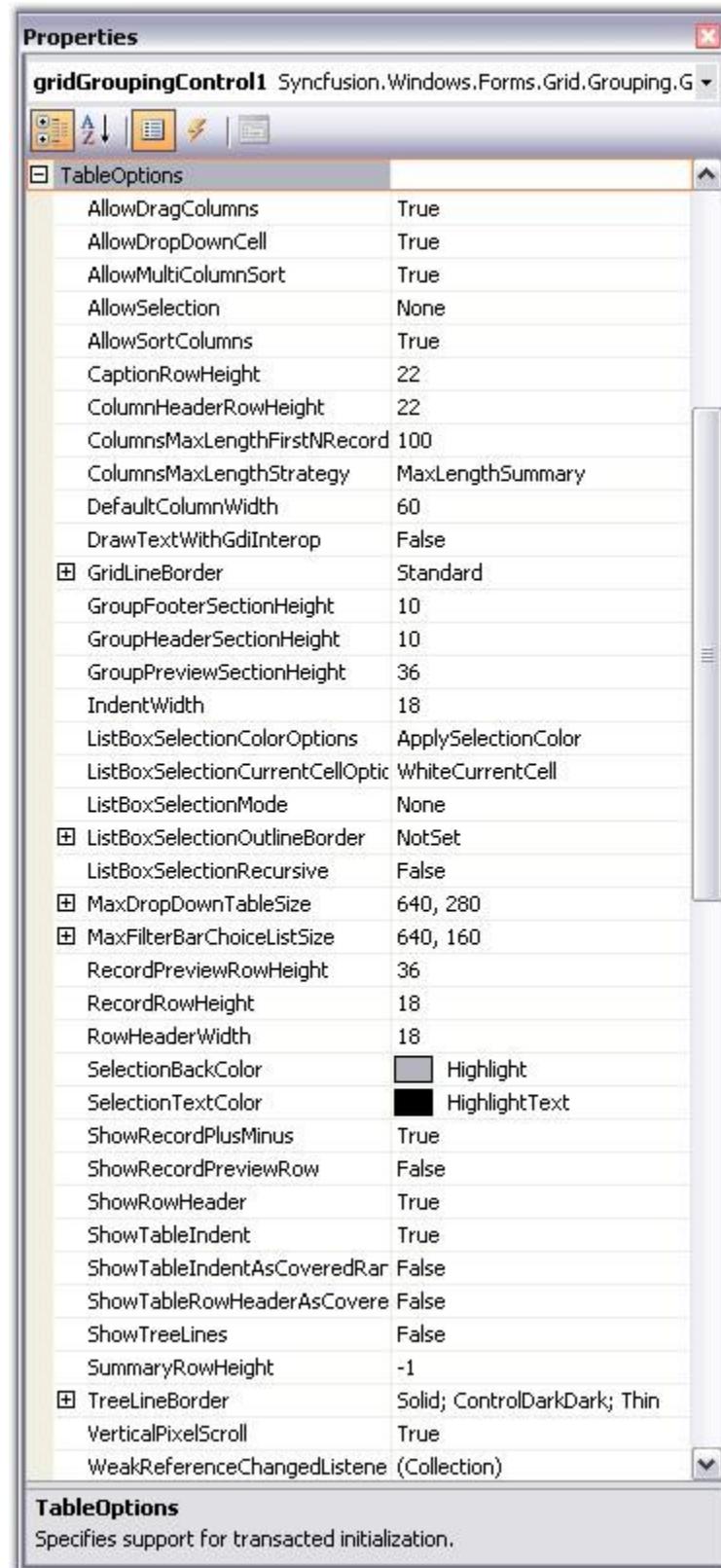


Figure 351: TableOptions Properties

Below table gives a brief description on the above properties.

TableOptions Property	Description
AllowDragColumns	Allows the user to re-arrange columns by dragging the headers.
AllowDropDownCell	Decides whether combo boxes are displayed for foreign key reference columns.
AllowSelection	Defines the selection behavior. Set it to none to use record selections.
AllowSortColumns	Allows user to sort the columns by clicking the column header.
AllowMultiColumnSort	Allows user to sort the table by multiple columns.
CaptionRowHeight	Displays height of caption rows in pixels.
ColumnHeaderRowHeight	Displays height of column header rows in pixels.
ColumnsMaxLengthFirstNRecords	Number of rows to be evaluated for GridColumnsMaxLengthStrategy.FirstNRecords.
ColumnsMaxLengthStrategy	Defines strategy for re sizing columns to optimal width.
DefaultColumnWidth	Default width for columns.
DrawTextWithGdiInterop	Specifies whether the text should be drawn using GDI interop routines.
GridLineBorder	Controls the style of the lines used to draw grid lines.
GridVisualStyles	Specifies the skin for the grid.
GroupFooterSectionHeight	Displays height of group footers in pixels.
GroupHeaderSectionHeight	Displays height of group headers in pixels.
GroupPreviewSectionHeight	Displays height of group previews in pixels.

IndentWidth	Displays width of the indentation of each child group in pixels.
ListBoxSelectionColor Option	Controls the appearance of selected cells.
ListBoxSelectionCurrentCell Options	Controls the appearance and behavior of the current cell when the ListBoxSelectionMode is set.
ListBoxSelectionMode	Enables list box-type selection behavior when the user moves the current cell.
MaxDropDownTableSize	Maximum size for the dropdown table associated with foreign keys.
RecordPreviewRowHeight	Displays height of the record previews in pixels.
RecordRowHeight	Displays height of the record rows in pixels.
RowHeaderWidth	Displays width of the row header cells in pixels.
SelectionBackColor	Sets background color for selected records.
SelectionTextColor	Sets text color for selected records.
ShowRecordPlusMinus	Indicates whether a PlusMinus cell should appear next to the records; only applicable for nested tables.
ShowRecordPreviewRow	Indicates whether a nested table has a preview row; only applicable for nested tables.
ShowRowHeader	Indicates whether the row header column should be visible.
ShowTableIndent	Indicates whether children of the records in the parent table should be indented; only applicable for nested tables.
ShowTableIndentAsCoveredRange	Indicates whether the cells in a particular indent level are treated as a single covered cell; only applicable for nested tables.
ShowTableRowHeaderAsCoveredRange	Indicates whether the row header cells for a particular nested table are treated as a single covered cell; only applicable for nested tables.

ShowTreeLines	Indicates whether the PlusMinus cells are shown connected with lines.
SummaryRowHeight	Height in pixels of the summary rows. The value -1 is a special setting to indicate that the summary row height should always be the same as the RecordRowHeight.
TreeLineBorder	Controls the style of the line that is used to draw the tree lines.

In the Preview, try various property settings to see their effect on the display. Below is a sample of what you might see. The properties changed here are CaptionRowHeight, ColumnHeaderRowHeight, GridLineBorder, GridVisualStyles, ListBoxSelectionMode, SelectionBackColor, SelectionTextColor and ShowTreeLines.

The screenshot shows the Syncfusion Windows Forms Grid Grouping Control in a Visual Studio environment. The main window displays a grid of football statistics with the following columns: ID, losses, School, Sport, ties, wins, and year. The data is grouped by Sport (Football) and further grouped by wins (0-3, 1-2, 2-8, 3-14). The properties window on the right shows the following settings:

TableOptions	Value
ShowTableIndent	= True
AllowDragColumns	True
AllowDropDownCell	True
AllowMultiColumnSort	True
AllowSelection	None
AllowSortColumns	True
CaptionRowHeight	20
ColumnHeaderRowHeight	22
ColumnsMaxLengthF	100
ColumnsMaxLengthS	MaxLengthSummary
DefaultColumnWidth	60
DrawTextWithGdiInt	False
GridLineBorder	Solid; 208, 215, 229, Th
GroupFooterSectionH	10
GroupHeaderSectionH	10
GroupPreviewSectionH	38
IndentWidth	19
ListBoxSelectionColor	ApplySelectionColor
ListBoxSelectionCurr	HideCurrentCell
ListBoxSelectionMod	MultiExtended
ListBoxSelectionOutl	NotSet
ListBoxSelectionRec	False
MaxDropDownTableH	640, 280
MaxFilterBarChoiceL	640, 160
RecordPreviewRowH	38
RecordRowHeight	19
RowHeaderWidth	19
SelectionBackColor	255, 128, 128
SelectionTextColor	Maroon
ShowRecordPlusMin	True
ShowRecordPreview	False
ShowRowHeader	False
ShowTableIndent	True
ShowTableIndentAsC	False
ShowTableRowHead	False
ShowTreeLines	True

A tooltip at the bottom left indicates the current focus is on the `Syncfusion.Windows.Forms.Grid.Grouping.GridTableOptionsStyleInfo TableOptions`.

Figure 352: Customized Groups and Child Groups of Grid Grouping Control



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Grouping.Windows\Samples\2.0\Grouping Grid Options\Table Options Demo

#### 4.3.4.6 Grid Layout

Grouping grid offers different layouts to organize the data display. With default layout, grouping grid combines the column headers into a single row docked to the top. Below the column headers are the rows of data records displayed with one record per row and only one record field per column.

This default arrangement can be modified to customize the data views. This section discusses two features offered by the grouping grid in this regard.

##### 4.3.4.6.1 Stacked MultiHeaders

Essential Grouping Grid Control offers built support for Stacked MultiHeaders. This feature allows you to create additional unbound header rows called **StackedHeaderRows** that span across visible grid columns. You can group some columns under each header row. It also supports Drag / Drop of these header rows. Grouped columns will also be rearranged along with the header.

##### The StackedHeaderRows Collection

Stacked Header rows for a given Grid Table are gathered under **TableDescriptor.StackedHeaderRows** collection. This contains the property definitions that controls the behavior and appearance of the Stacked Headers. A StackedHeaderRow collection can be viewed as a set of stacked header rows in which each header row contains a collection of stacked headers that span across multiple columns.

Every header in a Stacked Header Row is defined by a **GridStackedHeaderDescriptor**. All the headers for a given stacked header row is managed by **GridStackedHeaderRowDescriptor**. **GridStackedHeaderRowDescriptorCollection**, which is returned by **TableDescriptor.StackedHeaderRows** property, manages the collection of **GridStackedHeaderRowDescriptors** for a given table. It is the **GridStackedHeaderVisibleColumnDescriptor** that binds a Column or ColumnSet to the StackedHeaderCell.

The order in which the StackedHeaders will appear is determined by the VisibleColumns collection. When the layout of the GridStackedHeaderRow is calculated the grid will loop through the VisibleColumns collection and find the StackedHeader Descriptor that references the VisibleColumn. If the same StackedHeader references multiple neighboring VisibleColumns then the header for these columns will be drawn as one cell. If there are no visible columns specified for a StackedHeader, then it will span across all the visible columns similar to a Caption.

You could be able to rearrange the columns by dragging the stacked headers. While doing so, it is the Visible Columns collection that is being affected. Since the order of stacked headers is dependent on Visible Columns, the GridStackedHeaderCollections itself does not need to be modified.

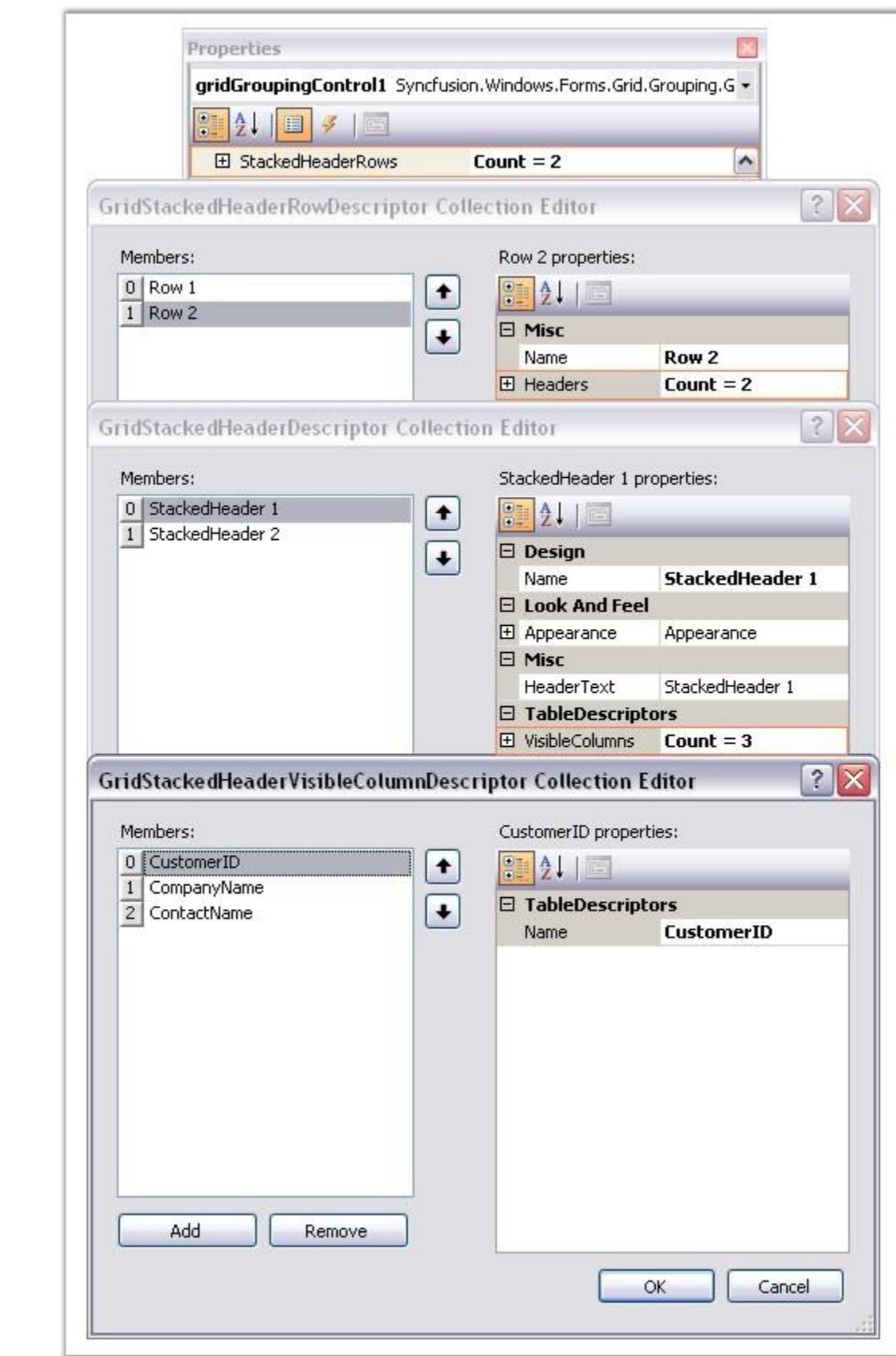
It is possible to add Stacked Headers for nested tables and groups too. You can enable the display of the StackedHeaders by setting the **ShowStackedHeaders** property to true.

- **TopLevelGroupOptions.ShowStackedHeaders** - Toggles the display of StackedHeaders for top most group.
- **ChildGroupOptions.ShowStackedHeaders** - Toggles the display of StackedHeaders for child groups.
- **NestedTableGroupOptions.ShowStackedHeaders** - Toggles the display of StackedHeaders for child table and its groups.

### Through Designer

Creating Stacked Headers is a two-step process.

1. As a first step, you must define the Stacked Header Rows by accessing the TableDescriptor.StackedHeaderRows property. This will open the GridStackedHeaderRowDescriptor Collection Editor wherein you can add as many header rows as you want, by specifying the different attributes like HeaderText, VisibleColumns, Appearance, and so on, for each header in the header row.



*Figure 353: Adding Stacked Headers by using the StackedHeaderRows Property*

### **Property Definitions**

<b>Property Name</b>	<b>Description</b>
Name	Specifies the name of the descriptor.
HeaderText	Specifies the text to be displayed in the StackedHeaderCell.
VisibleColumns	The collection of columns that should be combined under the Stacked Header.
Appearance	Controls the appearance of the StackedHeaderCell.

2. The second step is to enable the display of StackedHeaders for the given table and group by setting the ShowStackedHeaders property to true.

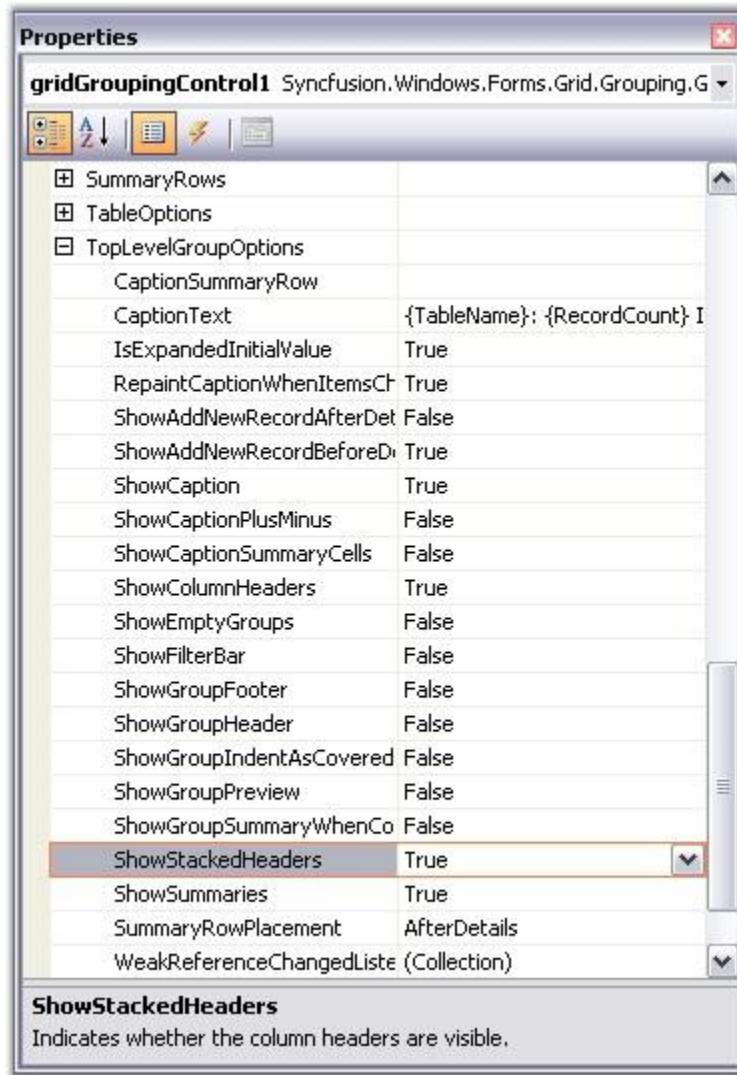
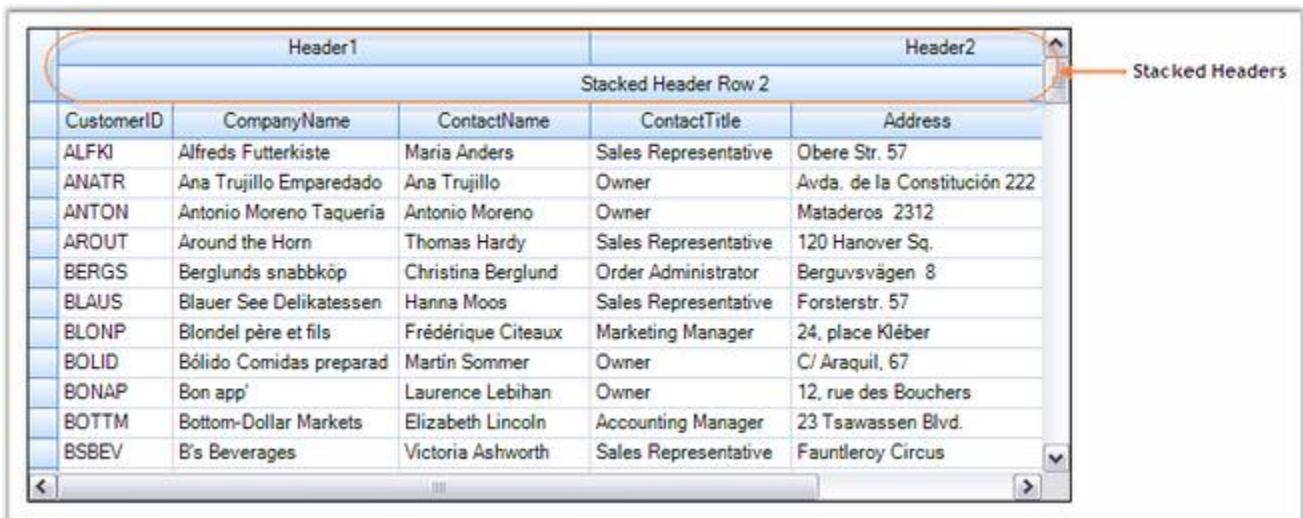


Figure 354: Setting ShowStackedHeaders property to True

## Output

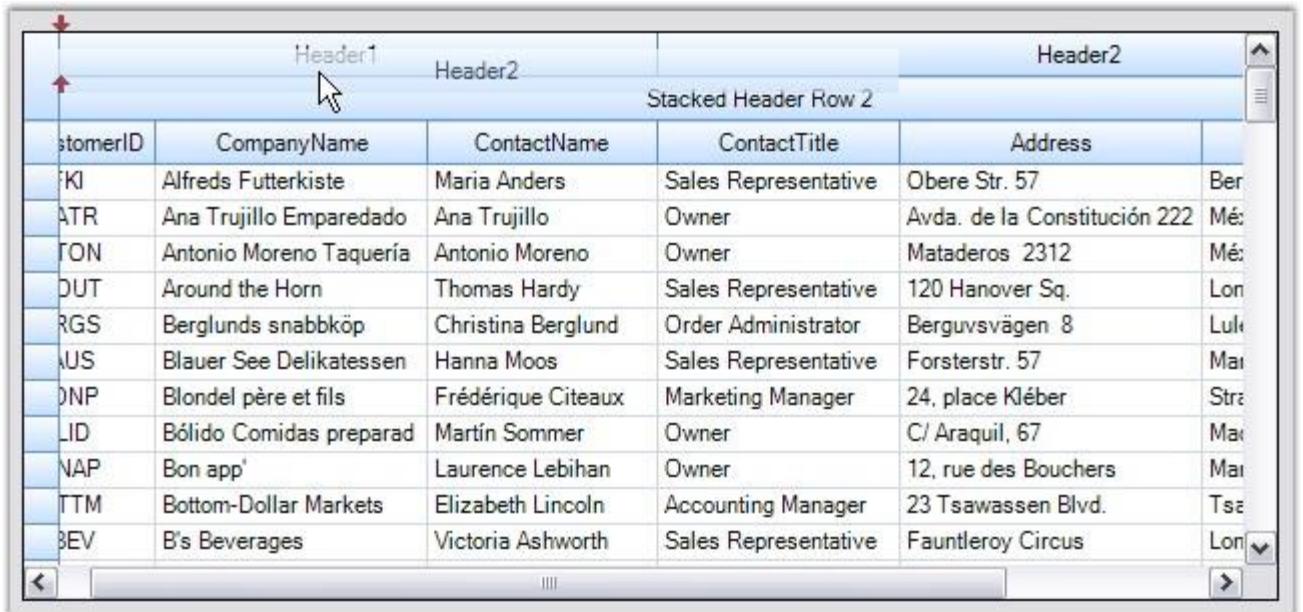
Here are the screen shots showing the Grouping Grid with two Stacked Header Rows. It illustrates the effect of doing Drag / Drop on StackedHeaders. You could notice that the order of the Visible columns gets affected automatically while rearranging the StackedHeaders.



The screenshot shows a Windows Form application window containing a grid control. The grid has three header rows. The first row contains two columns labeled "Header1" and "Header2". The second row, which is highlighted with a blue background, contains one column labeled "Stacked Header Row 2". The third row contains five columns labeled "CustomerID", "CompanyName", "ContactName", "ContactTitle", and "Address". The data rows below follow this structure. A red arrow points from the text "Stacked Headers" to the right edge of the "Header2" column header, indicating that dragging it will move all three header rows together.

Header1		Header2		
Stacked Header Row 2				
CustomerID	CompanyName	ContactName	ContactTitle	Address
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57
ANATR	Ana Trujillo Emparedado	Ana Trujillo	Owner	Avda. de la Constitución 222
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57
BLONP	Blondel père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber
BOLID	Bólido Comidas preparad	Martín Sommer	Owner	C/ Araquil, 67
BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Bouchers
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd.
BSBEV	B's Beverages	Victoria Ashworth	Sales Representative	Fauntleroy Circus

Figure 355: Grouping Grid with StackedHeaders



The screenshot shows the same grid interface as Figure 355, but with the "Header2" column header being dragged. A cursor arrow is positioned over the "Header2" header, and a red vertical arrow indicates the direction of movement. The "Stacked Header Row 2" row is visible above the data rows. The grid's scroll bars are visible on the right and bottom edges.

Header1		Header2	Header2		
Stacked Header Row 2					
CustomerID	CompanyName	ContactName	ContactTitle	Address	
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Ber
ANATR	Ana Trujillo Emparedado	Ana Trujillo	Owner	Avda. de la Constitución 222	Mé
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	Mé
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	Lon
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luk
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Ma
BLONP	Blondel père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Stra
BOLID	Bólido Comidas preparad	Martín Sommer	Owner	C/ Araquil, 67	Ma
BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Bouchers	Ma
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd.	Tsa
BSBEV	B's Beverages	Victoria Ashworth	Sales Representative	Fauntleroy Circus	Lon

Figure 356: Rearranging StackedHeaders by dragging and dropping Header2

	Header2				Header1
	Stacked Header Row 2				
ContactTitle	Address	City	Country	CustomerID	CompanyName
Sales Representative	Obere Str. 57	Berlin	Germany	ALFKI	Alfreds Futterkiste
Owner	Avda. de la Constitución 222	México D.F.	Mexico	ANATR	Ana Trujillo Emparedado
Owner	Mataderos 2312	México D.F.	Mexico	ANTON	Antonio Moreno Taquería
Sales Representative	120 Hanover Sq.	London	UK	AROUT	Around the Horn
Order Administrator	Berguvsvägen 8	Luleå	Sweden	BERGS	Berglunds snabbköp
Sales Representative	Forsterstr. 57	Mannheim	Germany	BLAUS	Blauer See Delikatessen
Marketing Manager	24, place Kléber	Strasbourg	France	BLONP	Blondel père et fils
Owner	C/ Araquil, 67	Madrid	Spain	BOLID	Bólido Comidas preparad
Owner	12, rue des Bouchers	Marseille	France	BONAP	Bon app'
Accounting Manager	23 Tsawassen Blvd.	Tsawassen	Canada	BOTTM	Bottom-Dollar Markets
Sales Representative	Fauntleroy Circus	London	UK	BSBEV	B's Beverages

Figure 357: VisibleColumns rearranged as a result of the above Drag / Drop

### StackedHeaders for NestedGroups

Stacked Headers can be enabled for Child Group by setting ChildGroupOptions.ShowStackedHeaders to true. The grouping grid in the below image displays the stacked headers for the nested groups.

Header1					Header2
Stacked Header Row 2					
CustomerID	CompanyName	ContactName	ContactTitle	Address	
ContactTitle: Accounting Manager - 10 Items					
Header1					Header2
Stacked Header Row 2					
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd.	
FISSA	FISSA Fabrica Inter. Salchichas S.A.	Diego Roel	Accounting Manager	C/ Moralzarzal, 86	
HANAR	Hanari Carnes	Mario Pontes	Accounting Manager	Rua do Paço, 67	
LILAS	LILA-Supermercado	Carlos González	Accounting Manager	Carrera 52 con Ave. Bolívar	
QUEDE	Que Delicia	Bernardo Batista	Accounting Manager	Rua da Panificadora, 12	
QUICK	QUICK-Stop	Horst Kloss	Accounting Manager	Taucherstraße 10	

Figure 358: Stacked Headers for Nested Groups

### Appearance

A couple of ways are there to control the appearance of the StackedHeaders. By one way you can access Appearance.StackedHeaderCell property to enter the appearance definitions. Appearance set this way will be applied to all the stacked header cells. An alternate way is to specify the appearance settings through the GridStackedHeaderRow Descriptor. In this way, you can have different settings for individual stacked headers in each StackedHeaderRow.

Here is the property window with GridStackedHeaderRowDescriptor Collection Editor showing the appearance settings of Stacked Headers defined.

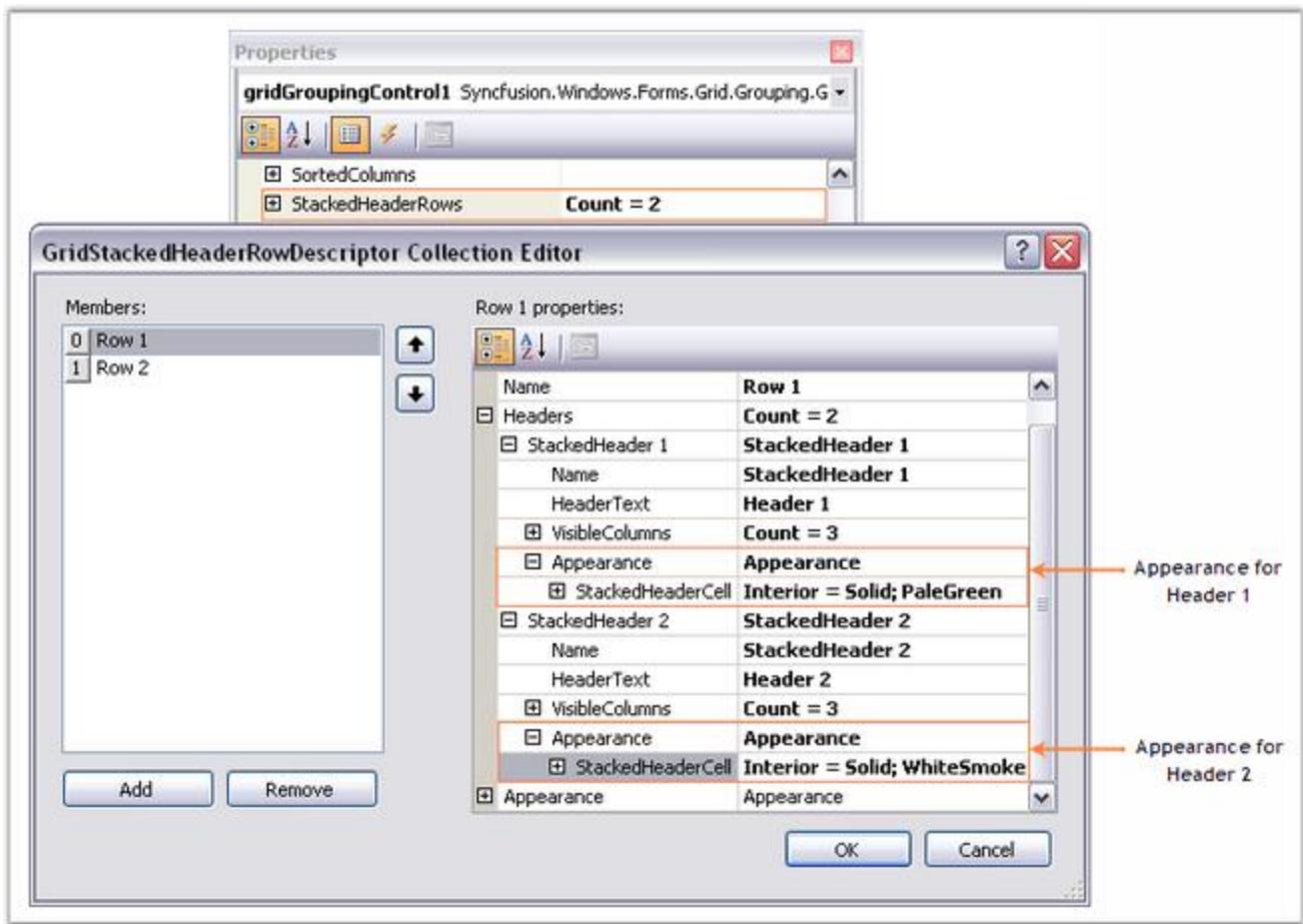


Figure 359: Appearance Settings for Stacked Headers

## Output

Here is the effect of the above settings.



The screenshot shows a Windows Form application window containing a grid control. The grid has a header row with two columns: 'Header1' and 'Header2'. Below this is a 'Stacked Header Row 2' which spans both columns. The main data area contains 10 rows of customer information. The columns are labeled: CustomerID, CompanyName, ContactName, ContactTitle, Address, and City. The data rows are as follows:

CustomerID	CompanyName	ContactName	ContactTitle	Address	City
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin
ANATR	Ana Trujillo Emparedados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	México D.F.
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim
BLONP	Blondel père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg
BOLID	Bólido Comidas preparad	Martín Sommer	Owner	C/ Araquil, 67	Madrid
BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Bouchers	Marseille
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd.	Tsawassen
BSBEV	B's Beverages	Victoria Ashworth	Sales Representative	Fauntleroy Circus	London

Figure 360: Stacked Headers created for the Grid Grouping Control

## Programmatically

You can add the Stacked Header Rows at run time too. To achieve this, first you must define a required number of GridStackedHeaderDescriptors by specifying the VisibleColumns for each. Next, create a StackedHeaderRow by instantiating GridStackedHeaderRowDescriptor and add the above defined stacked headers into it. Finally, add this header row into the TableDescriptor.StackedHeaderRows collection. The following code example illustrates this process.

[C#]

```
GridStackedHeaderDescriptor shd = new
GridStackedHeaderDescriptor("header1", "StackedHeader1");
shd.VisibleColumns.Add(new
GridStackedHeaderVisibleColumnDescriptor("CustomerName"));
shd.VisibleColumns.Add(new
GridStackedHeaderVisibleColumnDescriptor("CompanyName"));
GridStackedHeaderRowDescriptor shrd = new
GridStackedHeaderRowDescriptor("Row1",
new GridStackedHeaderDescriptor[] { shd });
this.gridGroupingControl1.TableDescriptor.StackedHeaderRows.Add(shrd);

// Customize the Appearance.
this.gridGroupingControl1.Appearance.StackedHeaderCell.BackColor =
Color.Teal;
```

[VB .NET]

```
Dim shd As GridStackedHeaderDescriptor = New  
GridStackedHeaderDescriptor("header1", "StackedHeader1")  
shd.VisibleColumns.Add(New  
GridStackedHeaderVisibleColumnDescriptor("CustomerName"))  
shd.VisibleColumns.Add(New  
GridStackedHeaderVisibleColumnDescriptor("CompanyName"))  
Dim shrd As GridStackedHeaderRowDescriptor = New  
GridStackedHeaderRowDescriptor("Row1",  
New GridStackedHeaderDescriptor() { shd })  
Me.gridGroupingControll.TableDescriptor.StackedHeaderRows.Add(shrd)  
  
' Customize the Appearance.  
Me.gridGroupingControll.Appearance.StackedHeaderCell.BackColor =  
Color.Teal
```



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\Version  
Number]\Windows\Grid.Grouping.Windows\Samples\2.0\Grouping Grid Layout\Stacked Multi  
Headers Demo

#### 4.3.4.6.2 Multi Row Record

Grid Grouping control offers built-in support for MultiRowRecords. This feature allows the records to span across multiple rows and columns. It is achieved through the property **TableDescriptor.ColumnSets**. It allows you to modify the default alignment of the visible columns.

##### The ColumnSets Collection

ColumnSets acts as a superset of TableDescriptor.Columns collection. Once the ColumnSets are defined, the grouping grid will then loop through the collection and organize the data display accordingly. Each ColumnSet is defined by a **GridColumnSetDescriptor**. ColumnSets are managed by **GridColumnSetDescriptorCollection** that is returned by the TableDescriptor.ColumnSets property.

##### Programmatically

Follow the steps below to span the records across multiple rows.

1. Define a GridColumnSpanDescriptor for each column to be spanned across grid rows or columns. Specify the range that the column spans. Rows and Columns are zero-based.

**[C#]**

```
GridColumnSpanDescriptor csd0 = new  
GridColumnSpanDescriptor("EmployeeID");  
csd0.Range = GridRangeInfo.Cells(0, 0, 1, 0);  
GridColumnSpanDescriptor csd1 = new  
GridColumnSpanDescriptor("Address");  
csd1.Range = GridRangeInfo.Cells(0, 1, 0, 2);  
GridColumnSpanDescriptor csd2 = new GridColumnSpanDescriptor("City");  
csd2.Range = GridRangeInfo.Cells(1, 1, 1, 1);  
GridColumnSpanDescriptor csd3 = new  
GridColumnSpanDescriptor("Country");  
csd3.Range = GridRangeInfo.Cells(1, 2, 1, 2);
```

**[VB .NET]**

```
Dim csd0 As GridColumnSpanDescriptor = New  
GridColumnSpanDescriptor("EmployeeID")  
csd0.Range = GridRangeInfo.Cells(0, 0, 1, 0)  
Dim csd1 As GridColumnSpanDescriptor = New  
GridColumnSpanDescriptor("Address")  
csd1.Range = GridRangeInfo.Cells(0, 1, 0, 2)  
Dim csd2 As GridColumnSpanDescriptor = New  
GridColumnSpanDescriptor("City")  
csd2.Range = GridRangeInfo.Cells(1, 1, 1, 1)  
Dim csd3 As GridColumnSpanDescriptor = New  
GridColumnSpanDescriptor("Country")  
csd3.Range = GridRangeInfo.Cells(1, 2, 1, 2)
```

2. Create a GridColumnSetDescriptor whose ColumnSpans property stores the information about the columns that need to be spanned. Hence you need to initialize the ColumnSpans property with the Columns (the ColumnSpanDescriptors of the desired columns) you want to spread.

**[C#]**

```
GridColumnSetDescriptor csd = new GridColumnSetDescriptor();  
csd.ColumnSpans.Add(csd0);  
csd.ColumnSpans.Add(csd1);  
csd.ColumnSpans.Add(csd2);  
csd.ColumnSpans.Add(csd3);
```

**[VB .NET]**

```
Dim csd As GridColumnSetDescriptor = New GridColumnSetDescriptor()
csd.ColumnSpans.Add(csd0)
csd.ColumnSpans.Add(csd1)
csd.ColumnSpans.Add(csd2)
csd.ColumnSpans.Add(csd3)
```

3. Finally bind this ColumnSet to the grid by adding the above created GridColumnSetDescriptor into the TableDescriptor.ColumnSets property.

**[C#]**

```
this.gridGroupingControl1.TableDescriptor.ColumnSets.Add(csd);
```

**[VB .NET]**

```
Me.gridGroupingControl1.TableDescriptor.ColumnSets.Add(csd)
```

4. Here is a sample output.

EmployeeID	Address	
	City	Country
1	507 - 20th Ave. E.	
	Seattle	USA
2	908 W. Capital Way	
	Tacoma	USA
3	722 Moss Bay Blvd.	
	Kirkland	USA
4	4110 Old Redmond R	
	Redmond	USA
5	14 Garrett Hill	
	London	UK
6	Coventry House	
	London	UK

*Figure 361: Spanning Records across Multiple Rows*

### Through Designer

To create ColumnSets that defines the ColumnSpans for a grid, select the TableDescriptor.ColumnSets property in the property window. This will open the GridColumnSetDescriptor Collection Editor that will let you specify the columns to span and the range, for each of the columns.

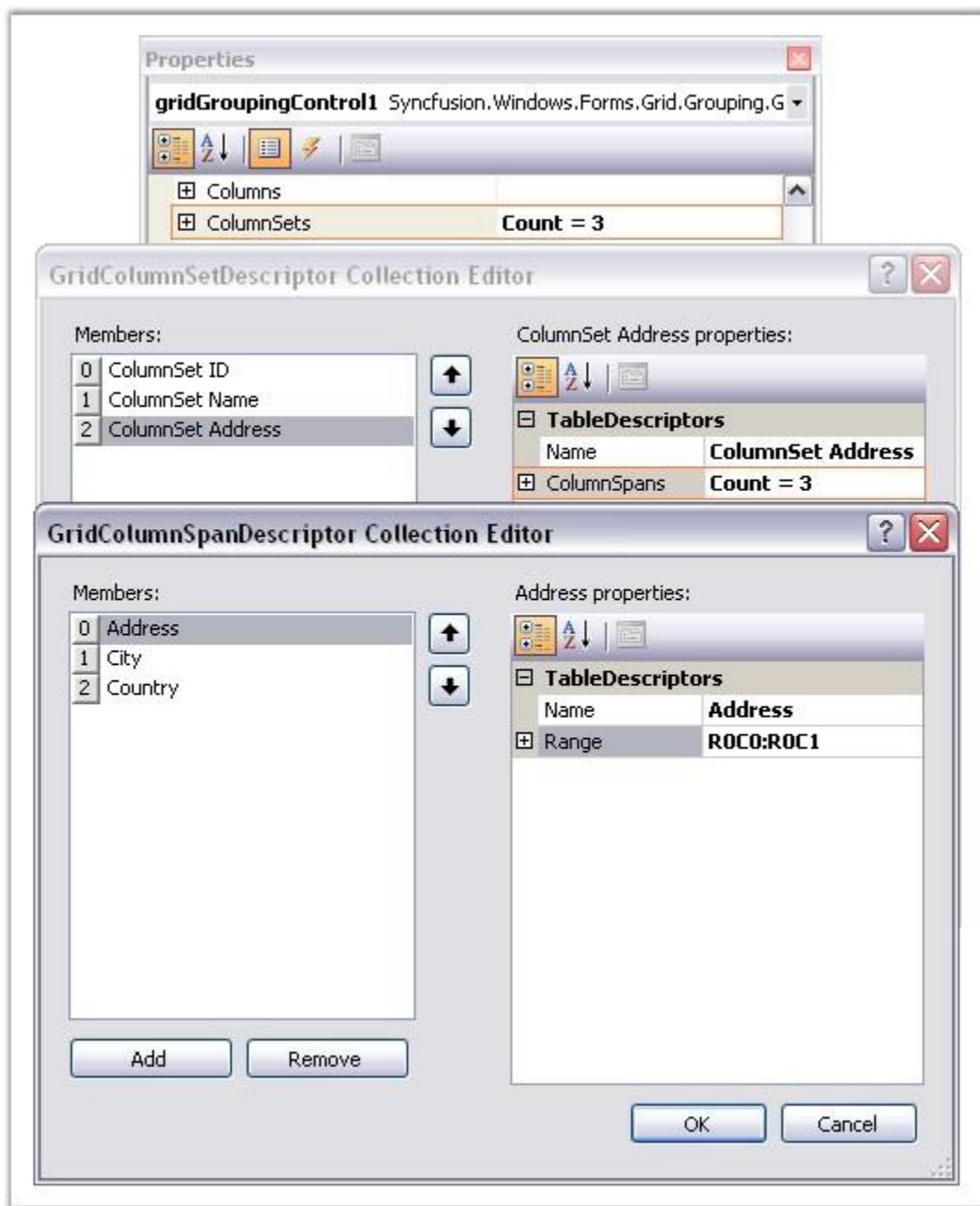


Figure 362: Creating Column Sets to Span Columns in the Grid

 **Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Grouping Grid Layout\Employee View Demo

#### 4.3.4.6.3 Grid Field Chooser

The view of a grid can be customized based on column visibility by using a plug-in utility called Field Chooser. A Field Chooser can be associated to Grid Grouping control to add or remove columns from a grid. It can be done by initializing the FieldChooser class where the constructor takes a parameter as a Grid Grouping control object.

Enabling the Field Chooser allows the user to right-click on a column header and select Field Chooser menu item to view the Field Chooser dialog. This dialog would list all the column names with check boxes beside them. The required columns can be made visible in the grid by selecting the check box adjacent to the required column.

The following code example illustrates the usage of Field Chooser.

[C#]

```
FieldChooser fchooser = new FieldChooser(this.gridGroupingControl1);
```

[VB.NET]

```
Dim fchooser As FieldChooser = New  
FieldChooser(Me.gridGroupingControl1)
```

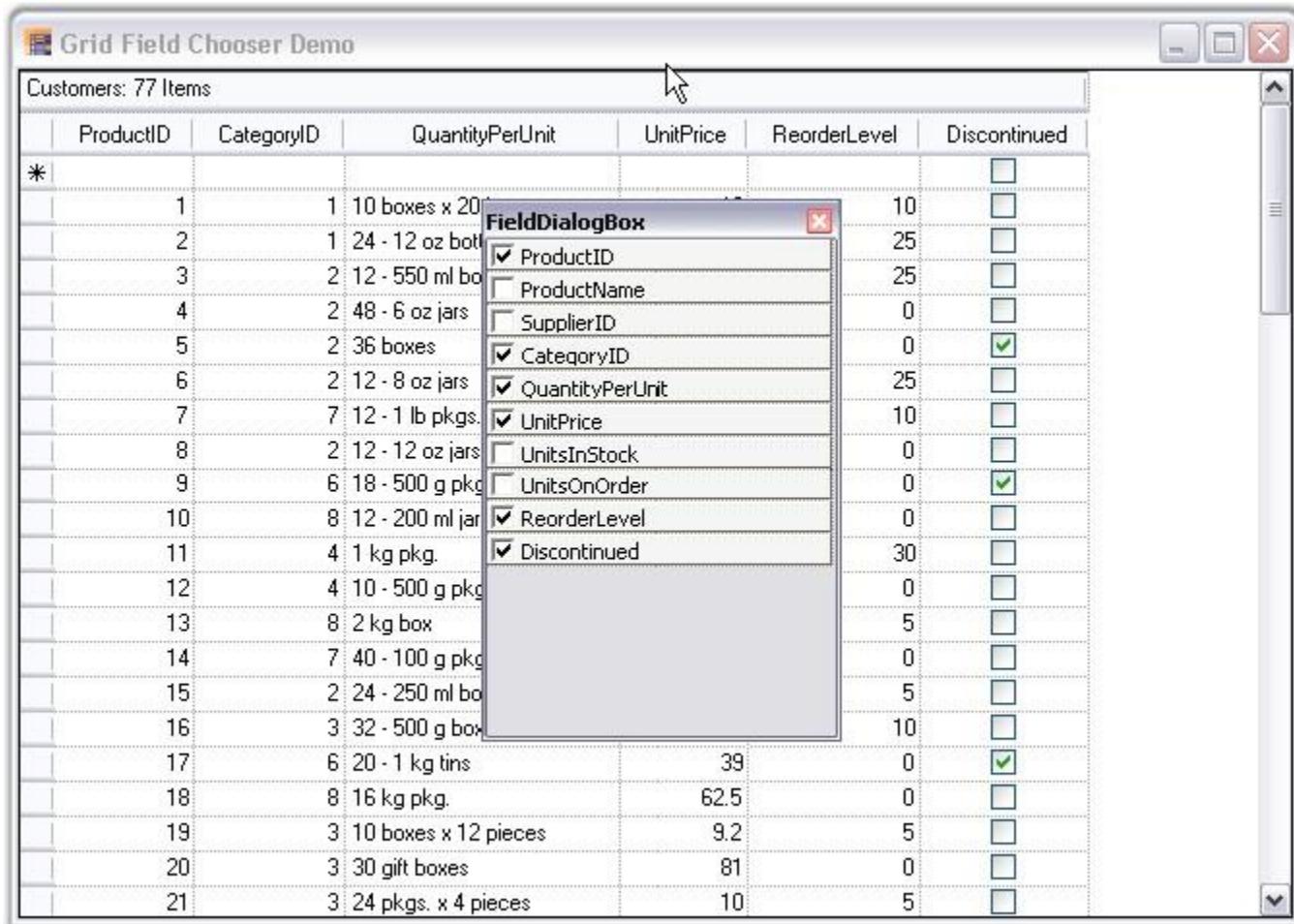


Figure 327: Grid Grouping control with Field Chooser

For more details, refer the following sample:

```
<Install Location>\Syncfusion\EssentialStudio\Version  
Number\Windows\Grid.Grouping.Windows\Samples\2.0\Grouping Grid Layout\Grid Field  
Chooser Demo
```

#### 4.3.4.6.3.1 Field Chooser Events

FieldChooser events are used to customize the FieldChooser dialog box. Field chooser events allow the user to modify the control of the FieldChooser dialog and change its caption name. It has events to perform operations in the FieldChooser dialog box such as **FieldChooserShowing**, **FieldChooserShown**, **FieldChooserClosing**, and **FieldChooserClosed** events.

#### FieldChooser Events Table

Event	Description	Arguments	Type
FieldChooserShowing	This event is handled before the FieldChooser DialogBox is shown either in the stacked header or column header. This event is generally used to change the caption of the DialogBox, get and set the control of the TreeviewAdv or Grid controls, and also used to cancel showing the FieldChooser dialog.	<code>public FieldChooserShowingE ventArgs(string caption, object fieldList);</code>	Event
FieldChooserShown	This event is handled after the FieldChooser dialog is shown. This event is generally used to get the caption name of the FieldChooser dialog box and get the control of the TreeviewAdv or Grid controls.	<code>public FieldChooserShownE ventArgs(string caption, object fieldList);</code>	Event
FieldChooserClosin g	This event is generally used to change the caption name of the dialog, get and set the control of the TreeviewAdv or Grid controls, and also used to cancel closing the FieldChooser dialog.	<code>public FieldChooserClosingE ventArgs(string caption, object fieldList);</code>	Event
FieldChooserClose d	This event is handled after the FieldChooser Dialog Box is closed. This event is generally used to get the caption name of the FieldChooser dialog, and get the control of the TreeviewAdv or Grid controls after closing the FieldChooser dialog.	<code>public FieldChooserClosedE ventArgs(string caption, object fieldList);</code>	Event

The following code sample illustrates handling the **FieldChooserShowing** event. It is used to change the caption of the FieldChooser dialog and cancel the FieldChooser dialog showing.

[C#]

```
// FieldChooserShowing Event
void gridGroupingControl1_FieldChooserShowing(object sender,
FieldChooserShowingEventArgs e)
{
    e.Caption = "Syncfusion";
    e.Cancel = true;
}
```

[VB.NET]

```
'FieldChooserShowing Event

Private Sub GroupingControl_FieldChooserShowing(ByVal sender As
Object, ByVal e As
Syncfusion.Windows.Forms.Grid.FieldChooserShowingEventArgs)
    e.Caption = "Syncfusion"
    e.Cancel = True
End Sub
```

The following code example illustrates handling the **FieldChooserShown** event. It is used to get the caption name of the FieldChooser dialog after showing the dialog box.

[C#]

```
// FieldChooserShown Event
void gridGroupingControll1_FieldChooserShown(object sender,
FieldChooserShownEventArgs e)
{
    string captionName = e.Caption;
    Console.WriteLine(captionName);
}
```

[VB.NET]

```
'FieldChooserShown Event

Private Sub GroupingControl_FieldChooserShown(ByVal sender As
Object, ByVal e As
Syncfusion.Windows.Forms.Grid.FieldChooserShownEventArgs)
    Dim captionName As String = e.Caption
    Console.WriteLine(captionName)

End Sub
```

The following code example illustrates handling the **FieldChooserClosing** event. Here it is used to change the caption name of the FieldChooser dialog and cancel closing it.

[C#]

```
// FieldChooserClosing Event
void gridGroupingControll1_FieldChooserClosing(object sender,
```

```
FieldChooserClosingEventArgs e)
{
    e.Caption = "Syncfusion Inc";
    e.Cancel = true;
}
```

**[VB.NET]**

```
'FieldChooserClosing Event
Private Sub GroupingControl_FieldChooserClosing(ByVal sender As
Object, ByVal e As
Syncfusion.Windows.Forms.Grid.FieldChooserClosingEventArgs)
    e.Caption = "Syncfusion Inc"
    e.Cancel = True
End Sub
```

The following code example illustrates handling the **FieldChooserClosed** event. Here it is used to get the caption name of the FieldChooser dialog after closing it.

**[C#]**

```
//FieldChooserClosed Event
void gridGroupingControll1_FieldChooserClosed(object sender,
FieldChooserClosedEventArgs e)
{
    string caption = e.Caption;
    Console.WriteLine(caption);
}
```

**[VB.NET]**

```
'FieldChooserClosed Event
Private Sub GroupingControl_FieldChooserClosed(ByVal sender As
Object, ByVal e As
Syncfusion.Windows.Forms.Grid.FieldChooserClosedEventArgs)
    Dim captionName As String = e.Caption.ToString()
    Console.WriteLine(captionName)
End Sub
```

#### 4.3.4.6.4 Field Chooser for Stacked Header

The GridGrouping control in Essential Grid provides field chooser support for stacked headers. The field chooser feature enables you to customize a column in a grid at run time without modifying the database it is bound to.

### Use Case Scenarios

When you want to show or hide the columns of a stacked header in a grid without deleting its bound records, you can achieve this using this feature.

### Properties

*Table 12: Property Table*

Property	Description	Type	Data Type
EnableColumnsInView	Used to enable or disable the column names in the field chooser dialog.	Property	Boolean

### Constructor

Constructor	Description	Parameters	Type	Return Type
FieldChooser	Used to wire the GridGroupingControl with the field chooser.	(<GridGroupingControl>)	Constructor	class

### Sample Link

A demo of this feature is available in the following location:

***..\\AppData\\Local\\Syncfusion\\EssentialStudio\\Installed  
Version}\\Windows\\Grid.Grouping.Windows\\Samples\\2.0\\Grouping Grid Layout\\Field  
Chooser in Stacked Header Demo***

### Adding Field Chooser Stacked Headers In GridGroupingControl

1. To add field chooser, create a constructor using the *FieldChooser* class and pass the *GridGroupingControl* as the parameter.

The following code illustrates this:

[C#]

```
// wire the GridGroupingControl with the field chooser.
FieldChooser fchooser = new
FieldChooser(this.gridGroupingControl1);
```

Here is the code snippet used to disable the `EnableColumnsInView` property.:

```
//disable the EnableColumnsInView property
fchooser.EnableColumnsInView = false;
```

**[VB]**

```
'Wire the GridGroupingControl with the field chooser.
Dim fchooser As FieldChooser = New
FieldChooser(Me.gridGroupingControl1)

'Disable the EnableColumnsInView property
fchooser.EnableColumnsInView = False
```

When the code runs, the entire grid will open.

2. Right click on a column header and select the **Field Chooser** menu item to view the **Field Chooser** dialog.



Figure 328: Field Chooser

3. This dialog will list all the column names with check boxes adjacent to them.



Figure 329: FieldDialogBox

4. Select the checkboxes of the columns you want to be displayed in the grid.
5. The grid will have only the columns which are selected in the Field Chooser **dialog**.

The screenshot shows a Windows application window titled "Field Chooser in Stacked Headers Demo". Inside, there is a grid control with the title "Orders" and a subtitle "Orders: 830 Items". The grid has four stacked headers: "Header One" (highlighted in orange), "Header Two" (yellow), "Header Three" (light blue), and "Header Four" (light green). The data rows show various order details like customer names, addresses, and ship regions. To the right of the grid, a "FieldTreeDialogBox" is open, displaying a hierarchical tree view of the columns. The "Header One" node is expanded, showing its children: OrderID, CustomerID, EmployeeID, and ShipAddress. The "Header Four" node is also expanded, showing its children: ShipRegion, ShipPostalCode, and ShipCountry. Other columns like Freight, RequiredDate, ShippedDate, ShipVia, ShipName, and ShipCity are listed under their respective header nodes.

Figure 330: Customized Grid

#### 4.3.4.7 Selections

There are two type of selection architectures in a Grid Grouping control. One is designed specifically for the Grid Grouping control referred as **Record-Based Selection** and the other is inherited from GridControlBase named as **Model-Based Selection**.

If you use the Record-Based selection functionality, then whole records are selected and these selections function properly with nested tables, sorting, and so on. If you choose to use the inherited selection capability, you will be able to select cell ranges, but the selections will have no knowledge of nested tables, grouping or sorts and thus is limited in a Grid Grouping control.

To use the Grid Grouping control record selections, you must set **AllowSelections** to None and then set **ListBoxSelectionMode** to something other than None. To use the inherited selection capability, set AllowSelections to something other than None.

In this section, you will learn about the following topics.

##### 4.3.4.7.1 Model Based Selection

**Model Based Selection** is cell-based that allows you to do a selection across the cell which is not possible with record-based selection. This derives from the GridControlBase and hence will not be aware of the grouping elements like nested tables, groups, and so on.

A model-based selection can be set by initializing **AllowSelection** property to a value other than None. The possible values for this type of selection is defined by the enum **GridSelectionFlags**. By setting the various flags in AllowSelection, you can control the selection behavior of the grouping grids.

##### Selection Flags

Flag Name	Description
AlphaBlend	Uses alpha blending to highlight selected cells.
Cell	Individual cells can be selected.

Column	Columns can be selected.
Row	Rows can be selected.
Table	Whole table can be selected.
Shift	Allows user to extend the existing selection by holding the Shift key and clicking a cell.
MixRangeType	Allows you to select multiple ranges by holding the CTRL key.
Multiple	Allows both rows and columns to be selected at the same time when GridSelectionFlags.Multiple is enabled.
Keyboard	Allows extend existing selection when user holds SHIFT+Arrow keys.
Any	Default behavior for selecting cells: Rows, Columns, Cells, Table, Multiple, Extends Shift Key support and alpha blending.
None	Disable selecting cells.

You can combine more than one flags to customize the current selection behavior.

### Example

Following code example illustrates how to set the selection mode for selecting multiple rows with alpha blending.

#### [C#]

```
this.gridGroupingControl1.TableOptions.AllowSelection =
GridSelectionFlags.AlphaBlend | GridSelectionFlags.Row |
GridSelectionFlags.Multiple;
```

#### [VB .NET]

```
Me.gridGroupingControl1.TableOptions.AllowSelection =
GridSelectionFlags.AlphaBlend Or GridSelectionFlags.Row Or
GridSelectionFlags.Multiple
```

Statistics								Drag a column header here to group by that column.
	ID	losses	School	Sport	ties	wins	year	
	1	7	Duke	Basketball	0	26	2003	
▶	2	10	Maryland	Basketball	0	21	2003	
	3	6	Wake Forest	Basketball	0	25	2003	
	4	15	Georgia Tech	Basketball	0	16	2003	
	5	16	Virginia	Basketball	0	16	2003	
	6	13	NC State	Basketball	0	18	2003	
	7	16	North Carolina	Basketball	0	19	2003	

Figure 331: Model Based Selection Illustrated

### Format Selection

It is possible to modify the default color used for alphablend selection. This can be achieved by assigning a desired color to the AlphaBlendSelectionColor property. The example given below uses Red Color for alpha blending.

#### [C#]

```
this.gridGroupingControl1.TableOptions.AllowSelection =
GridSelectionFlags.AlphaBlend | GridSelectionFlags.Cell;
this.gridGroupingControl1.TableModel.Options.AlphaBlendSelectionColor =
Color.Red;
```

#### [VB .NET]

```
Me.gridGroupingControl1.TableOptions.AllowSelection =
GridSelectionFlags.AlphaBlend Or GridSelectionFlags.Cell
Me.gridGroupingControl1.TableModel.Options.AlphaBlendSelectionColor =
Color.Red
```



The screenshot shows a Windows application window titled "Statistics". Inside, there is a table with the following data:

	ID	losses	School	Sport	ties	wins	year
	167	5	Georgia Tech	Football	0	6	1995
	168	10	Georgia Tech	Football	0	1	1994
►	169	6	Georgia Tech	Football	0	5	1993
	170	6	Georgia Tech	Football	0	5	1992
	171	3	Maryland	Football	0	11	2002
	172	2	Maryland	Football	0	10	2001
	173	6	Maryland	Football	0	5	2000
	174	6	Maryland	Football	0	5	1999

Figure 332: AlphaBlendSelectionColor = "Red"



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Grouping Grid Options\Table Options Demo

#### 4.3.4.7.2 Record Based Selection

This type of selection mechanism allows selection in terms of record. It is not cell based. This selection mode is specifically designed for a Grouping Grid and hence it is aware of nested tables, nested groups, and the like. Any selection that is record based affects the **Table.SelectedRecords** collection.

Grid Grouping control offers three types of record based selections which are together called as **ListBoxSelectionMode**. To enable record based selection, you need to set the **ListBoxSelectionMode** property to a value other than None. Once a listbox selection is enabled, it automatically turns off the model based selection by assigning **None** to the **AllowSelection** property.

Following code examples illustrate the different types of record based selections.

##### SelectionMode - One

It allows you to select only one item (record).

[C#]

```
this.gridGroupingControl1.TableOptions.ListBoxSelectionMode =  
SelectionMode.One;
```

[VB .NET]

```
Me.gridGroupingControl1.TableOptions.ListBoxSelectionMode =  
SelectionMode.One
```

ID	losses	School	Sport	ties	wins	year
1	7	Duke	Basketball	0	26	2003
2	10	Maryland	Basketball	0	21	2003
3	6	Wake Forest	Basketball	0	25	2003
4	15	Georgia Tech	Basketball	0	16	2003
5	16	Virginia	Basketball	0	16	2003
6	13	NC State	Basketball	0	18	2003

Figure 333: Selection Mode set to "One"

### SelectionMode - MultiSimple

You would be able to select multiple items individually. It does not support the use of SHIFT, CTRL and ARROW keys to extend the selection.

[C#]

```
this.gridGroupingControl1.TableOptions.ListBoxSelectionMode =  
SelectionMode.MultiSimple;
```

[VB .NET]

```
Me.gridGroupingControl1.TableOptions.ListBoxSelectionMode =  
SelectionMode.MultiSimple
```

	ID	losses	School	Sport	ties	wins	year
wins: 0 - 3 Items							
	139	11	Duke	Football	0	0	20
	140	11	Duke	Football	0	0	20
	144	11	Duke	Football	0	0	19
wins: 1 - 2 Items							
	168	10	Georgia Tech	Football	0	1	19
	222	10	Wake Forest	Football	0	1	19
wins: 2 - 8 Items							
	16	8	North Carolina	Football	0	2	20
	138	10	Duke	Football	0	2	20
	143	9	Duke	Football	0	2	19
	148	9	Duke	Football	0	2	19

Figure 334: Selection Mode set to "MultiSimple"

### SelectionMode - MultiExtended

This selection type allows multiple items selection through Shift, Ctrl and Arrow keys.

#### [C#]

```
this.gridGroupingControl1.TableOptions.ListBoxSelectionMode =  
SelectionMode.MultiExtended;
```

#### [VB .NET]

```
Me.gridGroupingControl1.TableOptions.ListBoxSelectionMode =  
SelectionMode.MultiExtended
```



Figure 335: Selection Mode set to "MultiExtended"

### Format ListBox Selections

ListBoxSelection appearance can be customized by setting the properties: SelectionBackColor, SelectionTextColor and ListBoxSelectionColorOptions.

By default, SystemColors.Highlight and SystemColors.HighlightText are the colors used as backcolor and textcolor to highlight the selected records. SelectionBackColor and SelectionTextColor property settings can be used to override these default colors.

ListBoxSelectionColorOptions is used to control the appearance of the selections. The GridListBoxSelectionColorOptions enumeration specifies the options for this property.

- ApplySelectionColor

Gets the required colors from the SelectionBackColor and SelectionTextColor properties.

[C#]

```
this.gridGroupingControl1.TableOptions.ListBoxSelectionColorOptions =
    GridListBoxSelectionColorOptions.ApplySelectionColor;
this.gridGroupingControl1.TableOptions.SelectionBackColor =
    Color.PaleGreen;
```

```
this.gridGroupingControl1.TableOptions.SelectionTextColor =  
Color.Green;
```

**[VB.NET]**

```
Me.gridGroupingControl1.TableOptions.ListBoxSelectionColorOptions =  
GridListBoxSelectionColorOptions.ApplySelectionColor  
Me.gridGroupingControl1.TableOptions.SelectionBackColor =  
Color.PaleGreen  
Me.gridGroupingControl1.TableOptions.SelectionTextColor = Color.Green
```

Here is the effect of the above settings.

ID	losses	School	Sport	ties	wins	year
136	3	Clemson	Football	0	9	1993
137	6	Clemson	Football	0	5	1992
138	10	Duke	Football	0	2	2002
139	11	Duke	Football	0	0	2001
140	11	Duke	Football	0	0	2000
▶ 141	8	Duke	Football	0	3	1999
◀ 142	7	Duke	Football	0	4	1998

Figure 336: SelectionBackColor = "PaleGreen" and SelectionTextColor = "Green"

- Draw Alphablend

Draws alphablending over the selected row.

**[C#]**

```
this.gridGroupingControl1.TableOptions.ListBoxSelectionColorOptions =  
GridListBoxSelectionColorOptions.DrawAlphablend;
```

**[VB.NET]**

```
Me.gridGroupingControl1.TableOptions.ListBoxSelectionColorOptions =  
GridListBoxSelectionColorOptions.DrawAlphablend
```

5	16	Virginia	Basketball	0	16	2003
► 6	13	NC State	Basketball	0	18	2003
7	16	North Carolina	Basketball	0	19	2003
8	13	Clemson	Basketball	0	15	2003
9	15	Florida State	Basketball	0	14	2003

*Figure 337: Drawing Alpha Blending over the Selected Row*

- InvertCells

Inverts the cells in selected row. As a result, the back color of the cell is used to draw the text and the CellTextColor becomes its BackColor.

**[C#]**

```
this.gridGroupingControl1.TableOptions.ListBoxSelectionColorOptions =
GridListBoxSelectionColorOptions.InvertCells;
```

**[VB .NET]**

```
Me.gridGroupingControl1.TableOptions.ListBoxSelectionColorOptions =
GridListBoxSelectionColorOptions.InvertCells
```

10	7	Duke	Football	0	3	2003
► 11	3	Maryland	Football	0	6	2003
12	5	Wake Forest	Football	0	5	2003

*Figure 338: Inverting Cells in the Selected Row*

- None

Do not change the appearance of the cells. The cell appearance could be specified manually by handling TableControlPrepareViewStyleInfo and TableControlCellDrawn events.

**[C#]**

```
this.gridGroupingControl1.TableOptions.ListBoxSelectionColorOptions =
GridListBoxSelectionColorOptions.None;
```

**[VB .NET]**

```
Me.gridGroupingControl1.TableOptions.ListBoxSelectionColorOptions =
GridListBoxSelectionColorOptions.None
```

	9	15	Florida State	Basketball	0	14	2003
▶	10	7	Duke	Football	0	3	2003
	11	3	Maryland	Football	0	6	2003

Figure 339: Cell Appearance customized by handling  
TableControlPrepareViewStyleInfo and TableControlCellDrawn Events

### ListBoxSelection CurrentCellOptions

When **ListBoxSelection** mode was set, you will be able to control the appearance and behavior of the **CurrentCell** by setting **ListBoxSelectionCurrentCellOptions** property to a desired value. Possible values are defined by the **GridListBoxSelectionCurrentCellOptions** enumeration which are described below.

- **HideCurrentCell**

Don't select a current cell in the current row.

	10	7	Duke	Football	0	3	2003
▶	11	3	Maryland	Football	0	6	2003
	12	5	Wake Forest	Football	0	5	2003

Figure 340: *ListBoxSelectionCurrentCellOptions.HideCurrentCell Enabled*

- **WhiteCurrentCell**

When a current cell is in current row, it is drawn with the original cell background color.

	8	13	Clemson	Basketball	0	15	2003
▶	9	15	Florida State	Basketball	0	14	2003
	10	7	Duke	Football	0	3	2003

Figure 341: *ListBoxSelectionCurrentCellOptions.WhiteCurrentCell Enabled*

- **None**

When a current cell is in current row, it is drawn with the same color used for highlighting the whole record.

	11	3	Maryland	Football	0	6	2003
▶	12	5	Wake Forest	Football	0	5	2003
	13	4	Georgia Tech	Football	0	5	2003

Figure 342: *ListBoxSelectionCurrentCellOptions.None Enabled*

- MoveCurrentCellWithMouse

Used only with SelectionMode.MultiExtended. Moves current cell when user extends the selection with mouse. Below image well illustrates this mode. Here, the selection started with the cell {R2:C1} and is extended up to Row4 through a mouse drag that made the current cell to shift to the cell {R4:C1} by following the mouse.

	ID	losses	School	Sport	ties	wins	year
	1	7	Duke	Basketball	0	26	2003
Selection Start	2	10	Maryland	Basketball	0	21	2003
	3	6	Wake Forest	Basketball	0	25	2003
Selection End	4	15	Georgia Tech	Basketball	0	16	2003
	5	16	Virginia	Basketball	0	16	2003
	6						
	7						
	8						
	9						
	10						
	11						
	12						
	13						
	14						
	15						
	16						

Figure 343: *ListBoxSelectionCurrentCellOptions.MoveCurrentCellWithMouse Enabled*

**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Grouping.Windows\Samples\2.0\Grouping Grid Options\Table Options Demo

#### 4.3.4.7.3 Focused Selection

Grouping Grid selection behavior can also be customized by handling appropriate events as required. Here is an example implementation that focuses the current selection onto the desired grid elements like only the Current Cell, only the entire Row that contains the current cell, only the entire column that contains the current cell or both the row and the column that contains the current cell. The focused grid element is highlighted to show the current selection.

##### Selection Options

Name	Description
Cell Only	Selects only the individual cells.
Row Only	Selects only the row.
Column Only	Selects only the column.

Row and Column	Selects the row and the column with respect to the current cell.
Default	Enables ListBoxSelection mode by default.
None	Disable the cell selection.

## Implementation

Follow the steps below to create a sample that shows the above selections.

1. Create a grid grouping control and bind it to any data table. This example uses the grouping grid that has been bound to the Statistics Table from Northwind.MDB.
2. Setup the designer to add options for different selection types. Add six radio buttons to the form to enable the selection options **Cell Only**, **Row Only**, **Column Only**, **Row and Column**, **Default** and **None**.
3. Set the required flags with respect to the current cell.

### [C#]

```
this.gridGroupingControl1.TableModel.Options.RefreshCurrentCellBehavior = GridRefreshCurrentCellBehavior.RefreshCell;
this.gridGroupingControl1.TableModel.Options.ShowCurrentCellBorderBehavior = GridShowCurrentCellBorder.GrayWhenLostFocus;
```

### [VB .NET]

```
Me.gridGroupingControl1.TableModel.Options.RefreshCurrentCellBehavior = GridRefreshCurrentCellBehavior.RefreshCell
Me.gridGroupingControl1.TableModel.Options.ShowCurrentCellBorderBehavior = GridShowCurrentCellBorder.GrayWhenLostFocus
```

4. Handle PrepareViewStyleInfo event to focus the current selection according to the chosen selection type. It also includes the code to highlight the current selection. This works for **CellOnly**, **RowOnly**, **ColumnOnly** and **Row and Column** types.

### [C#]

```
this.gridGroupingControl1.TableControl.PrepareViewStyleInfo += new GridPrepareViewStyleInfoEventHandler(TableControl_PreparesViewStyleInfo);
;

void TableControl_PreparesViewStyleInfo(object sender,
```

```
Syncfusion.Windows.Forms.Grid.GridPrepareViewStyleInfoEventArgs e)
{
    GridCurrentCell cc = gridGroupingControl1.TableControl.CurrentCell;
    GridControlBase grid =
this.gridGroupingControl1.TableControl.CurrentCell.Grid;

    // Code for RowOnly.
    if (radioButton3.Checked)
    {
        // Highlight the current row with SystemColors.Highlight and
Bold font.
        if (e.RowIndex > grid.Model.Rows.HeaderCount && e.ColIndex >
grid.Model.Cols.HeaderCount
            && cc.HasCurrentCellAt(e.RowIndex))
        {
            e.Style.Interior = new BrushInfo(SystemColors.Highlight);
            e.Style.TextColor = SystemColors.HighlightText;
            e.Style.Font.Bold = true;
        }
    }

    // Code for CellOnly.
    else if (radioButton2.Checked)
    {
        // Highlight the current cell with SystemColors.Highlight and
Bold font.
        if (e.RowIndex > grid.Model.Rows.HeaderCount && e.ColIndex >
grid.Model.Cols.HeaderCount
            && cc.HasCurrentCellAt(e.RowIndex, e.ColIndex))
        {
            e.Style.Interior = new BrushInfo(SystemColors.Highlight);
            e.Style.TextColor = SystemColors.HighlightText;
            e.Style.Font.Bold = true;
        }
    }

    // Code for ColumnOnly.
    else if (radioButton4.Checked)
    {
        // Highlight the current column with SystemColors.Highlight and
Bold font.
        if (e.RowIndex > grid.Model.Rows.HeaderCount && e.ColIndex >
grid.Model.Cols.HeaderCount
            && cc.ColIndex == e.ColIndex)
        {
            e.Style.Interior = new BrushInfo(SystemColors.Highlight);
            e.Style.TextColor = SystemColors.HighlightText;
        }
    }
}
```

```
        e.Style.Font.Bold = true;
    }

}

// Code for Row and Column.
else if (radioButton5.Checked)
{
    // Highlight the current row and column with
    SystemColors.Highlight and Bold font.
    if (e.RowIndex > grid.Model.Rows.HeaderCount && e.ColIndex >
grid.Model.Cols.HeaderCount
        && (cc.RowIndex == e.RowIndex || cc.ColIndex == e.ColIndex))
    {
        e.Style.Interior = new BrushInfo(SystemColors.Highlight);
        e.Style.TextColor = SystemColors.HighlightText;
        e.Style.Font.Bold = true;
    }
}
}
```

**[VB.NET]**

```
AddHandler gridGroupingControl1.TableControl.PrepareViewStyleInfo,
AddressOf TableControl_PreparesViewStyleInfo

Private Sub TableControl_PreparesViewStyleInfo(ByVal sender As Object,
ByVal e As GridPrepareViewStyleInfoEventArgs)
    Dim cc As GridCurrentCell =
    gridGroupingControl1.TableControl.CurrentCell
    Dim grid As GridControlBase =
    Me.gridGroupingControl1.TableControl.CurrentCell.Grid

    ' Code for RowOnly.
    If radioButton3.Checked Then

        ' Highlight the current row with SystemColors.Highlight and Bold
        font.
        If e.RowIndex > grid.Model.Rows.HeaderCount AndAlso e.ColIndex >
grid.Model.Cols.HeaderCount AndAlso
        cc.HasCurrentCellAt(e.RowIndex) Then
            e.Style.Interior = New BrushInfo(SystemColors.Highlight)
            e.Style.TextColor = SystemColors.HighlightText
            e.Style.Font.Bold = True
        End If

        ' Code for CellOnly.
    ElseIf radioButton2.Checked Then
```

```
' Highlight the current cell with SystemColors.Highlight and Bold font.  
If e.RowIndex > grid.Model.Rows.HeaderCount AndAlso e.ColIndex >  
grid.Model.Cols.HeaderCount AndAlso  
cc.HasCurrentCellAt(e.RowIndex, e.ColIndex) Then  
    e.Style.Interior = New BrushInfo(SystemColors.Highlight)  
    e.Style.TextColor = SystemColors.HighlightText  
    e.Style.Font.Bold = True  
End If  
  
' Code for ColumnOnly.  
ElseIf radioButton4.Checked Then  
  
' Highlight the current column with SystemColors.Highlight and Bold font.  
If e.RowIndex > grid.Model.Rows.HeaderCount AndAlso e.ColIndex >  
grid.Model.Cols.HeaderCount AndAlso cc.ColIndex = e.ColIndex Then  
    e.Style.Interior = New BrushInfo(SystemColors.Highlight)  
    e.Style.TextColor = SystemColors.HighlightText  
    e.Style.Font.Bold = True  
End If  
  
' Code for Row and Column.  
ElseIf radioButton5.Checked Then  
  
' Highlight the current row and column with SystemColors.Highlight and Bold font.  
If e.RowIndex > grid.Model.Rows.HeaderCount AndAlso e.ColIndex >  
grid.Model.Cols.HeaderCount AndAlso (cc.RowIndex =  
e.RowIndex OrElse cc.ColIndex = e.ColIndex) Then  
    e.Style.Interior = New BrushInfo(SystemColors.Highlight)  
    e.Style.TextColor = SystemColors.HighlightText  
    e.Style.Font.Bold = True  
End If  
End If  
End Sub
```

5. Enable ListBoxSelection mode to expose the default selection.

[C#]

```
// Code for Default option.  
private void radioButton1_CheckedChanged(object sender,  
System.EventArgs e)  
{  
    if (this.radioButton1.Checked)
```

```
this.gridGroupingControl1.TableOptions.ListBoxSelectionMode =  
SelectionMode.One;  
else  
this.gridGroupingControl1.TableOptions.ListBoxSelectionMode =  
SelectionMode.None;  
foreach (Table t in  
this.gridGroupingControl1.Engine.EnumerateTables())  
  
this.gridGroupingControl1.GetTable(t.TableDescriptor.Name).SelectedRecords.Clear();  
}
```

**[VB .NET]**

```
' Code for Default option.  
Private Sub radioButton1_CheckedChanged(ByVal sender As Object, ByVal e  
As System.EventArgs) Handles radioButton1.CheckedChanged  
If Me.radioButton1.Checked Then  
    Me.gridGroupingControl1.TableOptions.ListBoxSelectionMode =  
    SelectionMode.One  
Else  
    Me.gridGroupingControl1.TableOptions.ListBoxSelectionMode =  
    SelectionMode.None  
End If  
Dim t As Table  
For Each t In Me.gridGroupingControl1.Engine.EnumerateTables()  
    Me.gridGroupingControl1.GetTable(t.TableDescriptor.Name).Selected  
    Records.Clear()  
Next t  
End Sub
```

6. Below is the code for None option that disables the selection.

**[C#]**

```
// Code for None option.  
private void radioButton6_CheckedChanged(object sender,  
System.EventArgs e)  
{  
    if(this.radioButton6.Checked)  
  
this.gridGroupingControl1.TableModel.Options.ShowCurrentCellBorderBehav  
ior = GridShowCurrentCellBorder.HideAlways;  
    else  
  
this.gridGroupingControl1.TableModel.Options.ShowCurrentCellBorderBehav  
ior = GridShowCurrentCellBorder.GrayWhenLostFocus;
```

```
}
```

**[VB.NET]**

```
' Code for None option.  
Private Sub radioButton6_CheckedChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles radioButton6.CheckedChanged  
    If Me.radioButton6.Checked Then  
        Me.gridGroupingControl1.TableModel.Options.ShowCurrentCellBorderBehavior = GridShowCurrentCellBorder.HideAlways  
    Else  
        Me.gridGroupingControl1.TableModel.Options.ShowCurrentCellBorderBehavior = GridShowCurrentCellBorder.GrayWhenLostFocus  
    End If  
End Sub
```

7. Refresh the table control once the selection type is changed. You could also handle TableControlCurrentCellActivating event for this purpose.

**[C#]**

```
private void radioButton2_CheckedChanged(object sender,  
System.EventArgs e)  
{  
    this.gridGroupingControl1.TableControl.Refresh();  
}  
  
private void radioButton3_CheckedChanged(object sender,  
System.EventArgs e)  
{  
    this.gridGroupingControl1.TableControl.Refresh();  
}  
  
private void radioButton4_CheckedChanged(object sender,  
System.EventArgs e)  
{  
    this.gridGroupingControl1.TableControl.Refresh();  
}  
  
private void radioButton5_CheckedChanged(object sender,  
System.EventArgs e)  
{  
    this.gridGroupingControl1.TableControl.Refresh();  
}  
  
void TableControl_CurrentCellActivating(object sender,
```

```
GridCurrentCellActivatingEventArgs e)
{
    this.gridGroupingControl1.TableControl.Refresh();
}
```

**[VB.NET]**

```
Private Sub radioButton2_CheckedChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles radioButton2.CheckedChanged
    Me.gridGroupingControl1.TableControl.Refresh()
End Sub

Private Sub radioButton3_CheckedChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles radioButton3.CheckedChanged
    Me.gridGroupingControl1.TableControl.Refresh()
End Sub

Private Sub radioButton4_CheckedChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles radioButton4.CheckedChanged
    Me.gridGroupingControl1.TableControl.Refresh()
End Sub

Private Sub radioButton5_CheckedChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles radioButton5.CheckedChanged
    Me.gridGroupingControl1.TableControl.Refresh()
End Sub

Private Sub TableControl_CurrentCellActivating(ByVal sender As Object, ByVal e As GridCurrentCellActivatingEventArgs)
    Me.gridGroupingControl1.TableControl.Refresh()
End Sub
```

Here is a sample output that focuses the current row and column.

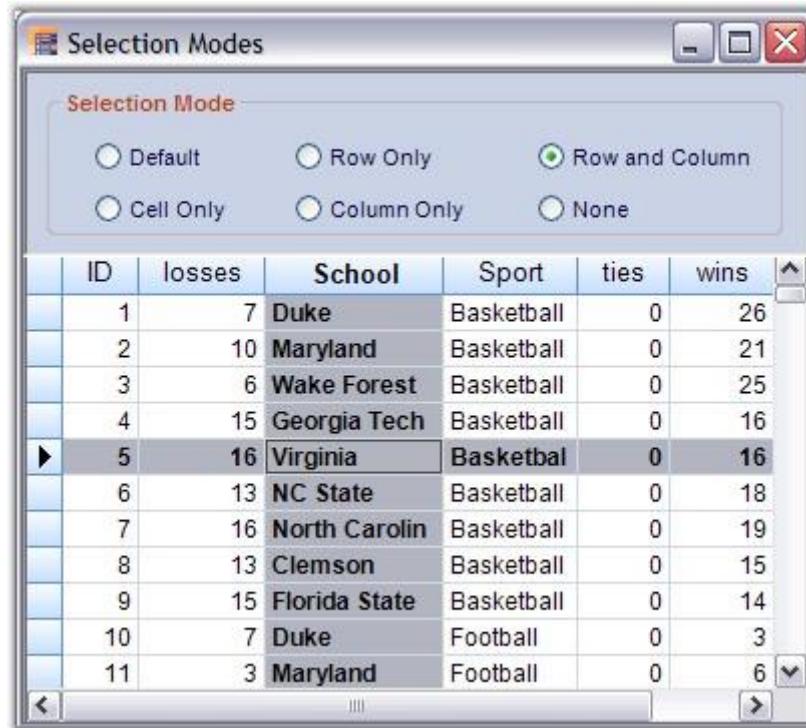


Figure 344: Focused Selection Illustrated



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Grouping.Windows\Samples\2.0\Selection\Focused Selection Demo

#### 4.3.4.7.4 Multiple Record Selection

Grid Table supports selection of multiple records. Each record that is being selected is added into the **SelectedRecords** collection which manages these records. You can iterate through this collection in order to step through all records marked as selected. When records are added or removed from this collection, the grid raises the events, **SelectedRecordsChanging** and **SelectedRecordsChanged**. This section demonstrates how to work with the SelectedRecords collection.

##### Selecting Multiple Records

Multiple records can be selected at a time by adding the desired record specifications into the **SelectedRecords** collection. The following code example illustrates this process. It selects the records with indexes 2, 4 and 0 by adding them into the **SelectedRecords** collection.

**[C#]**

```
Record r1 = this.gridGroupingControl1.Table.Records[2];
Record r2 = this.gridGroupingControl1.Table.Records[4];
Record r3 = this.gridGroupingControl1.Table.Records[0];

Table t = this.gridGroupingControl1.Table;
t.SelectedRecords.Add(r1);
t.SelectedRecords.Add(r2);
t.SelectedRecords.Add(r3);
```

**[VB.NET]**

```
Dim r1 As Record = Me.gridGroupingControl1.Table.Records(2)
Dim r2 As Record = Me.gridGroupingControl1.Table.Records(4)
Dim r3 As Record = Me.gridGroupingControl1.Table.Records(0)

Dim t As Table = Me.gridGroupingControl1.Table
t.SelectedRecords.Add(r1)
t.SelectedRecords.Add(r2)
t.SelectedRecords.Add(r3)
```

	OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate
	10248	VINET	5	7/4/1996	8/1/1996
	10249	TOMSP	6	7/5/1996	8/16/1996
	10250	HANAR	4	7/8/1996	8/5/1996
	10251	VICTE	3	7/8/1996	8/5/1996
	10252	SUPRD	4	7/9/1996	8/6/1996
	10253	HANAR	3	7/10/1996	7/24/1996
	10254	CHOPS	5	7/11/1996	8/8/1996
	10255	RICSU	9	7/12/1996	8/9/1996
	10256	WELLI	3	7/15/1996	8/12/1996
	10257	LILAS	4	7/16/1996	8/13/1996

Figure 345: Selecting Multiple Records

 Note: For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Selection\Multi Record Selection Demo

### RecordSelection with NestedTables

When nested tables are used, you can extend the record selection mechanisms to each of the child table by accessing the SelectedRecords collection of the desired child table.

#### [C#]

```
// For Parent Table.
Record r1 = this.gridGroupingControl1.Table.Records[1];
Record r2 = this.gridGroupingControl1.Table.Records[2];

Table t = this.gridGroupingControl1.Table;
t.SelectedRecords.Add(r1);
t.SelectedRecords.Add(r2);
t.SelectedRecords.Add(r3);

// For Child Table.
Record cr1 = this.gridGroupingControl1.GetTable("Orders").Records[7];
Record cr2 = this.gridGroupingControl1.GetTable("Orders").Records[12];

this.gridGroupingControl1.GetTable("Orders").SelectedRecords.Add(or1);
this.gridGroupingControl1.GetTable("Orders").SelectedRecords.Add(or2);
```

#### [VB .NET]

```
' For Parent Table.
Dim r1 As Record = Me.gridGroupingControl1.Table.Records(1)
Dim r2 As Record = Me.gridGroupingControl1.Table.Records(2)

Dim t As Table = Me.gridGroupingControl1.Table
t.SelectedRecords.Add(r1)
t.SelectedRecords.Add(r2)
t.SelectedRecords.Add(r3)

' For Child Table.
Dim cr1 As Record =
Me.gridGroupingControl1.GetTable("Orders").Records(7)
Dim cr2 As Record =
Me.gridGroupingControl1.GetTable("Orders").Records(12)
```

```
Me.gridGroupingControll.GetTable("Orders").SelectedRecords.Add(or1)
Me.gridGroupingControll.GetTable("Orders").SelectedRecords.Add(or2)
```

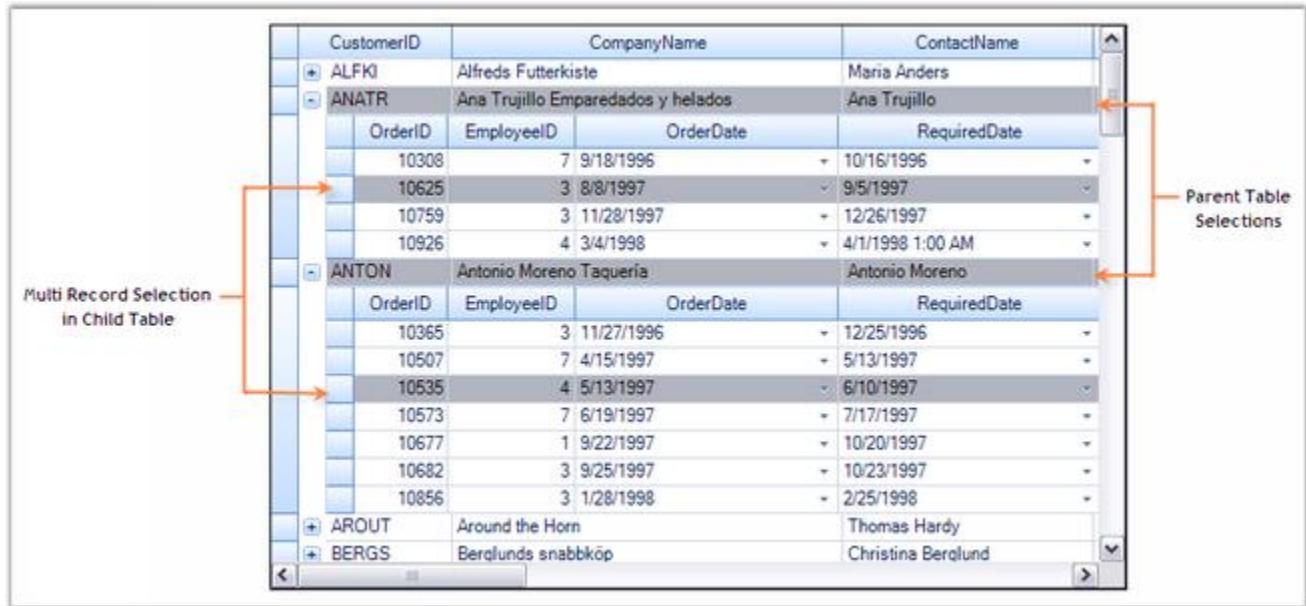


Figure 346: Record Selection in Nested Tables

## Record Search

To search for a particular record, the `SelectedRecords` collection provides a method called `FindRecord()`. This method search for the occurrences of the specified record and returns a zero-based index of the occurrence found. If there is no such record, then returns -1. It comes in two versions: one accepts the whole record as its parameter and the other accepts the position of the record in the underlying datasource.

### [C#]

```
Record rec = this.gridGroupingControll.Table.Records[2];

// Search for the record 'rec'.
int index =
this.gridGroupingControll.Table.SelectedRecords.FindRecord(rec);

// Search for the record with index 2.
int index2 =
this.gridGroupingControll.Table.SelectedRecords.FindRecord(2);
```

### [VB .NET]

```
Dim rec As Record = Me.gridGroupingControl1.Table.Records(2)

' Search for the record 'rec'.
Dim index As Integer =
Me.gridGroupingControl1.Table.SelectedRecords.FindRecord(rec)

' Search for the record with index 2.
Dim index2 As Integer =
Me.gridGroupingControl1.Table.SelectedRecords.FindRecord(2)
```

### Removing a RecordSelection

A record can be removed from the SelectedRecords collection by using the methods Remove() and RemoveAt(). A call to Remove() requires you to specify the whole record itself as parameter. In case if you know only the record index, you could then make use of RemoveAt(). Both the methods remove the specified record from the collection and mark it as deselect.

#### [C#]

```
Record rec = this.gridGroupingControl1.Table.Records[2];

// Remove the record 'rec'.
this.gridGroupingControl1.Table.SelectedRecords.Remove(rec);

// Remove the record at the index 2.
this.gridGroupingControl1.Table.SelectedRecords.RemoveAt(2);
```

#### [VB .NET]

```
Dim rec As Record = Me.gridGroupingControl1.Table.Records(2)

' Remove the record 'rec'.
Me.gridGroupingControl1.Table.SelectedRecords.Remove(rec)

' Remove the record at the index 2.
Me.gridGroupingControl1.Table.SelectedRecords.RemoveAt(2)
```

### Clear Selection

To remove all the selections from the grid, you can call SelectedRecords.Clear() method that removes all the elements from the collection and mark them as deselect.

#### [C#]

```
this.gridGroupingControl1.Table.SelectedRecords.Clear();
```

[VB .NET]

```
Me.gridGroupingControl1.Table.SelectedRecords.Clear()
```



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Selection\Multi Record Selection Demo

#### 4.3.4.7.5 Selected Ranges Collection

The selections that are made by the user are saved into a collection named **TableModel.SelectedRanges**. If the Selection option is turned on, then the grid will always listen to the selections that are being made and records all those selections into the SelectedRanges collection. You can loop through every selection range of this collection to get the information about the records that have been selected. The **SelectedRanges.ActiveRange** property gives the current selection range (i.e. last range in the collection).

#### Example

This example shows how to loop through the **SelectedRanges** collection to retrieve the information about the records that are being selected.

1. Turn on any type of selection. Here the record-based selection is active. It is enabled by setting the **ListBoxSelectionMode** property to a value other than None. You could set the selection colors as well.

[C#]

```
this.gridGroupingControl1.TableOptions.ListBoxSelectionMode =
SelectionMode.MultiExtended;
this.gridGroupingControl1.TableOptions.ListBoxSelectionColorOptions =
GridListBoxSelectionColorOptions.DrawAlphablend;
this.gridGroupingControl1.TableModel.Options.AlphaBlendSelectionColor =
Color.Red;
```

**[VB.NET]**

```
Me.gridGroupingControl1.TableOptions.ListBoxSelectionMode =  
SelectionMode.MultiExtended  
Me.gridGroupingControl1.TableOptions.ListBoxSelectionColorOptions =  
GridListBoxSelectionColorOptions.DrawAlphablend  
Me.gridGroupingControl1.TableModel.Options.AlphaBlendSelectionColor =  
Color.Red
```

2. The below code loops through the ranges of all the selections and write the record values that have been selected to a listbox control.

**[C#]**

```
foreach (GridRangeInfo range in  
gridGroupingControl1.TableModel.SelectedRanges)  
{  
    if (range.IsRows)  
    {  
        for (int i = range.Top; i <= range.Bottom; i++)  
        {  
            Record rec =  
gridGroupingControl1.Table.DisplayElements[i].GetRecord();  
            listBox1.Items.Add(rec.ToString());  
        }  
    }  
}
```

**[VB.NET]**

```
For Each range As GridRangeInfo In  
gridGroupingControl1.TableModel.SelectedRanges  
    If range.IsRows Then  
        Dim i As Integer = range.Top  
        Do While i <= range.Bottom  
            Dim rec As Record =  
gridGroupingControl1.Table.DisplayElements(i).GetRecord()  
            listBox1.Items.Add(rec.ToString())  
            i += 1  
        Loop  
    End If  
Next range
```

3. Here is a sample screenshot.

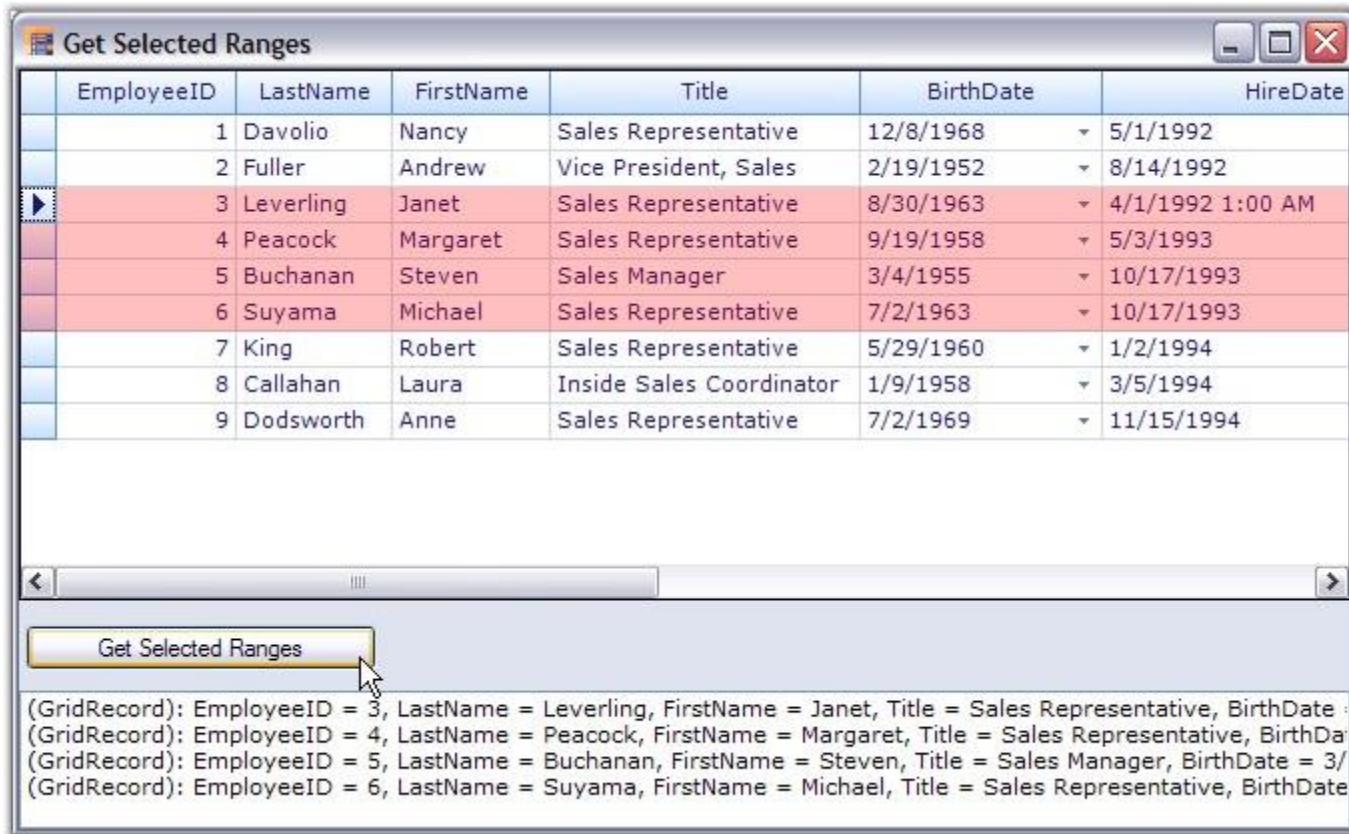


Figure 347: Retrieving Information about Selected Records by using the SelectedRanges Collection

#### 4.3.4.8 Serialization

In this section, we will discuss how to serialize and deserialize grouping grid schema information.

**Serialization** is the process of saving the object state into a stream of bytes for further use. The reverse process is **Deserialization**. Through serialization, the objects are made portable so that they can be serialized at one end and then transferred to the other end of a network where they will again be deserialized into its original form for use.

Grid Grouping control supports two forms of serialization.

##### 4.3.4.8.1 Xml Serialization

With **XmlSerialization**, the grid schema information can be converted into Xml format. Grouping Grid provides two methods to support Xml Serialization.

- **WriteXmlSchema** - It writes the engine settings into an Xml stream(Serialization).
- **ApplyXmlSchema** - It loads the engine settings from an Xml stream(Deserialization).

All the grid elements can be serialized. Not only the data but also the look and feel of the grid can be serialized and deserialized. The following code example best illustrates this process.

### Example

1. Setup a grouping grid and load it with some data. Save the initial state of the grid schema so that it could be used to reset the grid.

#### [C#]

```
// Knowing the initial state.  
System.IO.MemoryStream stream;  
stream = new System.IO.MemoryStream();  
this.gridGroupingControl1.WriteXmlSchema(new XmlTextWriter(stream,  
null));
```

#### [VB .NET]

```
' Knowing the initial state.  
Private stream As System.IO.MemoryStream  
stream = New System.IO.MemoryStream()  
Me.gridGroupingControl1.WriteXmlSchema(New XmlTextWriter(stream,  
Nothing))
```

2. Apply the look and feel properties that you desire.

#### [C#]

```
// Customize the Appearance.  
this.gridGroupingControl1.TableOptions.GridVisualStyles =  
GridVisualStyles.Office2007Blue;  
this.gridGroupingControl1.TableOptions.GridLineBorder = new  
GridBorder(GridBorderStyle.Solid, Color.FromArgb(208, 215, 229),  
GridBorderWeight.Thin);  
this.gridGroupingControl1.TopLevelGroupOptions.ShowCaption = false;  
this.gridGroupingControl1.Appearance.AnyCell.Font.Facename = "Verdana";  
this.gridGroupingControl1.Appearance.AnyCell.TextColor =  
Color.MidnightBlue;  
this.gridGroupingControl1.TableDescriptor.Appearance.AlternateRecordFie
```

```
ldCell.Interior = New BrushInfo(Color.Orange);
```

**[VB.NET]**

```
' Customize the Appearance.  
Me.gridGroupingControl1.TableOptions.GridVisualStyles =  
GridVisualStyles.Office2007Blue  
Me.gridGroupingControl1.TableOptions.GridLineBorder = New  
GridBorder(GridBorderStyle.Solid, Color.FromArgb(208, 215, 229),  
GridBorderWeight.Thin)  
Me.gridGroupingControl1.TopLevelGroupOptions.ShowCaption = False  
Me.gridGroupingControl1.Appearance.AnyCell.Font.Facename = "Verdana"  
Me.gridGroupingControl1.Appearance.AnyCell.TextColor =  
Color.MidnightBlue  
Me.gridGroupingControl1.TableDescriptor.Appearance.AlternateRecordField  
Cell.Interior = New BrushInfo(Color.Orange)
```

3. Create a button name 'Serialize', clicking which will start the serialization process. Add the below code into the ButtonClick event handler. This will save the grid schema into an Xml file.

**[C#]**

```
// Serialization  
private void Serialize_Click(object sender, System.EventArgs e)  
{  
    FileDialog dlg = new SaveFileDialog();  
    dlg.AddExtension = true;  
    dlg.Filter = "xml files (*.xml)|*.xml|All files (*.*)|*.*";  
    if (dlg.ShowDialog() == DialogResult.OK)  
    {  
        XmlTextWriter xw = new XmlTextWriter(dlg.FileName,  
System.Text.Encoding.UTF8);  
        xw.Formatting = System.Xml.Formatting.Indented;  
        this.gridGroupingControl1.WriteXmlSchema(xw);  
        xw.Close();  
    }  
}
```

**[VB.NET]**

```
' Serialization  
Private Sub Serialize_Click(ByVal sender As Object, ByVal e As  
System.EventArgs) Handles btnSaveXmlSchema.Click  
    Dim dlg As FileDialog = New SaveFileDialog()  
    dlg.AddExtension = True
```

```
dlg.Filter = "xml files (*.xml)|*.xml|All files (*.*)|*.*"
If dlg.ShowDialog() = DialogResult.OK Then
    Dim xw As XmlTextWriter = New XmlTextWriter(dlg.FileName,
        System.Text.Encoding.UTF8)
    xw.Formatting = System.Xml.Formatting.Indented
    Me.gridGroupingControl1.WriteXmlSchema(xw)
    xw.Close()
End If
End Sub
```

4. Create another button named 'Deserialize' to deserialize the grid. The following code will help you to load the grid schema back from an Xml file.

**[C#]**

```
// Deserialization
private void btnLoadXmlSchema_Click(object sender, System.EventArgs e)
{
    FileDialog dlg = new OpenFileDialog();
    dlg.Filter = "xml files (*.xml)|*.xml|All files (*.*)|*.*";
    if (dlg.ShowDialog() == DialogResult.OK)
    {
        XmlReader xr = new XmlTextReader(dlg.FileName);
        this.gridGroupingControl1.ApplyXmlSchema(xr);
        xr.Close();
    }
}
```

**[VB .NET]**

```
' Deserialization
Private Sub btnLoadXmlSchema_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles btnLoadXmlSchema.Click
    Dim dlg As FileDialog = New OpenFileDialog()
    dlg.Filter = "xml files (*.xml)|*.xml|All files (*.*)|*.*"
    If dlg.ShowDialog() = DialogResult.OK Then
        Dim xr As XmlReader = New XmlTextReader(dlg.FileName)
        gridGroupingControl1.ApplyXmlSchema(xr);
        xr.Close()
    End If
End Sub
```

5. Create a third button named '**Reset**' which will reset the look and feel of the grid.

**[C#]**

```
// Reset Grid.  
private void reset_Click(object sender, System.EventArgs e)  
{  
    System.IO.MemoryStream stream2 = new  
System.IO.MemoryStream(stream.ToArray());  
    this.gridGroupingControl1.ApplyXmlSchema (new  
XmlTextReader(stream2));  
}
```

**[VB .NET]**

```
' Reset Grid.  
Private Sub reset_Click(ByVal sender As Object, ByVal e As  
System.EventArgs) Handles reset.Click  
    Dim stream2 As System.IO.MemoryStream = New  
System.IO.MemoryStream(stream.ToArray())  
    Me.gridGroupingControl1.ApplyXmlSchema (New XmlTextReader(stream2))  
End Sub
```

6. While running the sample, click the **Serialize** button and save the grid schema into an XML file. Then click **Reset** button to switch the grid to its default state. It removes all the appearance settings done in first step. You can also make changes in the TableDescriptor of the grid manually like rearranging columns through drag and drop so that after reloading the grid schema, you could notice that the entire grid schema has been serialized. Reloading will transform the grouping grid back to the state before serialization.

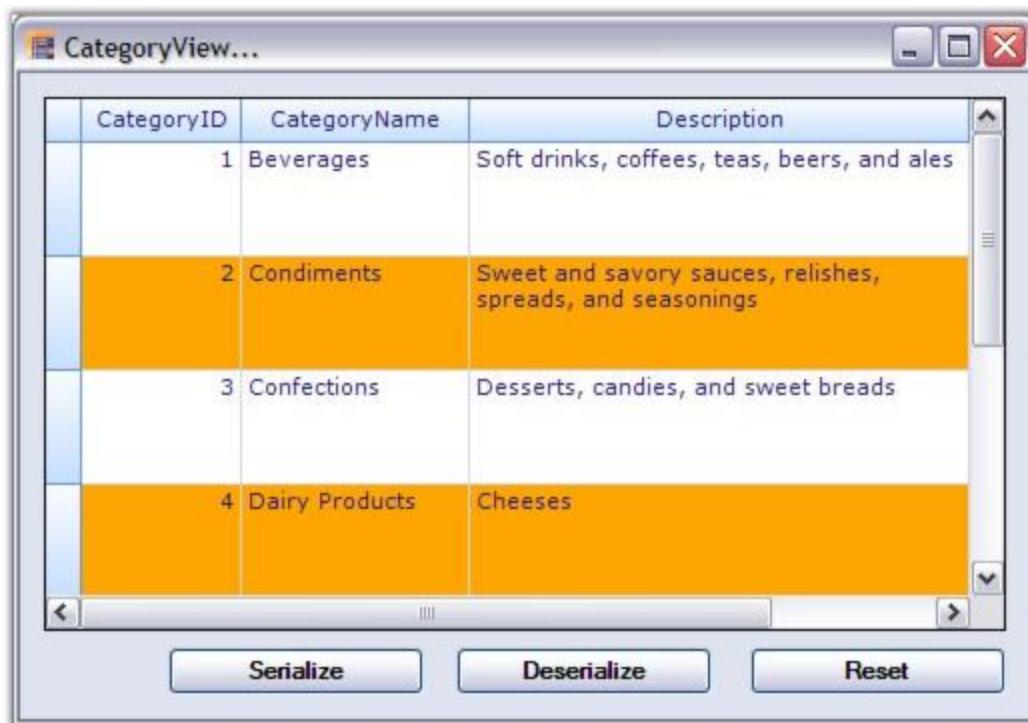


Figure 348: Startup Screen - Click Serialize to save Grid Schema

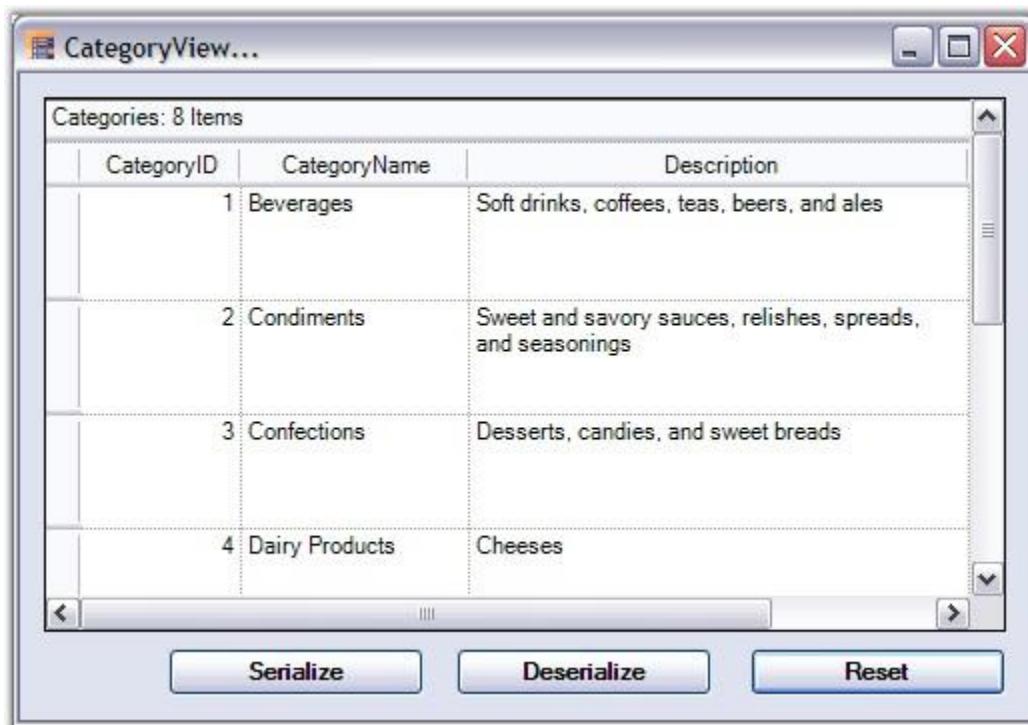


Figure 349: After Serialization, reset the Grid

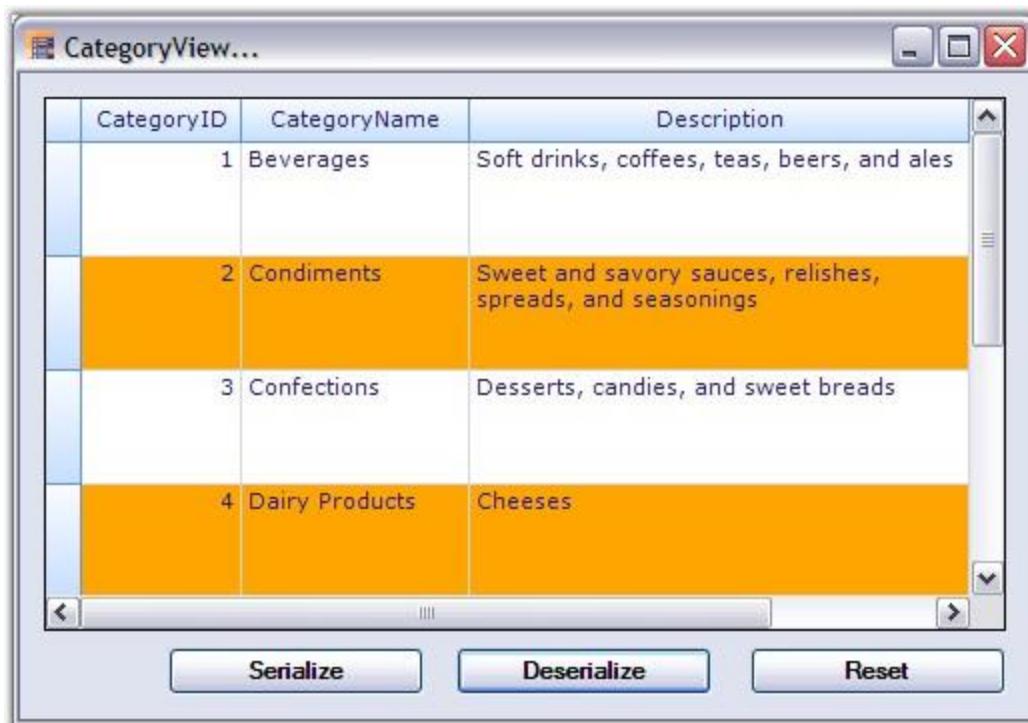


Figure 350: Restoring the Grid Schema by Deserialization



**Note:** For more details, refer the following browser samples:

<Install Location>\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Grouping.Windows\Samples\2.0\Serialization\XML.Serialization Demo

<Install Location>\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Grouping.Windows\Samples\2.0\Serialization\Employee View Xml Demo

#### Saving / Restoring Look and Feel Properties

You can save, look and feel properties in XML format. This will allow you to design a basic look and feel to use with all your Grid Grouping controls and then easily apply this look and feel to new grids at design-time or run-time.

It can be done in the following ways.

- Through Verbs

- Through Code

### Through Verbs

The verbs "Save Look and Feel" and "Choose Look and Feel" that are found at the bottom of the **property grid** of the Grid Grouping control will allow you to easily accomplish this task. Use the Save verb to save the Look and Feel properties of the current Grid Grouping control. Then use the Choose verb to apply the saved settings to a different control.

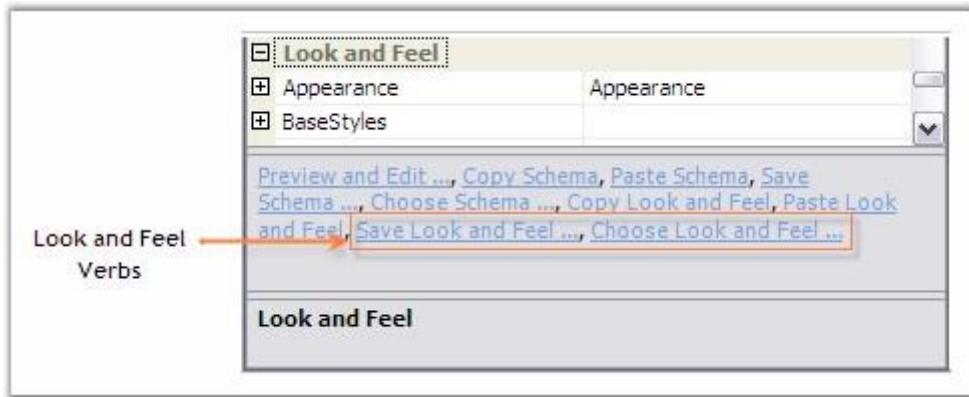


Figure 351: Look and Feel Customization Options

### Through Code

To apply the Look and Feel properties saved as XML at run-time, simply call `ApplyXmlLookandFeel` method. For example, the code below shows the code that is necessary to load such a file in the form's constructor.

#### [C#]

```
public Form1()
{
    System.Xml.XmlReader xr = new
System.Xml.XmlReader("BaseLandF.xml");
    this.gridGroupingControl1.ApplyXmlLookAndFeel(xr);
}
```

#### [VB .NET]

```
Public Sub New()
    Dim xr As System.Xml.XmlReader = New
System.Xml.XmlReader("BaseLandF.xml")
    Me.gridGroupingControl1.ApplyXmlLookAndFeel(xr)
```

```
End Sub
```

#### 4.3.4.8.2 Excel Export

**Export to Excel** is one of the most common functionalities that are required in the .Net world. The Essential Grid Control has in-built support for **Excel Export**. Users can download the data from the Grouping Grid control into an Excel spreadsheet for offline verification and/or computation. This can be achieved by making use of the **GroupingGridExcelConverter** class. This section will walk you through the conversion of the contents of the grid to an excel file as well as discuss the various converter options.

**GroupingGridExcelConverter** class derives from **GridExcelConverterBase**. It contains a number of methods that helps in exporting different components of the grouping grid. Its properties will let you control the export of the grid schema like styles and grid elements. You can be able to export Styles, RecordPlusMinus, GroupCaptionPlusMinus, Borders and PreviewRows as well.

#### ConverterOptions

Exporting of a Grouping Grid to Excel has two options: **Visible** and **Default**. **Visible** option will allow you to export only the visible contents of the grid whereas the **Default** option exports the entire elements of the grouping grid. Converter options are defined in **GridExcelConverter.ConverterOptions** enumeration.

The following table lists the properties offered by Grouping Grid Excel Converter. By setting these properties, you could be able to choose the elements you need to export.

Property	Description
ExportBorders	Specifies if borders should be exported.
ExportStyle	Indicates if style should be exported.
ExportGroupPlusMinus	Specifies if GridGroup should be exported as Excel Group.
ExportRecordPlusMinus	When true, the record with related tables will be exported as Excel Group.
ExportPreviewRows	When enabled, PreviewRows will be exported.
CaptionBackColor	Specifies the color to be used for Caption in the worksheet.

HeaderBackColor	Indicates the color to be used for Header in the worksheet.
-----------------	---

### Method

Grouping Grid Excel Converter control provides a method called GroupingGridToExcel. This is the method that does the conversion of grouping grid contents to an excel file. It accepts three parameters: grouping grid to be converted, filename of the destination worksheet and convert options.

### Syntax

#### [C#]

```
GroupingGridExcelConverterControl converter = new  
GroupingGridExcelConverterControl();  
converter.GroupingGridToExcel(this.gridGroupingControll, "Grid.xls",  
ConverterOptions.Visible);
```

#### [VB .NET]

```
Dim converter As GroupingGridExcelConverter = New  
GroupingGridExcelConverter()  
converter.GroupingGridToExcel(Me.gridGroupingControll, "Grid.xls",  
ConverterOptions.Visible)
```

### Events

QueryExportPreviewRowInfo is an event offered by the Grouping Grid Excel Converter control that aids in the conversion process. It occurs for each PreviewRow element before exporting the grid and lets you customize the preview row. It accepts two parameters: one contains the Element to export and the other is the GridStyleInfo object representing the style information.

### Syntax

#### [C#]

```
GroupingGridExcelConverter converter = new  
GroupingGridExcelConverter();  
  
converter.QueryExportPreviewRowInfo += new  
GroupingGridExcelConverterControl.GroupingGridExportPreviewRowQueryInfo  
EventHandler(converter_QueryExportPreviewRowInfo);
```

```
void converter_QueryExportPreviewRowInfo(object sender,
GroupingGridExportPreviewRowQueryEventArgs e)
{
}
```

**[VB .NET]**

```
Dim converter As GroupingGridExcelConverter = New
GroupingGridExcelConverter()

AddHandler converter.QueryExportPreviewRowInfo, AddressOf
converter_QueryExportPreviewRowInfo

Private Sub converter_QueryExportPreviewRowInfo(ByVal sender As Object,
ByVal e As GroupingGridExportPreviewRowQueryEventArgs)
End Sub
```

## Implementation

Here is an example that illustrates the conversion of a grouping grid to an Excel file.

1. Include the required namespaces.

**[C#]**

```
using Syncfusion.XlsIO;
using Syncfusion.GridExcelConverter;
using Syncfusion.GroupingExcelConverter;

using Syncfusion.Grouping;
using Syncfusion.Windows.Forms.Grid;
using Syncfusion.Windows.Forms.Grid.Grouping;
```

**[VB .NET]**

```
Imports Syncfusion.XlsIO
Imports Syncfusion.GridExcelConverter
Imports Syncfusion.GroupingExcelConverter

Imports Syncfusion.Grouping
Imports Syncfusion.Windows.Forms.Grid
Imports Syncfusion.Windows.Forms.Grid.Grouping
```

2. Setup an hierarchical datasource.

[C#]

```
private int numberParentRows = 50;
private int numberChildRows = 200;
private int numberGrandChildRows = 500;

private DataTable GetParentTable()
{
    DataTable dt = new DataTable("ParentTable");
    dt.Columns.Add(new DataColumn("parentID"));
    dt.Columns.Add(new DataColumn("ParentName"));
    dt.Columns.Add(new DataColumn("GroupID"));

    Random r = new Random();
    for (int i = 0; i < numberParentRows; i++)
    {
        DataRow dr = dt.NewRow();
        dr[0] = i;
        dr[1] = string.Format("parentName{0}", i);
        dr[2] = r.Next(99, 111);
        dt.Rows.Add(dr);
    }
    return dt;
}

private DataTable GetChildTable()
{
    DataTable dt = new DataTable("ChildTable");

    dt.Columns.Add(new DataColumn("childID"));
    dt.Columns.Add(new DataColumn("Name"));
    dt.Columns.Add(new DataColumn("ParentID"));
    dt.Columns.Add(new DataColumn("ChildGroupID"));

    Random r = new Random();
    for (int i = 0; i < numberChildRows; i++)
    {
        DataRow dr = dt.NewRow();
        dr[0] = i.ToString();
        dr[1] = string.Format("ChildName{0}", i);
        dr[2] = (i % numberParentRows).ToString();
        dr[3] = r.Next(994, 1006);
        dt.Rows.Add(dr);
    }
}
```

```
    return dt;
}
```

**[VB.NET]**

```
Private numberParentRows As Integer = 50
Private numberChildRows As Integer = 200
Private numberGrandChildRows As Integer = 500

Private Function GetParentTable() As DataTable
    Dim dt As DataTable = New DataTable("ParentTable")

    dt.Columns.Add(New DataColumn("parentID"))
    dt.Columns.Add(New DataColumn("ParentName"))
    dt.Columns.Add(New DataColumn("GroupID"))

    Dim r As Random = New Random()
    Dim i As Integer = 0
    Do While i < numberParentRows
        Dim dr As DataRow = dt.NewRow()
        dr(0) = i
        dr(1) = String.Format("parentName{0}", i)
        dr(2) = r.Next(99, 111)
        dt.Rows.Add(dr)
        i += 1
    Loop
    Return dt
End Function

Private Function GetChildTable() As DataTable
    Dim dt As DataTable = New DataTable("ChildTable")

    dt.Columns.Add(New DataColumn("childID"))
    dt.Columns.Add(New DataColumn("Name"))
    dt.Columns.Add(New DataColumn("ParentID"))
    dt.Columns.Add(New DataColumn("ChildGroupID"))

    Dim r As Random = New Random()
    Dim i As Integer = 0

    Do While i < numberChildRows
        Dim dr As DataRow = dt.NewRow()
        dr(0) = i.ToString()
        dr(1) = String.Format("ChildName{0}", i)
        dr(2) = (i Mod numberParentRows).ToString()
        i += 1
    Loop
    Return dt
End Function
```

```
    dr(3) = r.Next(994, 1006)
    dt.Rows.Add(dr)
    i += 1
Loop
Return dt
End Function
```

3. Setup a grouping grid and load it with data from a data source. Customize the grid by adding preview rows, groups, and the like as required.

**[C#]**

```
DataTable parentTable = GetParentTable();
DataTable childTable = GetChildTable();

// Add Summary row to parent table.
GridSummaryColumnDescriptor gridSummaryColumnDescriptor = new
GridSummaryColumnDescriptor();
gridSummaryColumnDescriptor.DisplayColumn = "GroupID";
gridSummaryColumnDescriptor.Format = " {Count} Records.";
gridSummaryColumnDescriptor.Name = "SummaryColumn";
gridSummaryColumnDescriptor.SummaryType = SummaryType.Count;
this.gridGroupingControl1.TableDescriptor.SummaryRows.Add(new
GridSummaryRowDescriptor("SummaryRow", new
GridSummaryColumnDescriptor[] {gridSummaryColumnDescriptor}));

// Manually specify relations in grouping engine.
GridRelationDescriptor parentToChildRelationDescriptor = new
GridRelationDescriptor();
parentToChildRelationDescriptor.ChildTableName = "MyChildTable"; // same
as SourceListSetEntry.Name for childTable
parentToChildRelationDescriptor.RelationKind =
RelationKind.RelatedMasterDetails;
parentToChildRelationDescriptor.RelationKeys.Add("parentID",
"ParentID");

// Add Summary Row to child table.
gridSummaryColumnDescriptor = new GridSummaryColumnDescriptor();
gridSummaryColumnDescriptor.DisplayColumn = "ChildGroupID";
gridSummaryColumnDescriptor.Format = " {Count} Records.";
gridSummaryColumnDescriptor.Name = "SummaryColumn";
gridSummaryColumnDescriptor.SummaryType = SummaryType.Count;
parentToChildRelationDescriptor.ChildTableDescriptor.SummaryRows.Add(ne
w GridSummaryRowDescriptor("SummaryRow", new
Syncfusion.Windows.Forms.Grid.Grouping.GridSummaryColumnDescriptor[]
{gridSummaryColumnDescriptor}));
```

```

// Add relation to parent table.
gridGroupingControl1.TableDescriptor.Relations.Add(parentToChildRelationDescriptor);

// Register any DataTable/IList with SourceListSet, so that
// RelationDescriptor can resolve the name
this.gridGroupingControl1.Engine.SourceListSet.Add("MyParentTable",
parentTable);
this.gridGroupingControl1.Engine.SourceListSet.Add("MyChildTable",
childTable);

this.gridGroupingControl1.DataSource = parentTable;
this.gridGroupingControl1.ShowGroupDropArea = true;
this.gridGroupingControl1.AddGroupDropArea("MyChildTable");

// The TrackWidthOfParentColumn property of a column descriptor ensures
// that
// columns are aligned and synchronized.

this.gridGroupingControl1.TableDescriptor.Columns[0].Width = 200;
this.gridGroupingControl1.TableDescriptor.Columns[1].Width = 150;
this.gridGroupingControl1.TableDescriptor.Columns[2].Width = 150;

// Synchronize width of columns in child record with width of column in
parent record.
for(int n=0;n<3;n++)
parentToChildRelationDescriptor.ChildTableDescriptor.Columns[n].TrackWidthOfParentColumn =
gridGroupingControl1.TableDescriptor.Columns[n].Name;

this.gridGroupingControl1.TableDescriptor.GroupedColumns.Add("GroupID");
;
this.gridGroupingControl1.TableOptions.ShowRecordPreviewRow = true;
this.gridGroupingControl1.ChildGroupOptions.ShowGroupPreview = true;

this.gridGroupingControl1.TableDescriptor.Columns["GroupID"].Appearance
.AnyHeaderCell.HorizontalAlignment = GridHorizontalAlignment.Right;
this.gridGroupingControl1.TableDescriptor.Columns["GroupID"].Appearance
.AnyHeaderCell.VerticalAlignment = GridVerticalAlignment.Bottom;
this.gridGroupingControl1.Appearance.AnySummaryCell.Interior = new
BrushInfo(Color.FromArgb(255, 231, 162));

// Hook up this event to handle preview rows.
this.gridGroupingControl1.QueryCellStyleInfo += new
GridTableCellStyleInfoEventHandler(gridGroupingControl1_QueryCellStyleI
nfo);

Private Sub gridGroupingControl1_QueryCellStyleInfo(ByVal sender As

```

```
Object, ByVal e As GridTableViewCellEventArgs)
If e.TableCellIdentity.TableCellType =
GridTableCellType.RecordPreviewCell Then
    Dim el As Element = e.TableCellIdentity.DisplayElement
    e.Style.CellValue = "Preview notes for Record (" &
el.ParentTableDescriptor.Fields(0).Name & ":" &
el.ParentRecord.GetValue(el.ParentTableDescriptor.Fields(0).Name)
& ")"
End If
If e.TableCellIdentity.TableCellType =
GridTableCellType.GroupPreviewCell Then
    Dim el As Element = e.TableCellIdentity.DisplayElement
    e.Style.CellValue = "Preview notes for Group (" &
el.ParentGroup.Name & ":" & el.ParentGroup.Category.ToString() &
")"
End If
End Sub
```

**[VB.NET]**

```
Dim parentTable As DataTable = GetParentTable()
Dim childTable As DataTable = GetChildTable()

' Add Summary row to parent table.
Dim gridSummaryColumnDescriptor As New GridSummaryColumnDescriptor()
gridSummaryColumnDescriptor.DisplayColumn = "GroupID"
gridSummaryColumnDescriptor.Format = " {Count} Records."
gridSummaryColumnDescriptor.Name = "SummaryColumn"
gridSummaryColumnDescriptor.SummaryType = SummaryType.Count
Me.gridGroupingControll1.TableDescriptor.SummaryRows.Add(New
GridSummaryRowDescriptor("SummaryRow", New
GridSummaryColumnDescriptor() {gridSummaryColumnDescriptor})) 

' Manually specify relations in grouping engine.
Dim parentToChildRelationDescriptor As New GridRelationDescriptor()
parentToChildRelationDescriptor.ChildTableName = "MyChildTable" ' same
as SourceListSetEntry.Name for childTable
parentToChildRelationDescriptor.RelationKind =
RelationKind.RelatedMasterDetails
parentToChildRelationDescriptor.RelationKeys.Add("parentID",
"ParentID")

' Add Summary Row to child table.
gridSummaryColumnDescriptor = New GridSummaryColumnDescriptor()
gridSummaryColumnDescriptor.DisplayColumn = "ChildGroupID"
gridSummaryColumnDescriptor.Format = " {Count} Records."
gridSummaryColumnDescriptor.Name = "SummaryColumn"
```

```
gridSummaryColumnDescriptor.SummaryType = SummaryType.Count
parentToChildRelationDescriptor.ChildTableDescriptor.SummaryRows.Add(New
GridSummaryRowDescriptor("SummaryRow", New
Syncfusion.Windows.Forms.Grid.Grouping.GridSummaryColumnDescriptor()
{gridSummaryColumnDescriptor}))

' Add relation to parent table.
gridGroupingControl1.TableDescriptor.Relations.Add(parentToChildRelatio
nDescriptor)

' Register any DataTable/IList with SourceListSet, so that
RelationDescriptor can resolve the name
Me.gridGroupingControl1.Engine.SourceListSet.Add("MyParentTable",
parentTable)
Me.gridGroupingControl1.Engine.SourceListSet.Add("MyChildTable",
childTable)

Me.gridGroupingControl1.DataSource = parentTable
Me.gridGroupingControl1.ShowGroupDropArea = True
Me.gridGroupingControl1.AddGroupDropArea("MyChildTable")

' The TrackWidthOfParentColumn property of a column descriptor ensures
that
' columns are aligned and synchronized.

Me.gridGroupingControl1.TableDescriptor.Columns(0).Width = 200
Me.gridGroupingControl1.TableDescriptor.Columns(1).Width = 150
Me.gridGroupingControl1.TableDescriptor.Columns(2).Width = 150

' Synchronize width of columns in child record with width of column in
parent record.
For n As Integer = 0 To 2
parentToChildRelationDescriptor.ChildTableDescriptor.Columns(n).TrackWi
dthOfParentColumn =
gridGroupingControl1.TableDescriptor.Columns(n).Name
Next n

Me.gridGroupingControl1.TableDescriptor.GroupedColumns.Add("GroupID")
Me.gridGroupingControl1.TableOptions.ShowRecordPreviewRow = True
Me.gridGroupingControl1.ChildGroupOptions.ShowGroupPreview = True

Me.gridGroupingControl1.TableDescriptor.Columns("GroupID").Appearance.A
nyHeaderCell.HorizontalAlignment = GridHorizontalAlignment.Right
Me.gridGroupingControl1.TableDescriptor.Columns("GroupID").Appearance.A
nyHeaderCell.VerticalAlignment = GridVerticalAlignment.Bottom
Me.gridGroupingControl1.Appearance.AnySummaryCell.Interior = New
BrushInfo(Color.FromArgb(255, 231, 162))
```

```
' Hook up this event to handle preview rows.  
AddHandler gridGroupingControll.QueryCellStyleInfo, AddressOf  
gridGroupingControll_QueryCellStyleInfo  
  
private void gridGroupingControll_QueryCellStyleInfo(Object  
sender,GridTableCellStyleEventArgs e)  
    If e.TableCellIdentity.TableCellType =  
    GridTableCellType.RecordPreviewCell Then  
        Dim el As Element = e.TableCellIdentity.DisplayElement  
        e.Style.CellValue = "Preview notes for Record (" &  
        el.ParentTableDescriptor.Fields(0).Name & ":" &  
        el.ParentRecord.GetValue(el.ParentTableDescriptor.Fields(0).Name  
        & ")" )  
    End If  
    If e.TableCellIdentity.TableCellType =  
    GridTableCellType.GroupPreviewCell Then  
        Dim el As Element = e.TableCellIdentity.DisplayElement  
        e.Style.CellValue = "Preview notes for Group (" &  
        el.ParentGroup.Name & ":" & el.ParentGroup.Category.ToString() &  
        ")" )  
End Sub
```

4. Set up a Grouping Grid Excel Converter and choose the elements you want to export by setting the appropriate properties.

**[C#]**

```
GroupingGridExcelConverterControl converter = new  
GroupingGridExcelConverterControl();  
converter.ExportBorders = true;  
converter.ExportGroupPlusMinus = true;  
converter.ExportPreviewRows = true;  
converter.ExportRecordPlusMinus = true;  
converter.ExportStyle = true;  
converter.HeaderBackColor = Color.Orange;  
converter.CaptionBackColor = Color.Lavender;
```

**[VB .NET]**

```
Dim converter As GroupingGridExcelConverterControl = New  
GroupingGridExcelConverterControl()  
converter.ExportBorders = True  
converter.ExportGroupPlusMinus = True  
converter.ExportPreviewRows = True  
converter.ExportRecordPlusMinus = True  
converter.ExportStyle = True  
converter.HeaderBackColor = Color.Orange
```

```
converter.CaptionBackColor = Color.Lavender
```

5. Handle the `QueryExportPreviewRowInfo` event to customize the preview rows before export.

**[C#]**

```
converter.QueryExportPreviewRowInfo += new  
GroupingGridExcelConverterControl.GroupingGridExportPreviewRowQueryInfo  
EventHandler(converter_QueryExportPreviewRowInfo);  
  
void converter_QueryExportPreviewRowInfo(object sender,  
GroupingGridExportPreviewRowQueryInfoEventArgs e)  
{  
    if (e.Element.Kind == DisplayElementKind.GroupPreview)  
    {  
        Element el = e.Element;  
        e.Style.CellValue = "Preview notes for Group (" +  
el.ParentGroup.Name + ": " + el.ParentGroup.Category.ToString() + ")";  
        e.Style.BackColor = Color.MistyRose;  
        e.Handled = true;  
    }  
    if (e.Element.Kind == DisplayElementKind.RecordPreview)  
    {  
        Element el = e.Element;  
        e.Style.CellValue = "Preview notes for Record (" +  
el.ParentTableDescriptor.Fields[0].Name + ": " +  
el.ParentRecord.GetValue(el.ParentTableDescriptor.Fields[0].Name) +  
")";  
        e.Style.BackColor = Color.MistyRose;  
        e.Handled = true;  
    }  
}
```

**[VB.NET]**

```
AddHandler converter.QueryExportPreviewRowInfo, AddressOf  
converter_QueryExportPreviewRowInfo)  
  
Private Sub converter_QueryExportPreviewRowInfo(ByVal sender As Object,  
ByVal e As GroupingGridExportPreviewRowQueryInfoEventArgs)  
    If e.Element.Kind = DisplayElementKind.GroupPreview Then  
        Dim el As Element = e.Element  
        e.Style.CellValue = "Preview notes for Group (" &  
el.ParentGroup.Name & ": " & el.ParentGroup.Category.ToString() &  
")"  
        e.Style.BackColor = Color.MistyRose
```

```
    e.Handled = True
End If
If e.Element.Kind = DisplayElementKind.RecordPreview Then
    Dim el As Element = e.Element
    e.Style.CellValue = "Preview notes for Record (" &
    el.ParentTableDescriptor.Fields(0).Name & ":" &
    el.ParentRecord.GetValue(el.ParentTableDescriptor.Fields(0).Name)
    & ")"
    e.Style.BackColor = Color.MistyRose
    e.Handled = True
End If
End Sub
```

6. Export the grouping grid to an Excel file.

**[C#]**

```
converter.GroupingGridToExcel(this.gridGroupingControll,
"ExcelGrid.xls", ConverterOptions.Default);
```

**[VB.NET]**

```
converter.GroupingGridToExcel(Me.gridGroupingControll, "ExcelGrid.xls",
ConverterOptions.Default)
```

7. Here are screen shots showing grouping grid and the exported grid in Excel file.

ParentTable			GroupID
MyChildTable Drag a column header here to group by that column.			
ParentTable; 50 Items			
	parentID	ParentName	GroupID
+	GroupID: 100 - 2 Items		
*	1	parentName1	100
	Preview notes for Record (parentID: 1)		
+	6	parentName6	100
	Preview notes for Record (parentID: 6)		
	2 Records.		
+	GroupID: 101 - 4 Items		
	Preview notes for Group (GroupID: 101)		
+	GroupID: 102 - 6 Items		
	Preview notes for Group (GroupID: 102)		

*Figure 352: Grouping Grid*

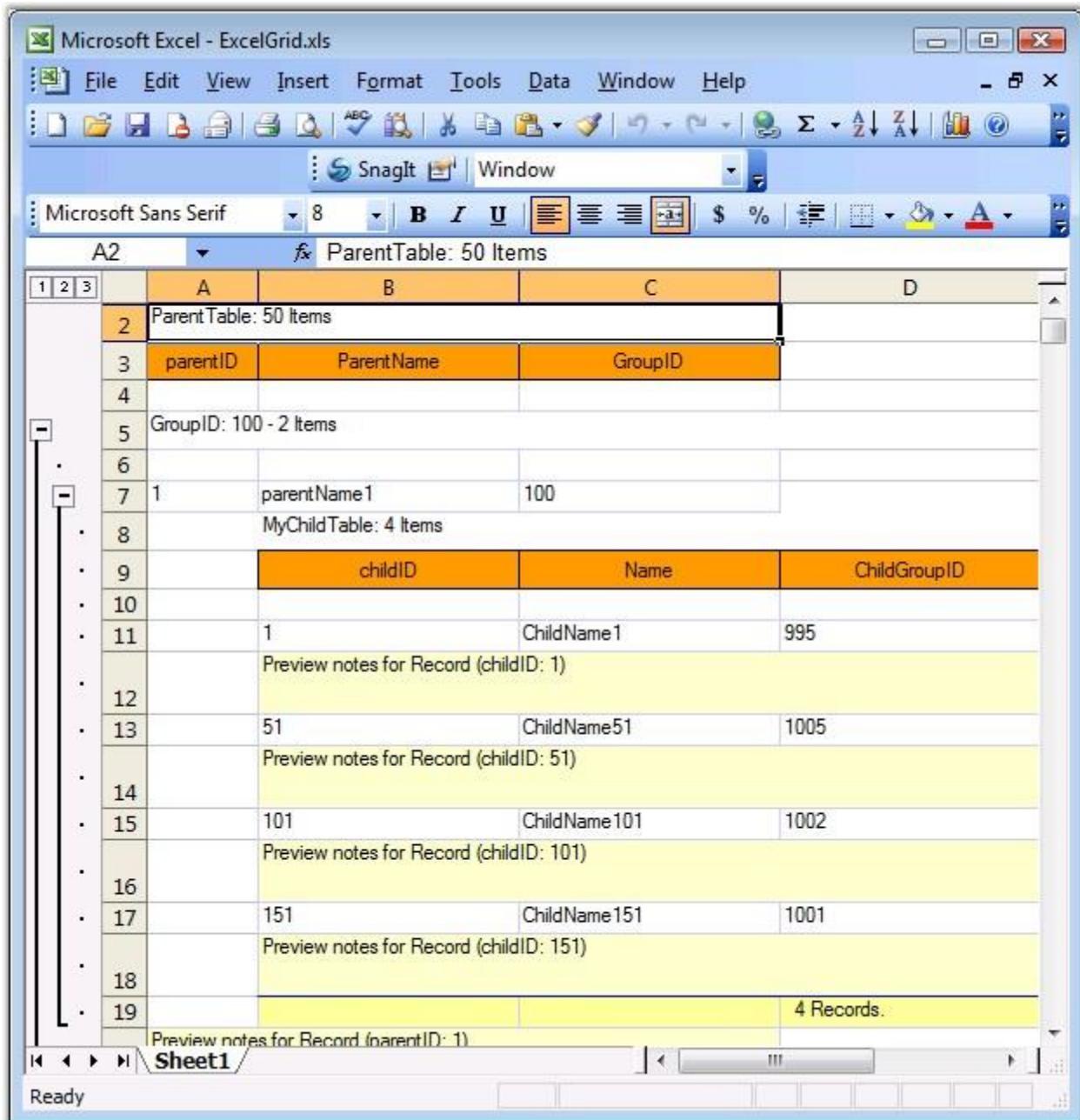


Figure 353: Grid In Excel



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Grouping.Windows\Samples\2.0\Export\MS Excel Export Demo

#### 4.3.4.8.3 Word Converter

Export to Word is one of the most common functionalities that are required in the .NET world. The Essential Grid Control has in-built support for Word Export. Users can download the data from the Grouping Grid control into a Word document for offline verification and/or computation. This can be achieved by making use of the GroupingGridWordConverter class. This section will walk you through the conversion of the contents of the grid to a word file as well as discuss the various converter options.

GroupingGridWordConverter class derives from GridWordConverterBase. It contains a number of methods that helps in exporting different components of the grouping grid. You can be able to export NestedTables as well.

The following table lists the properties offered by Grouping Grid Word Converter. By setting these properties, you could be able to choose the elements you need to export.

Property	Description
ShowHeader	Specifies if header should be displayed.
ShowFooter	Indicates if footer should be displayed.

#### Method

Grouping Grid Word Converter control provides a method called GroupingGridToWord. This is the method that does the conversion of grouping grid contents to a word file. It accepts two parameters: grouping grid to be converted and filename of the destination word document.

#### Syntax

##### [C#]

```
GroupingGridWordConverter converter = new GroupingGridWordConverter();
converter.GroupingGridToWord("Grid.doc", this.gridGroupingControl1);
```

##### [VB .NET]

```
Dim converter As GroupingGridWordConverter = New GroupingGridWordConverter()
converter.GroupingGridToWord("Grid.doc", Me.gridGroupingControl1)
```

#### Events

DrawHeader and DrawFooter are the events offered by the Grouping Grid Word Converter that aids in adding as well as customizing the header and footer in the destination word document.

### Sample Output

Below images depicts the conversion of grid content to a word file.

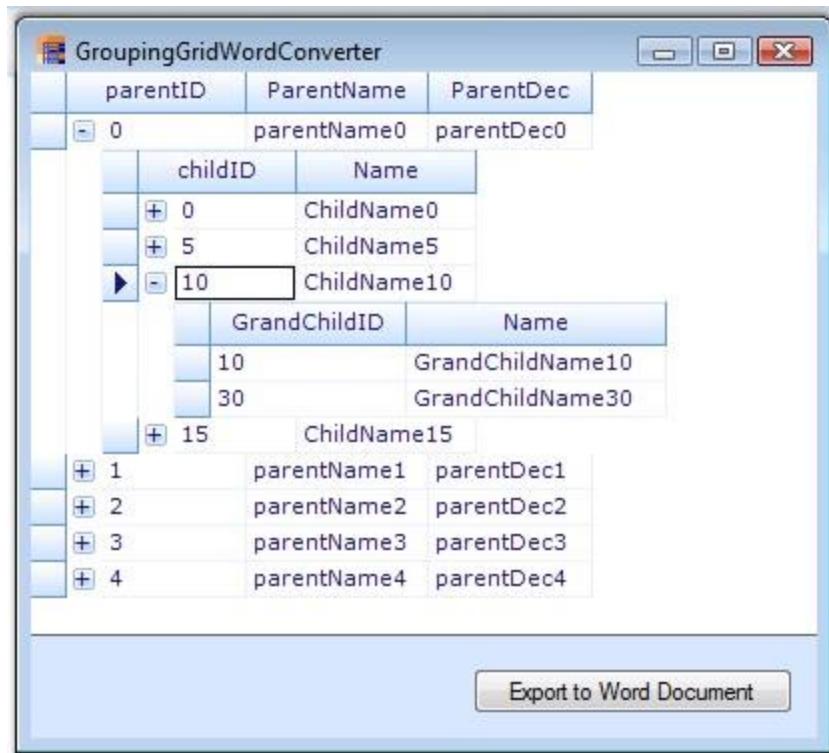
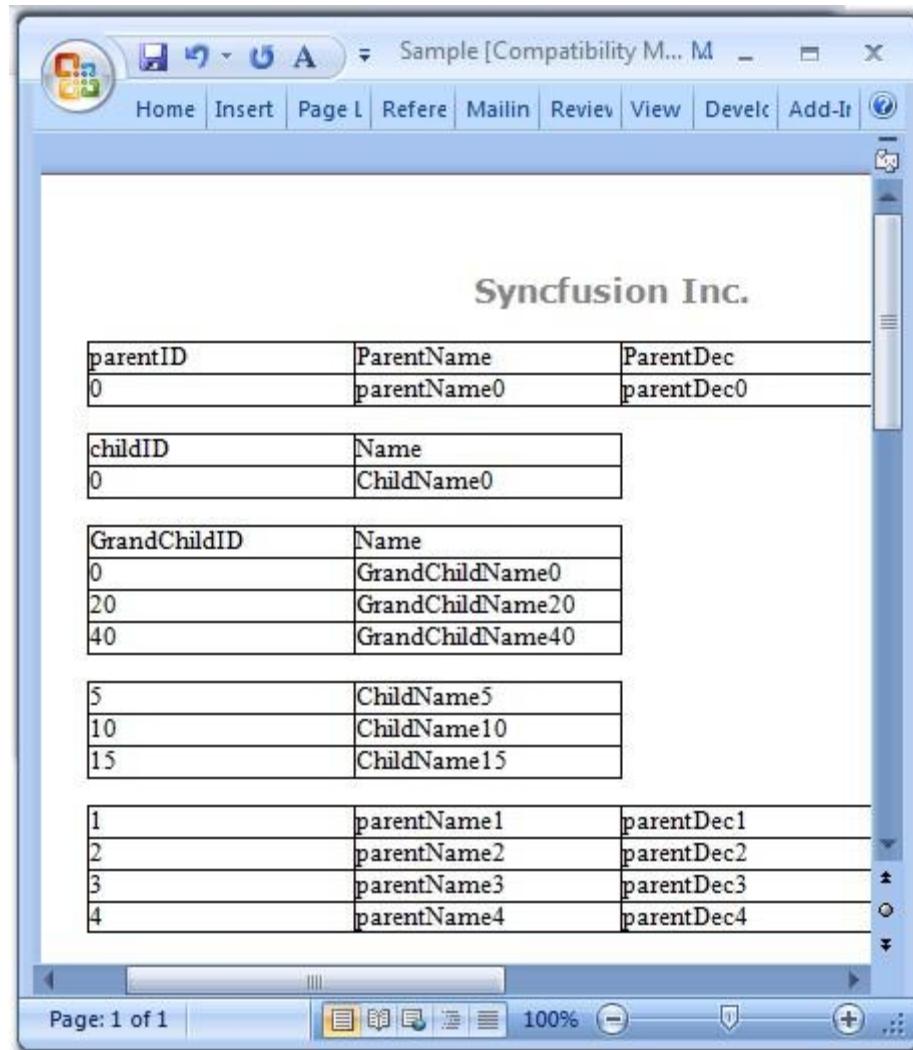


Figure 354: Grid to be Exported



*Figure 355: Grid Exported to a Word File*



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Export\Word Converter Demo

#### 4.3.4.8.4 PDF Converter

##### PDF Export

Essential Grid Grouping control supports conversion of grid contents to a PDF file. Users can convert data from the Grid Grouping control into a PDF document using the GridPDFConverter class. PDF libraries are used to support the conversion of grid content to a PDF page.

To ensure the conversion of grid data to PDF document, the following dll files should be added, along with the default dll files in the reference folder:

- Syncfusion.Pdf.Base
- Syncfusion.GridHelperClasses.Windows

The **ExportToPdf** method should be used to export the grid to a PDF file.

The following code example illustrates the conversion of grid data to PDF document.

**[C#]**

```
GridPDFConverter pdfConvertor = new GridPDFConverter();
pdfConvertor.ExportToPdf("Sample.pdf", this.gridGroupingControl1.TableControl);

// Launching the PDF file by using the default Application [Acrobat Reader].
System.Diagnostics.Process.Start("Sample.pdf");
```

**[VB .NET]**

```
Dim pdfConvertor As GridPDFConverter = New GridPDFConverter()
pdfConvertor.ExportToPdf("Sample.pdf", Me.gridGroupingControl1.TableControl)

' Launching the PDF file by using the default Application [Acrobat Reader].
System.Diagnostics.Process.Start("Sample.pdf")
```

**PDF Export Demo**

**Customers** Drag a column header here to group by that column.

ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice
1	Chai	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35
6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars	25
7	Uncle Bob's Organic Dried Pears	3	7	12 - 1 lb pkgs.	30
8	Northwoods Cranberry Sauce	3	2	12 - 12 oz jars	40
9	Mishi Kobe Niku	4	6	18 - 500 g pkgs.	97
10	Ikura	4	8	12 - 200 ml jars	31
11	Queso Cabrales	5	4	1 kg pkg.	21
12	Queso Manchego La Pastora	5	4	10 - 500 g pkgs.	38
13	Konbu	6	8	2 kg box	6
14	Tofu	6	7	40 - 100 g pkgs.	23.25
15	Genen Shouyu	6	2	24 - 250 ml bottles	15.5
16	Pavlova	7	3	32 - 500 g boxes	17.45
17	Alice Mutton	7	6	20 - 1 kg tins	39
18	Carnarvon Tigers	7	8	16 kg pkg.	62.5
19	Teatime Chocolate Biscuits	8	3	10 boxes x 12 pieces	9.2
20	Sir Rodney's Marmalade	8	3	30 gift boxes	81

**Header Footer**

Show Header

Show Footer

**Margin**

Left: 1    Right: 1    Top: 1    Bottom: 1

**Export to PDF**

Figure 356: Grid Grouping Data

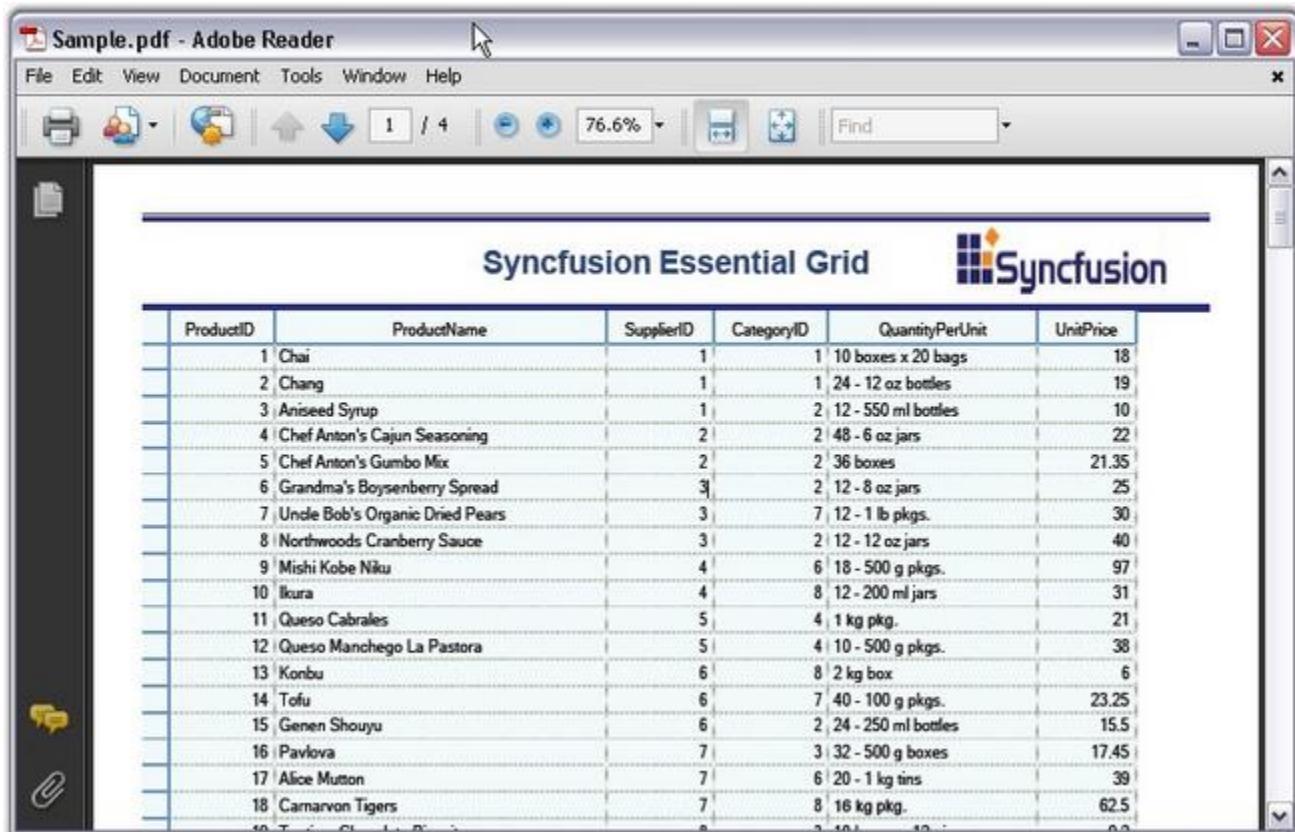


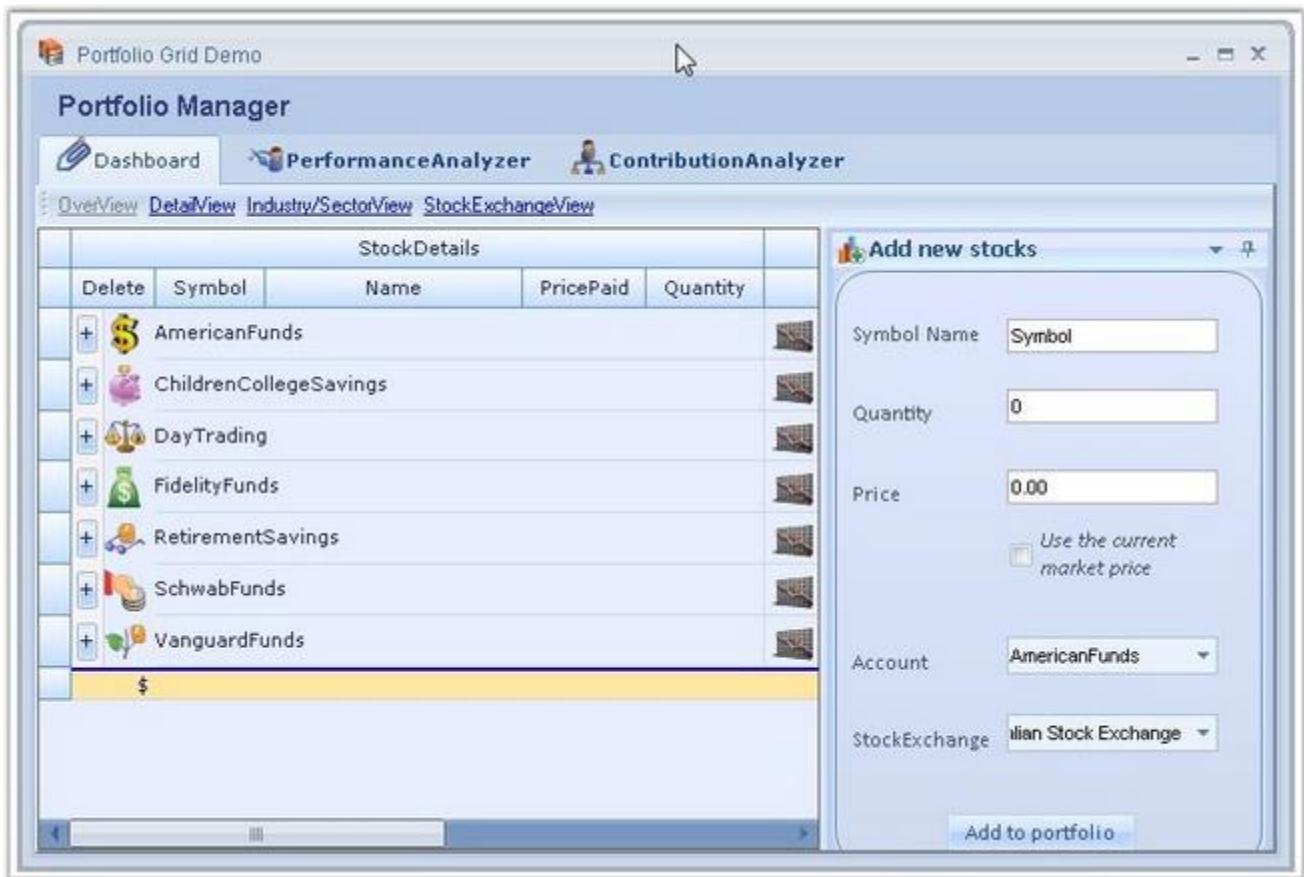
Figure 357: Grid Grouping Data converted to PDF

#### 4.3.4.9 Real Time Applications

This section provides information on the real-time applications of Essential Grouping Grid. It includes the following topic:

##### 4.3.4.9.1 Portfolio Grid

The user can simulate a Portfolio Manager application by using Essential Grid. It allows you to track all your investments and assets in stocks and mutual funds. It provides you with an insight on current market pricing that would help to plan your future investments.



*Figure 358: Portfolio Manager*

Refer the following sample browser for more details:

`<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Product Showcase\Portfolio Grid Demo`

**Example :** This sample comes with three modules whose features are explained below.

**Dashboard :** This is the place where you can track the status of all your investments. It comes along with a variety of views:

- **Overview** - Provides a consolidated view of your mutual fund holdings. You can get useful information on your investments such as rate of change per day, current market price, total returns, etc. You have options to add or delete desired stocks.
- **Detail View** - Provides a more detailed view, allowing you to get the stock details of the mutual funds.
- **Industry/Sector View** - Provides a consolidated view of your stock holdings.

- **Stock Exchange View** – Displays the details with respect to the individual stock exchange.

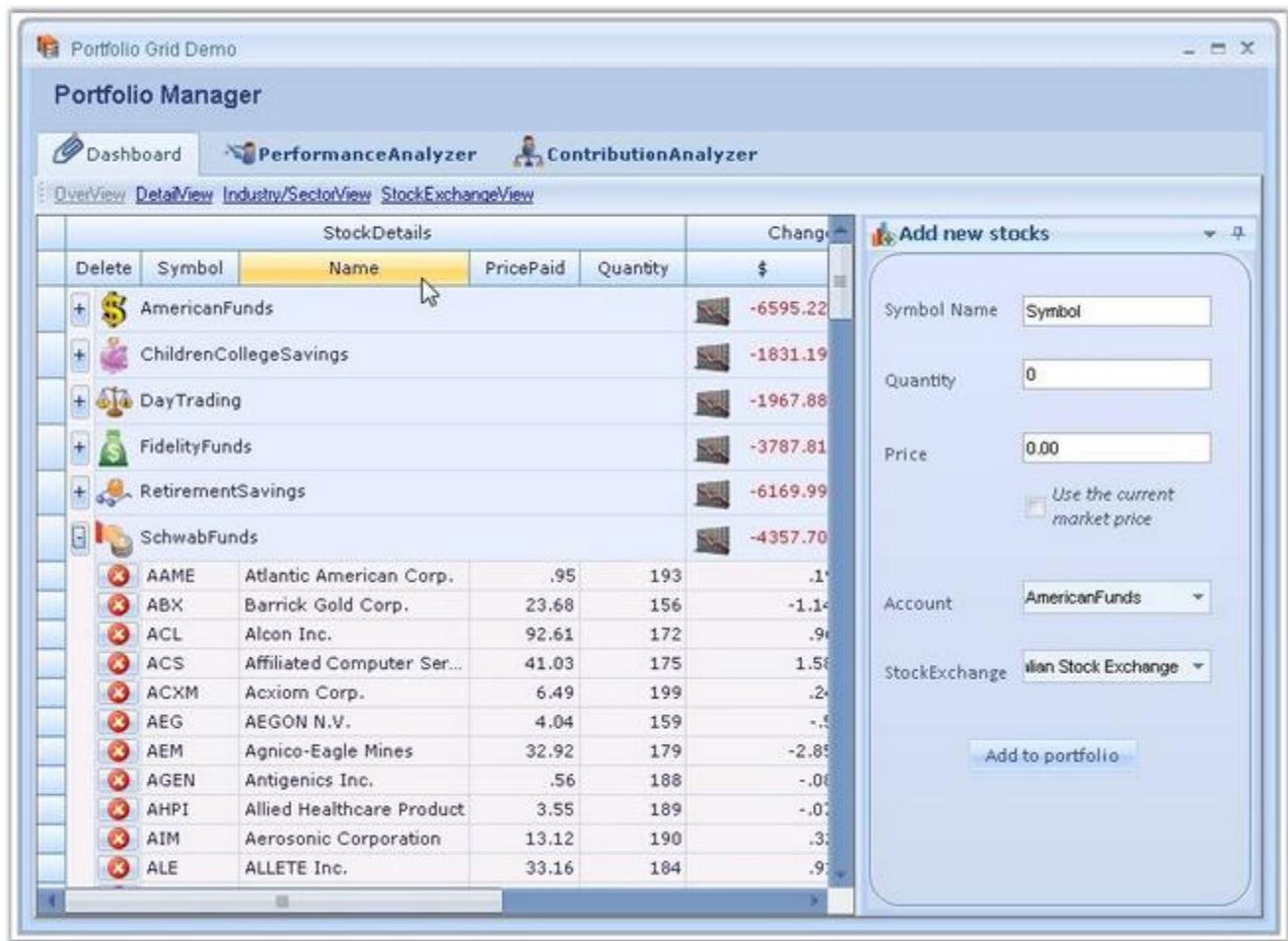
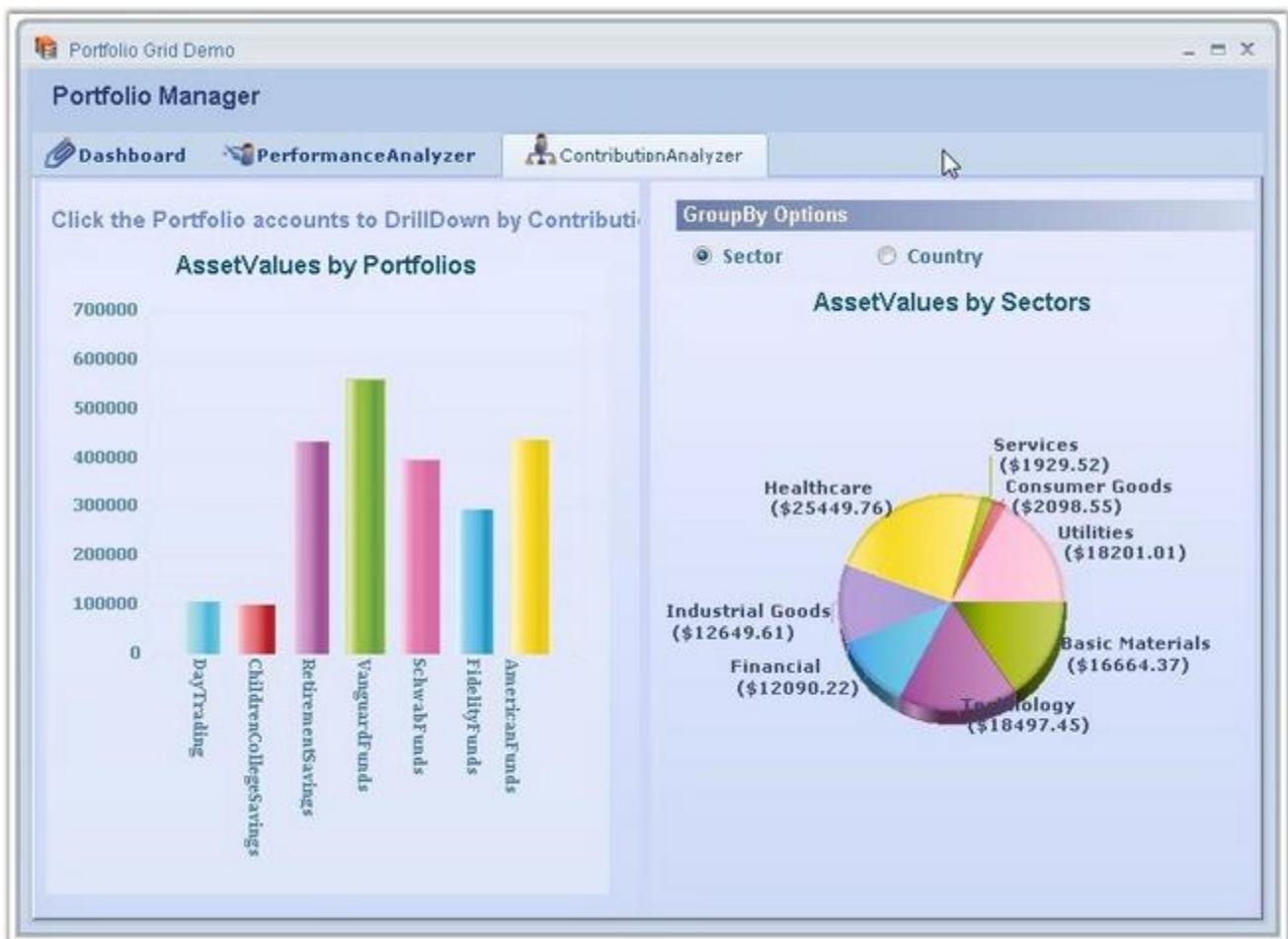


Figure 359: Portfolio Manager-Dashboard

**Contribution Analyzer** : This module illustrates the contributions of different stocks for every portfolio account in a graphical representation by using chart controls. Click the desired portfolio account on the left side chart to view its contributions.



*Figure 360: Portfolio Manager-Contribution Analyzer*

**Performance Analyzer :** This module does a performance analysis and displays the market history for the top three large cap accounts. You have options to view the history for last three months, last six months, and last year.

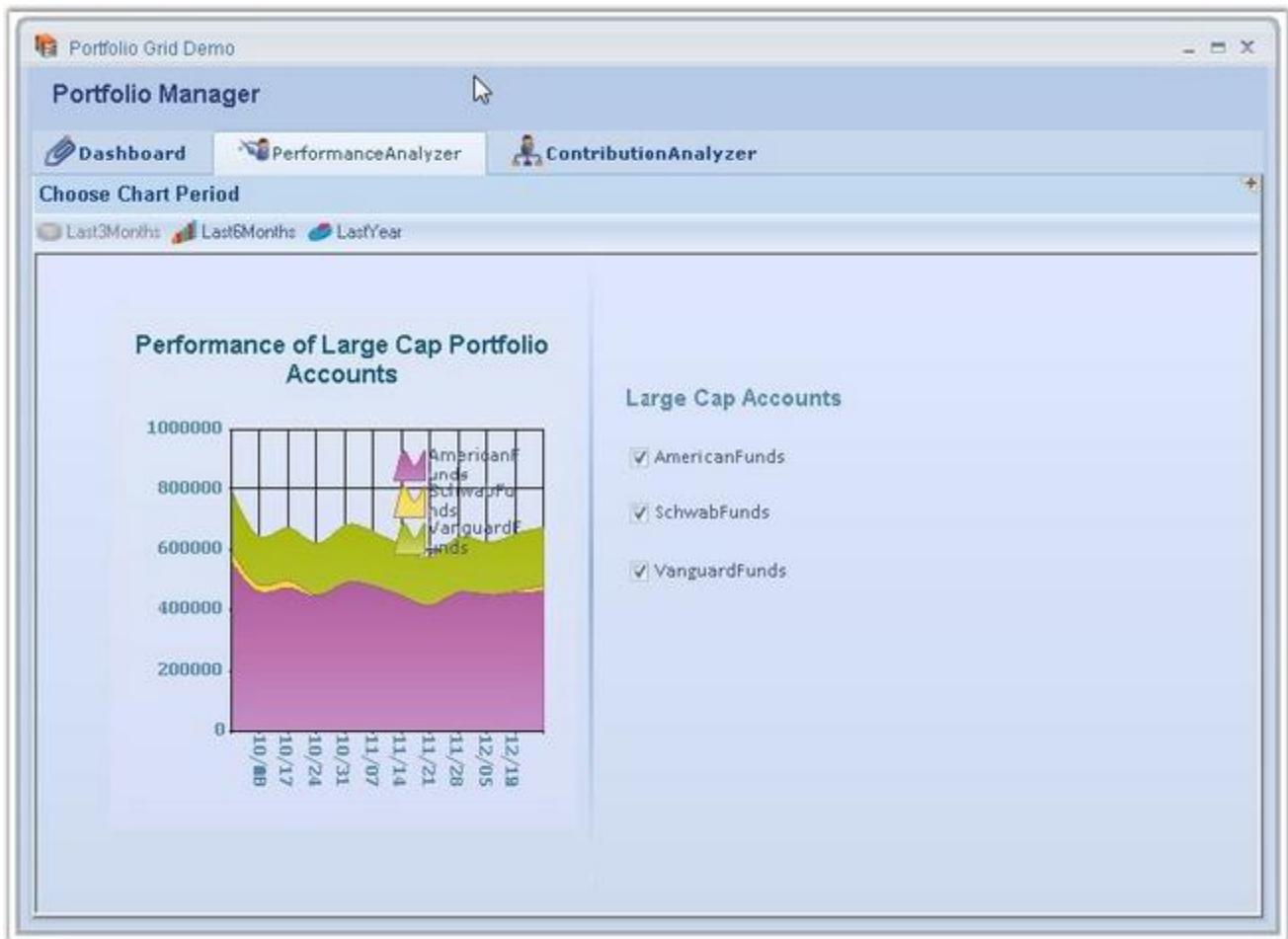


Figure 361: Portfolio Manager-Performance Analyzer

#### 4.3.4.9.2 Outlook 2007 Demo

You can implement Microsoft Outlook 2007-like interface by using Essential Grid Grouping control and other components from Essential Studio. Essential Grid has been extended to support all the Office 2007 themes (Blue, Black, and Silver) with the look and feel of Outlook 2007.

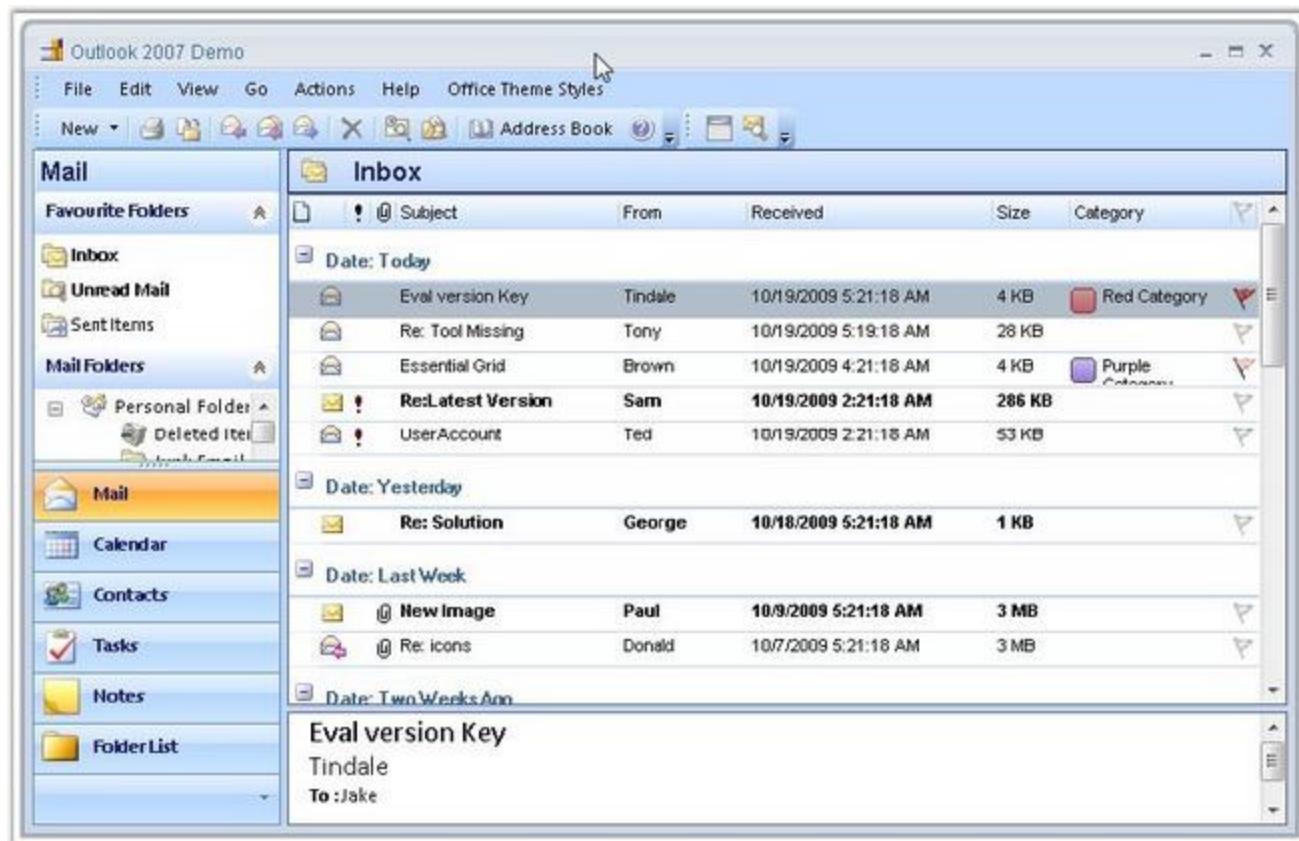


Figure 362: Microsoft Outlook 2007-like Interface implemented by using Grid Grouping Control

Refer the following browser sample for more details:

<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Product Showcase\Outlook 2007 Demo

**Example:** Following are the major features of the sample.

### Features

- Office 2007 themes - Blue, Black, and Silver to see the effects of the application.
- The grouping of columns in the mails display area. Right click on the column headers and select ArrangeBy in the column.
- Custom selection navigation with all arrow keys.
- Custom grouping for dates, size, priority, and flag fields.
- Dynamic formatting when narrowly switching between column-set view and normal view.
- Events when deleting records.

- Customized current view.
- Outlook appearance and functionalities.

#### **4.3.4.10 Grid Designer**

The Grid Grouping control has strong designer support. You can control all aspects of the grid's appearance through the designer. Additional commands (verbs) will let you save layouts and restore them. You can also use a preview feature that will let you load data into your control and then further set the Grid Grouping control properties that can be persisted as design-time properties.

Clicking the Preview and Edit verb will allow you to view the Grid Grouping control, populated with data along with a companion property grid as seen in the picture below. It also displays help description for the properties that are being selected. You can use the property grid to change the Grid Grouping control's properties and see the effect immediately upon the populated control. When you close the preview, you will have the option of saving any changed properties to the property grid in the designer. We will use this Preview and Edit support to see the effects of setting the various TableOption properties.

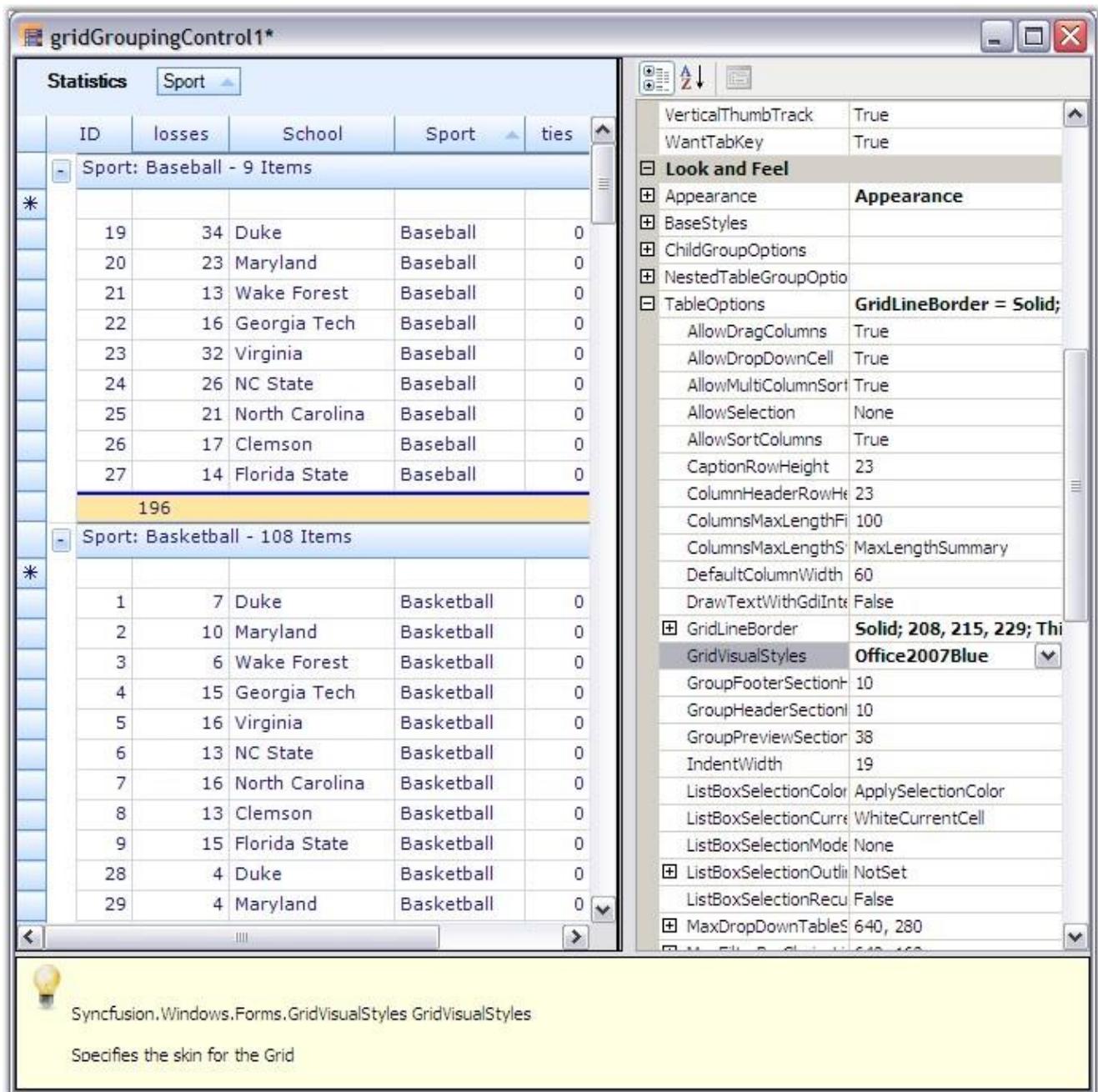


Figure 363: Preview and Edit Support

Grid Designer presents populated Grid Grouping control along with a property grid listing out the related properties. It also includes an integrated help feature to display a brief description on the property selected. You will be able to set any kind of properties using the designer so that you could see the results immediately. Here is a brief discussion on how to work with grid elements through the designer.

## Grouping

Designer provides full drag/drop capability so that you could be able to group the records by dragging a column header and dropping it into the GroupDropArea, provided the GroupDropArea is enabled by setting **ShowGroupDropArea** to true. Likewise you can group the data against any number of columns across the tables when multiple nested tables are used.



Figure 364: Dragging a column header to group grid by that Column

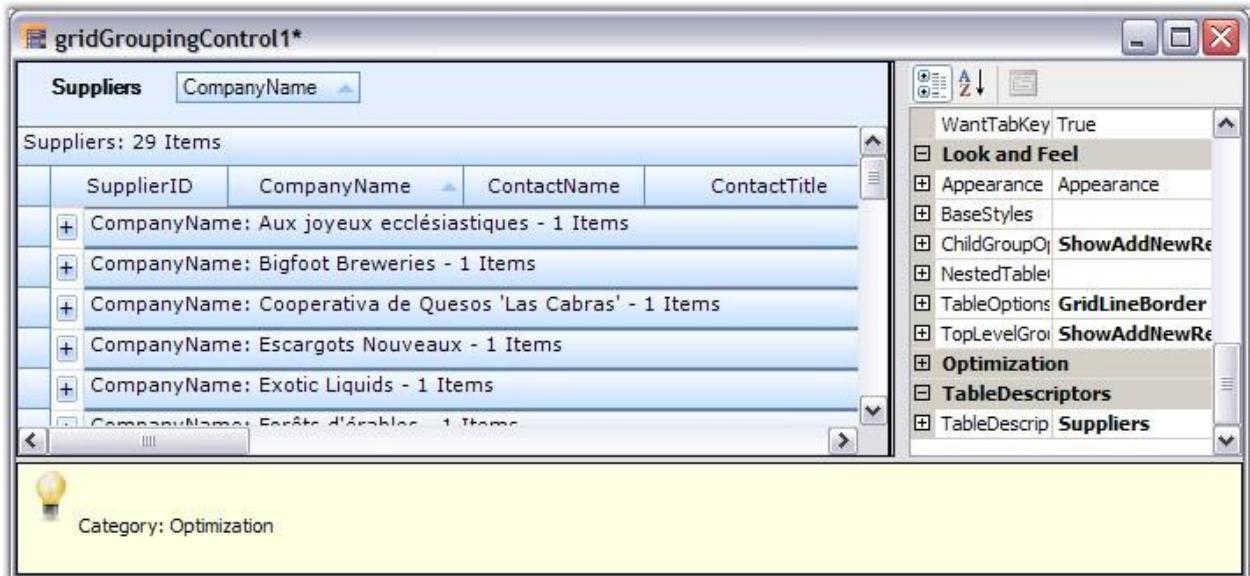


Figure 365: Grouped Grid

You can also use **TableDescriptor.GroupedColumns** property to add groups where you need to specify the column names based on which the table has to be grouped.

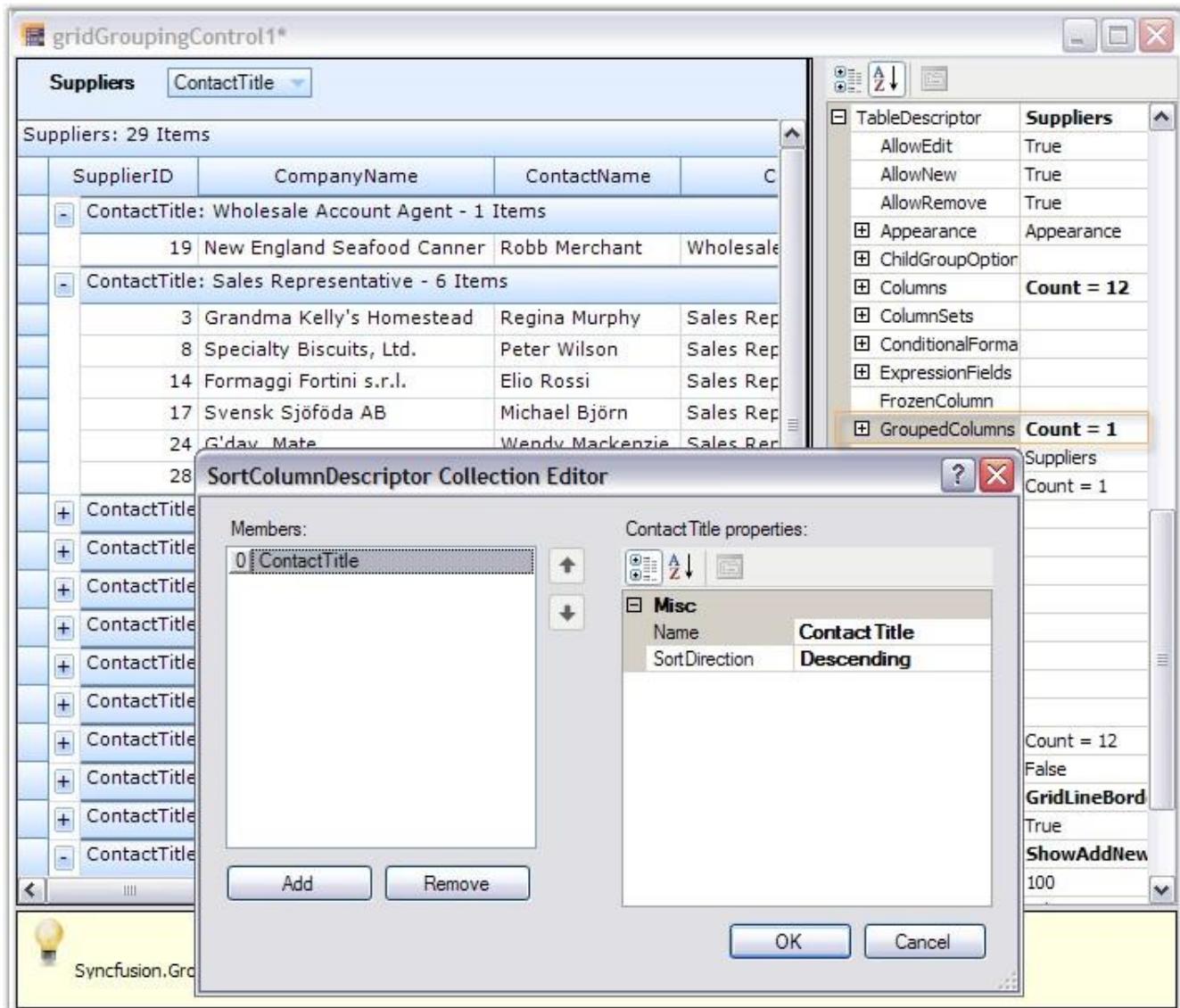


Figure 366: Grouping Columns by adding Groups

## Sorting

Sorting can be done on the table data by simply clicking the desired column header by which the values need to be sorted. Once sorting is done, the grouping grid displays a ListSortIcon in the respective column header to indicate the Sort Direction. You could also make use of the **TableDescriptor.SortedColumns** property to perform sorting on table data wherein you need to provide the column to be sorted and a sort order.

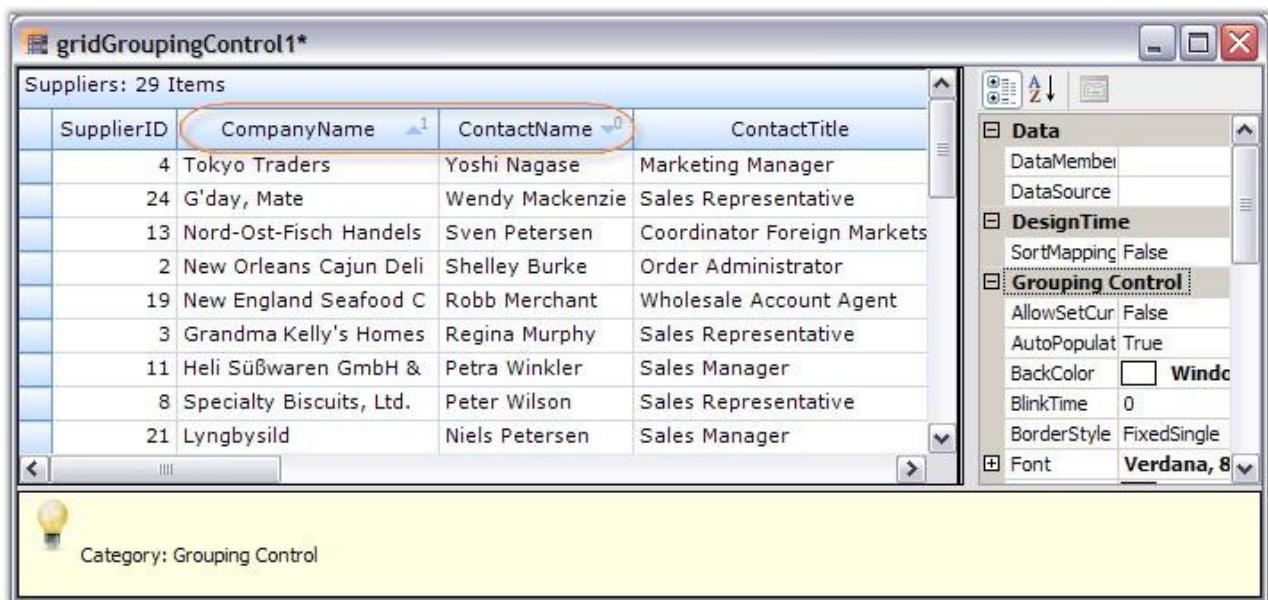


Figure 367: Sorted Grid highlighting the Sorted Columns

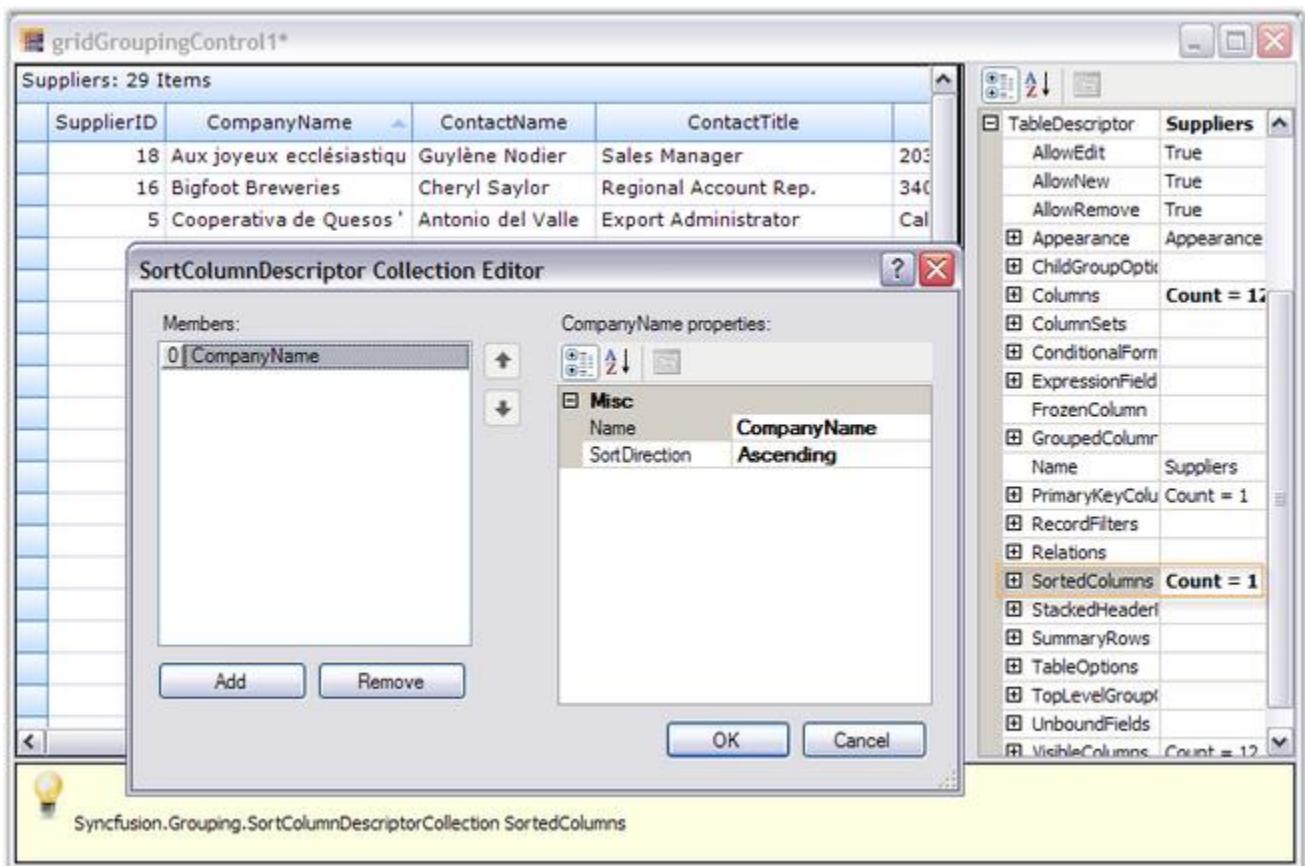


Figure 368: Sorted Grid showing the SortedColumns Editor

## Summaries

Summaries can be added in the designer itself by accessing the property, TableDescriptor.SummaryRows property. You can add as many summary rows as you need, each with a desired number of summary columns where you can specify the type of summary, summary format, the column based on whose values the summary has to be calculated and the like for each of the summary columns.

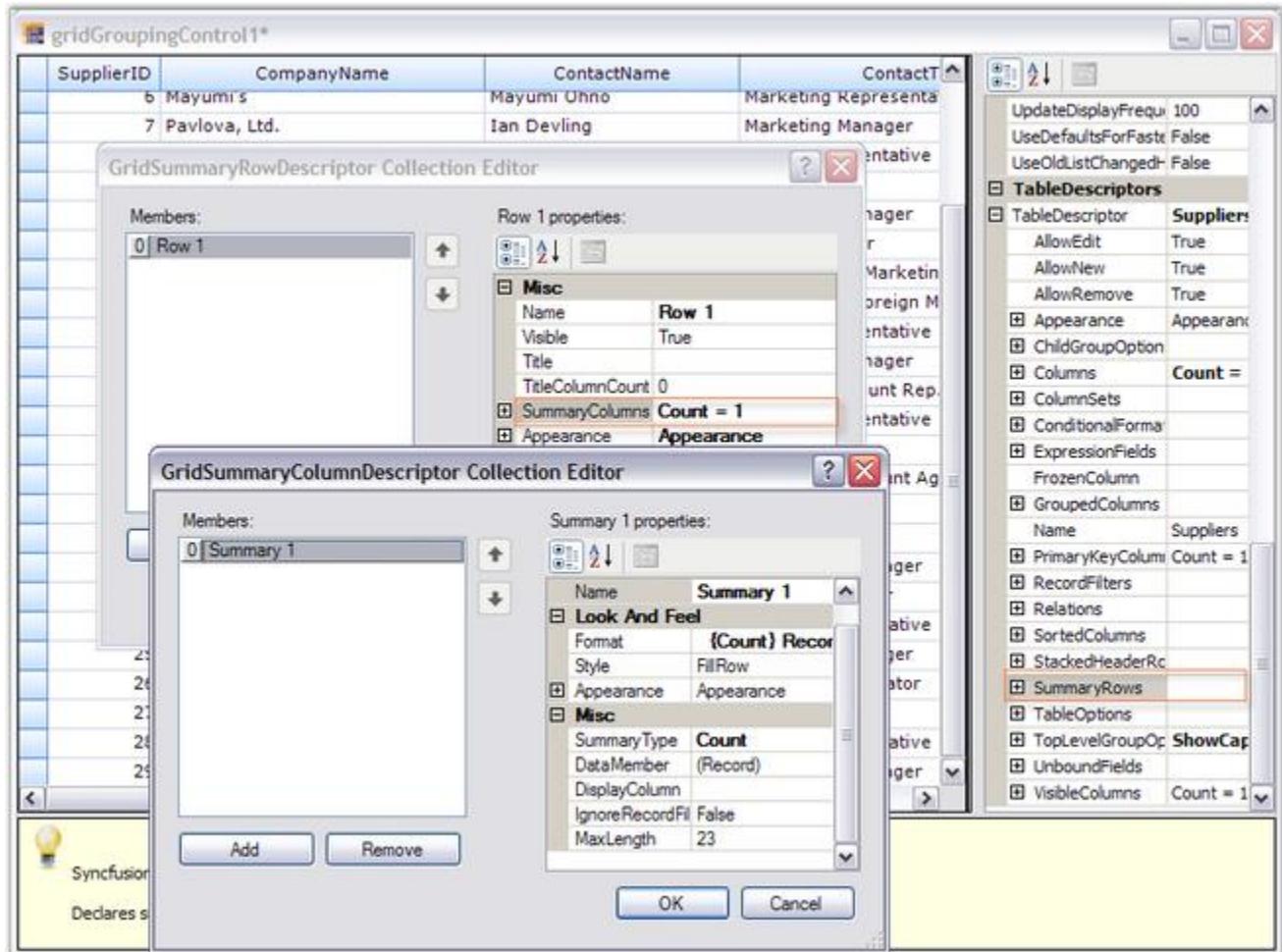


Figure 369: Adding Summary Rows

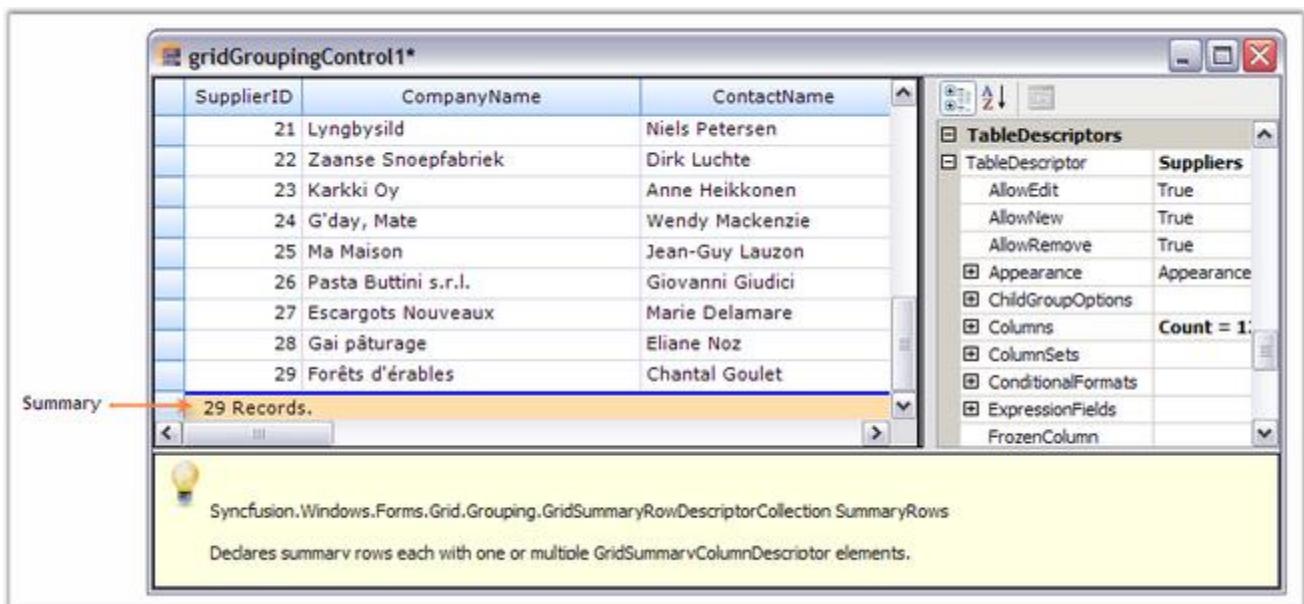


Figure 370: Grouping Grid displaying Summary

## Record Filters

By using the `TableDescriptor.RecordFilters` property, you can add row filters for your grid table. Once you have specified the filter criteria and the column name whose values have to be checked against the given criteria, the grouping grid will display only the subset of records that satisfy the given criteria.

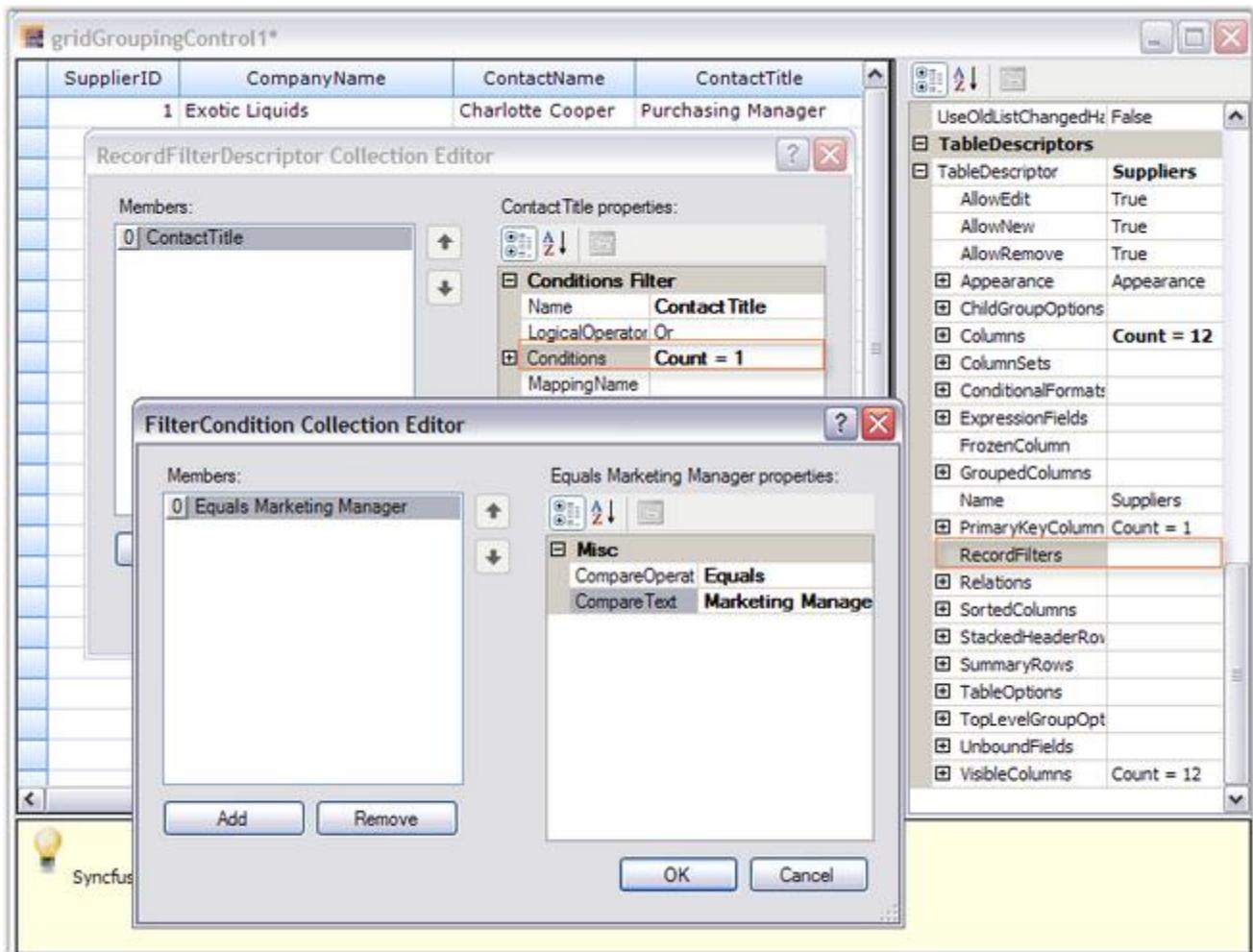


Figure 371: Adding row filters through RecordFilters Property

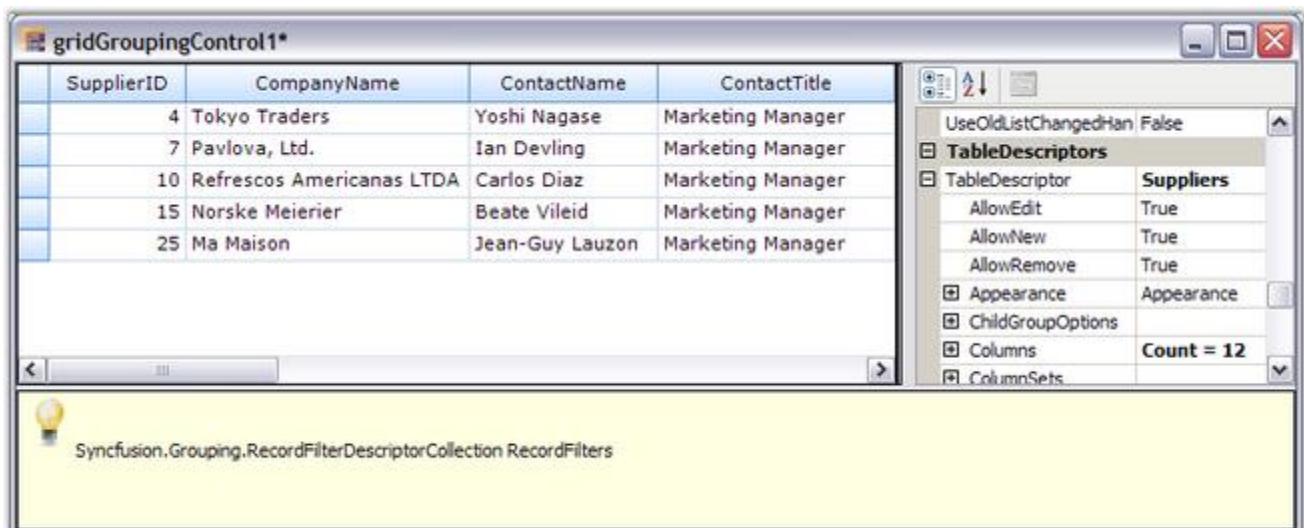


Figure 372: Filtered Grid

Grid Grouping control provides an **AutoFilterRow** which can be enabled by setting the **ShowFilterBar** property to true. Once it is done, you must enable **AllowFilter** property for the desired columns to enable filtering on those columns.

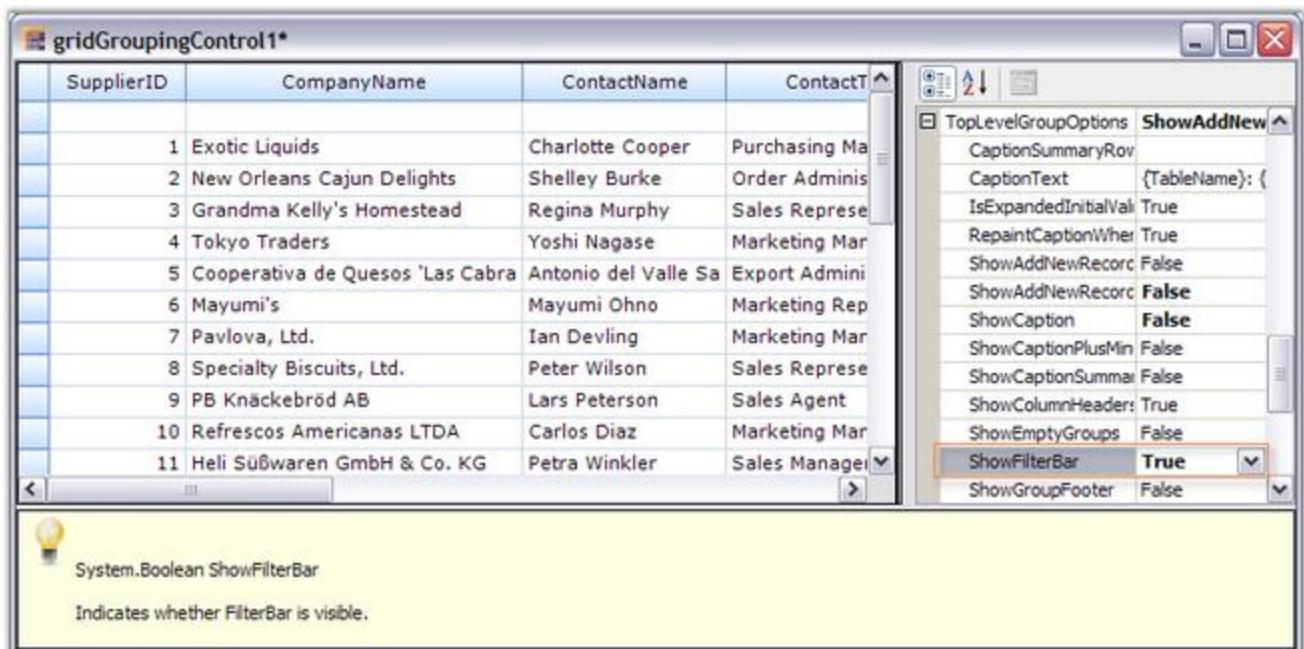


Figure 373: Grid with Auto-Filter Row Enabled

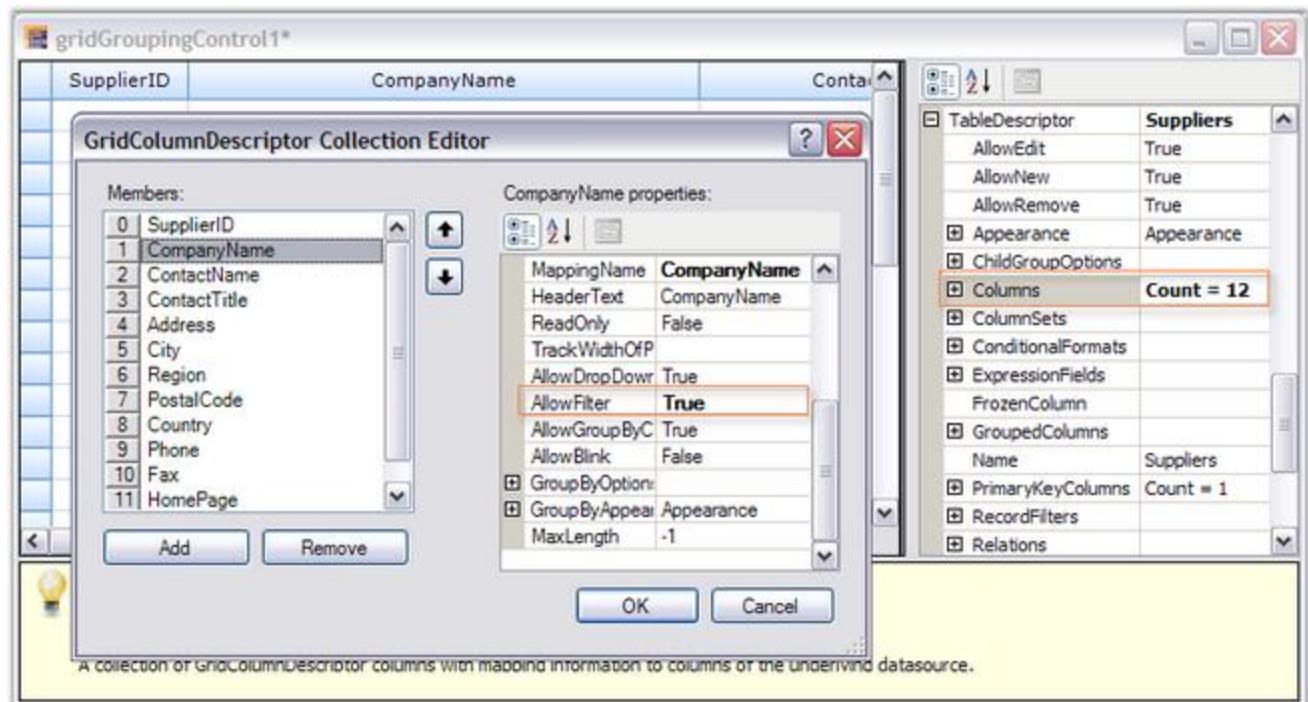


Figure 374: Setting AllowFilter property for 'CompanyName' Column

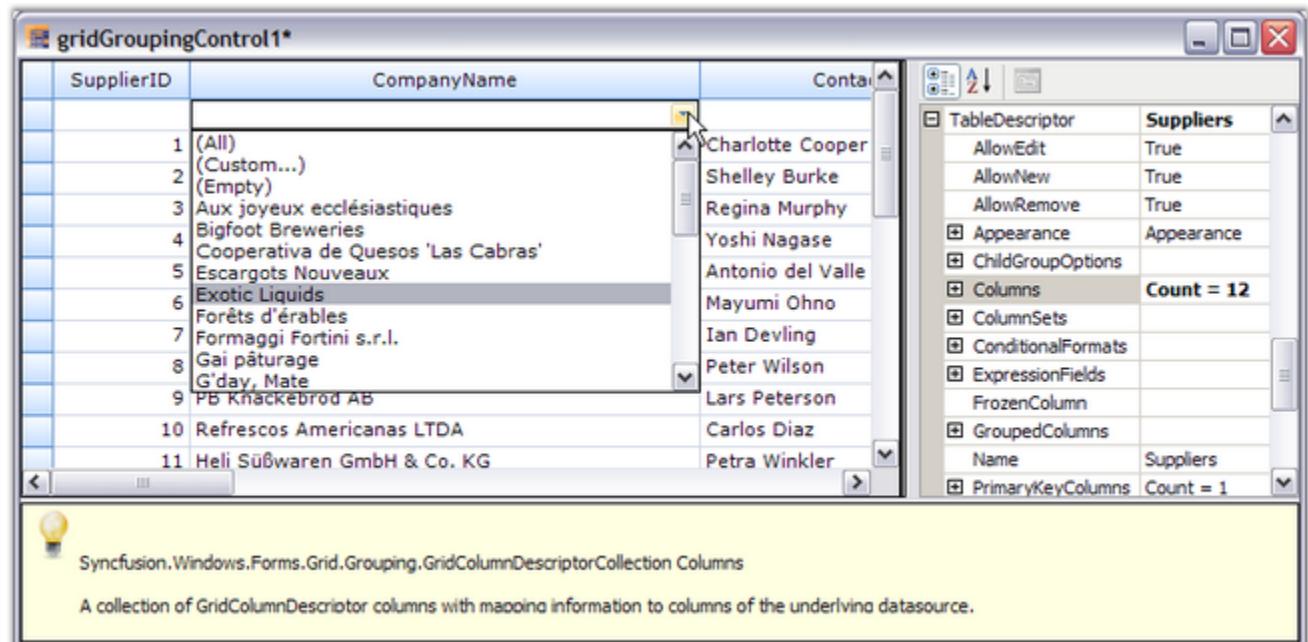


Figure 375: FilterBar drop down showing filtering options for the column 'CompanyName'

## Expression Fields

When there is a need to display calculated values based on the values on other fields in the same record, ExpressionFields would be the right choice to use. ExpressionFields can be created by using the TableDescriptor.ExpressionFields property. This will open an editor wherein you can add any number of expression fields each with its own expression used to calculate the results.

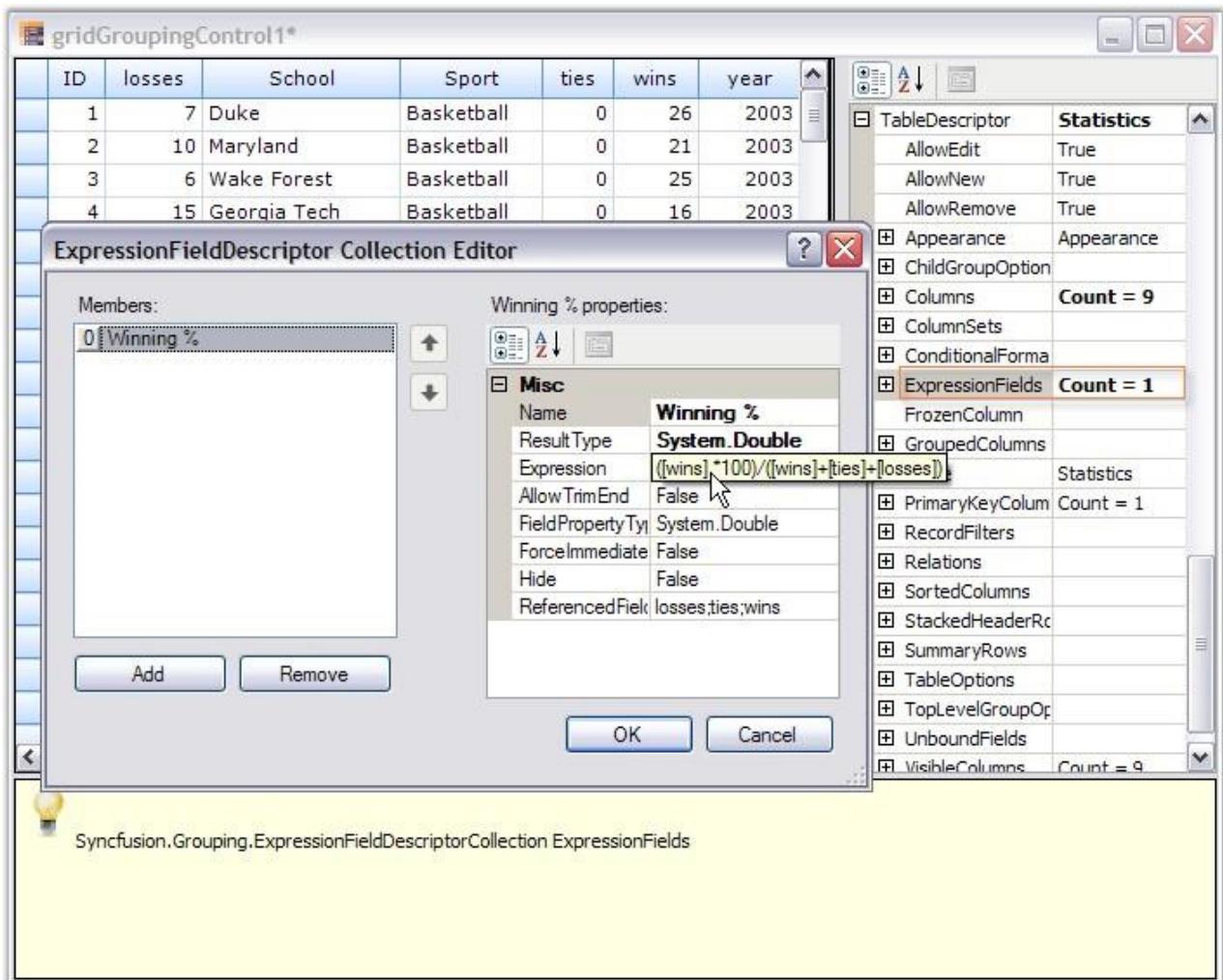


Figure 376: Adding an Expression Field

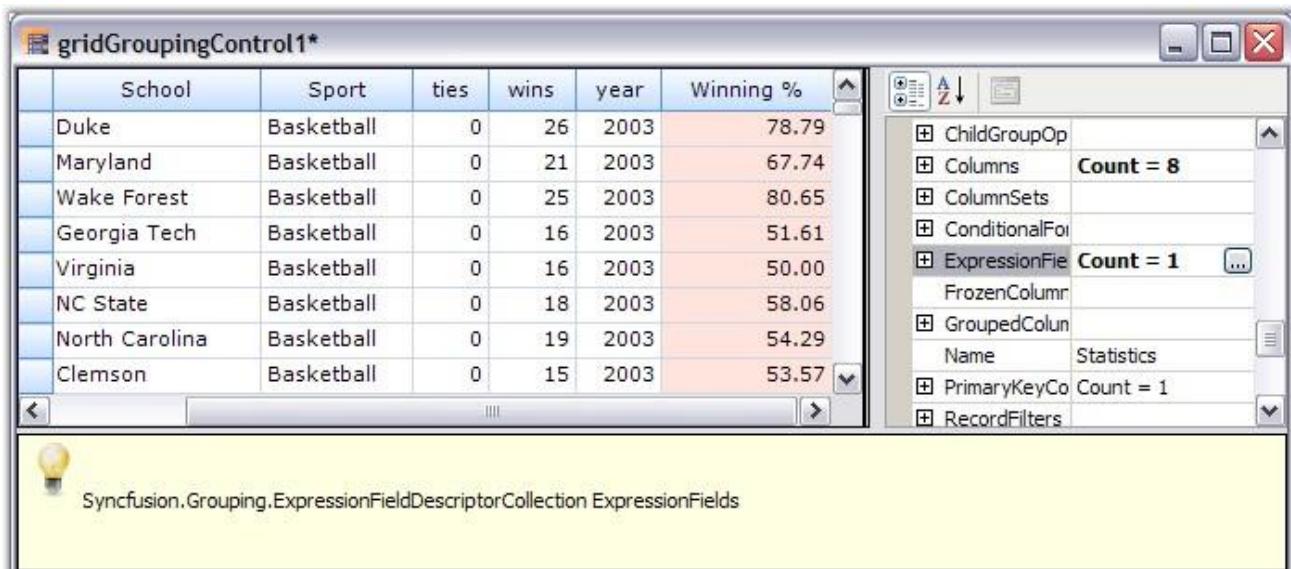


Figure 377: Grouping Grid showing the ExpressionField 'Winning%'

## Relations

It is possible to specify the relation to be used across the tables in case multiple tables are used. It can be done by accessing the `TableDescriptor.Relations` property wherein you can specify the relation type, name of the child table, relation keys consisting of the keys in parent and child tables and other information necessary to setup the relation.

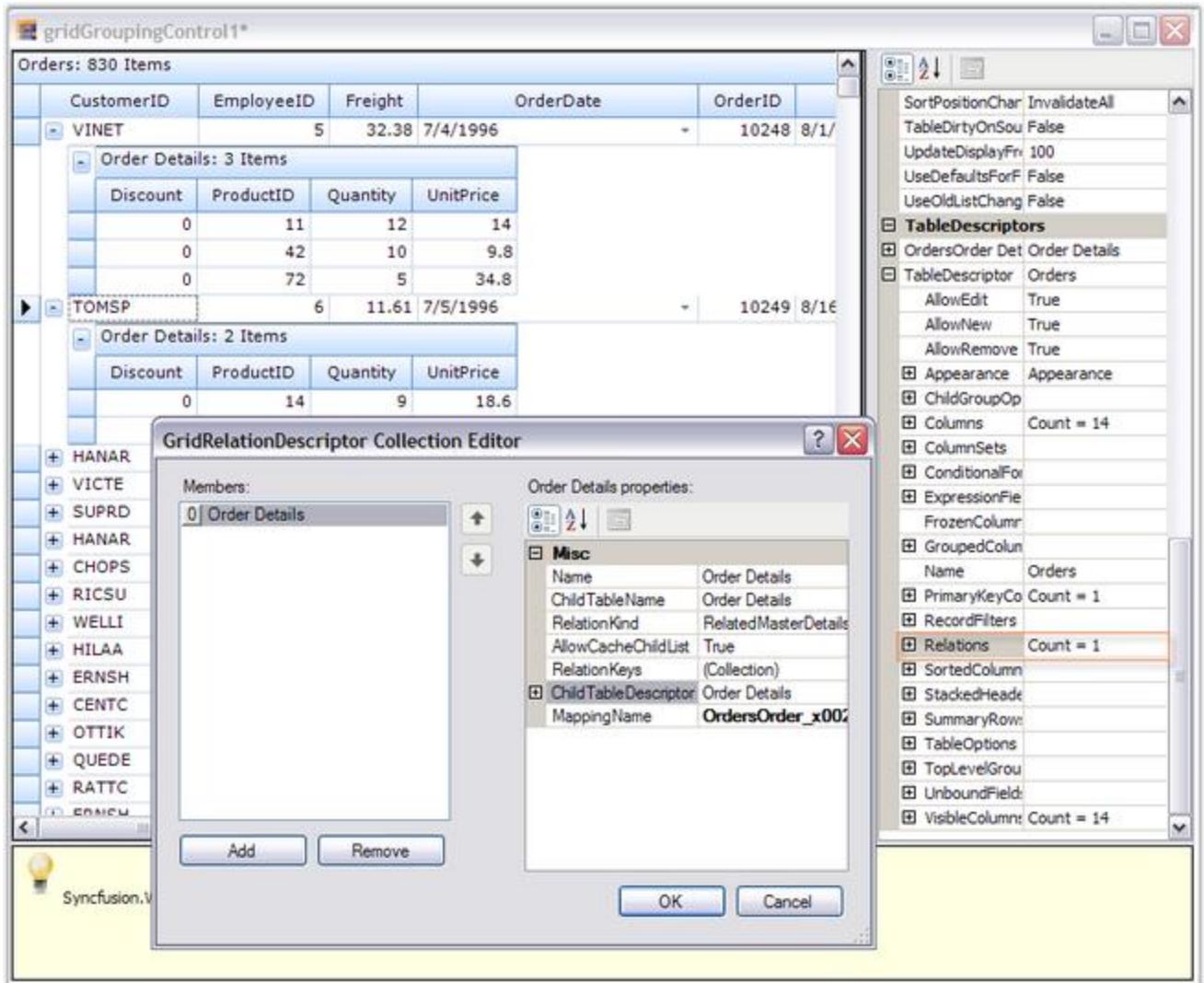


Figure 378: Hierarchical Grid with RelationKind 'RelatedMasterDetails'

## Appearance

The appearance of every grid element can be customized by accessing the Appearance property. It allows you to set GridStyleInfo properties like cell type, value, back color, font, etc. for grid cells. It holds a sub tree of different grid elements each with its own set of formatting properties. For instance, when you want to set appearance for alternate record field cell, you can make use of Appearance.AlternateRecordFieldCell property; if you want to customize summary cells, you will have to use Appearance.SummaryFieldCell or related property.

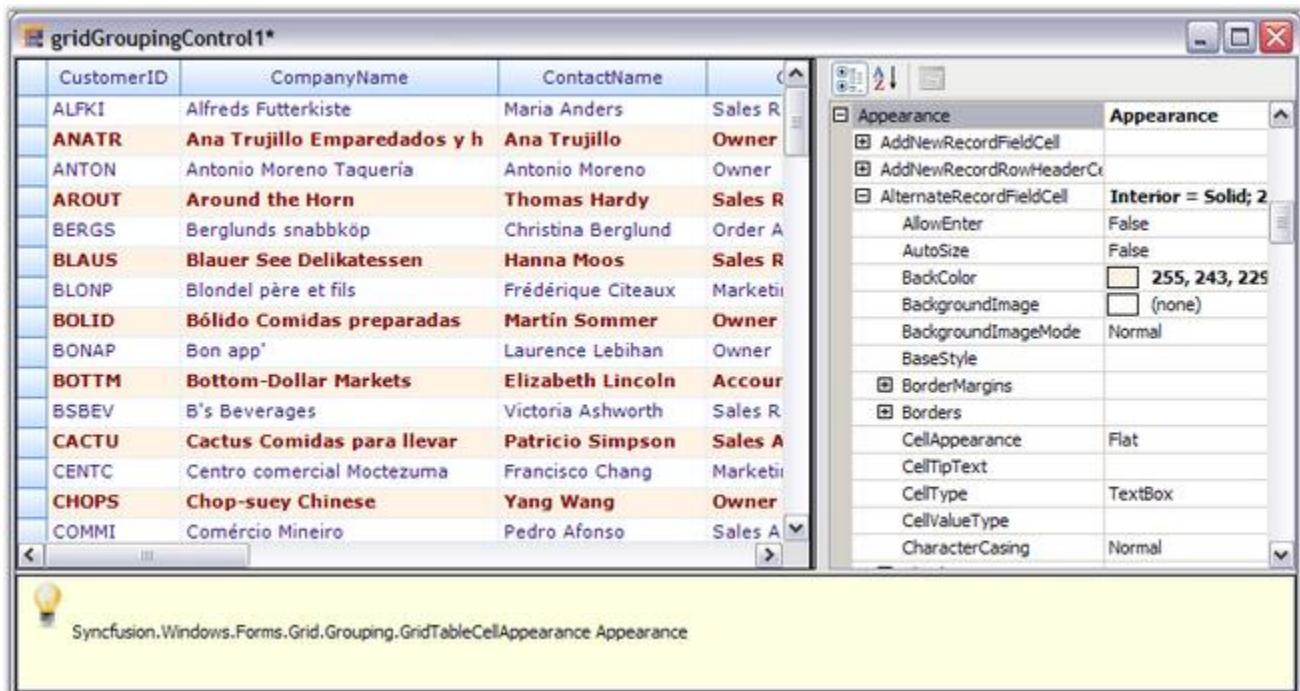


Figure 379: Grid Designer showing the appearance settings for AlternateRecordFieldCell

## Skins

You can change the appearance and behavior of every grid element to provide grid with a rich look and feel by setting skins. Grouping Grid currently offers five such skins: Office2007Blue, Office2007Silver, Office2007Black, Office2003 and SystemTheme(Default XP theme). To set a skin, the GridVisualStyles property which is under TableOptions section is used. It lists the possible skin options in a drop down, which will make the entire grid redrawn with the chosen style.

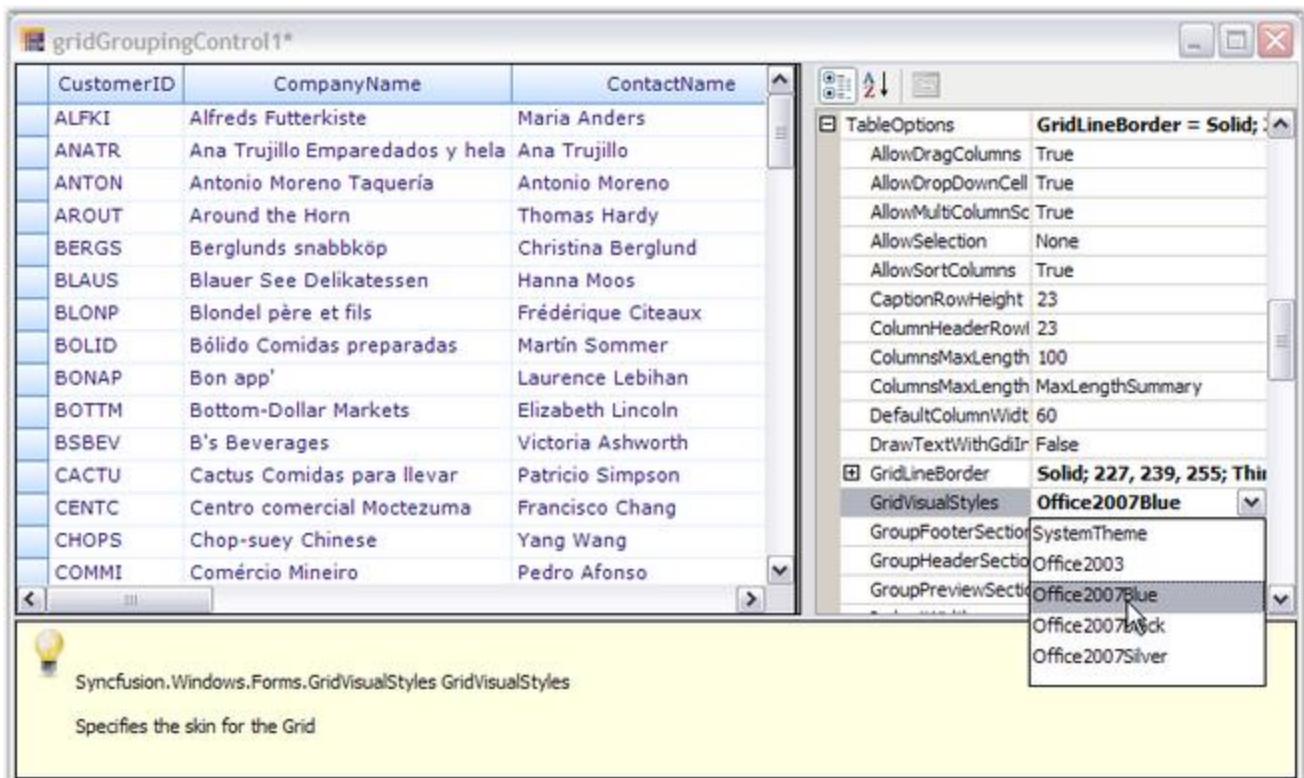


Figure 380: Grid Designer displaying the possible skins in a Drop Down

## See Also

[Grouping](#), [Sorting](#), [Summaries](#), [Record Filters](#), [Expression Fields](#), [Relations](#), [Appearance](#), [Grid Skins](#)

### 4.3.4.11 Navigation Bar

Grid Grouping control comes with an in-built Navigation Control that allows the user to browse through the records with ease. The navigation bar consists of buttons that facilitate navigation to first, next, previous, last records and also to the AddNew record in the grid. It also contains a label that displays the current record number together with the total record count.

NavigationBar can be enabled by setting **ShowNavigationBar** to true. It is possible to customize the default appearance of the navigation bar by setting the appropriate properties. Tooltips can be enabled for the navigation bar by setting the property, **ShowNavigationBarToolTips** to true. **ShowNavigationBar** must be set to true to enable tooltips.

Grid Grouping Control Property	Description
--------------------------------	-------------

ShowNavigationBar	Specifies whether to show the record navigation bar.
ShowNavigationBarToolTips	Specifies whether to show tooltips when the user hovers the mouse over the elements of the RecordNavigationBar.

The following code examples illustrate the above settings.

### 1. Using C#

[C#]

```
this.gridGroupingControl1.ShowNavigationBar = true;
this.gridGroupingControl1.ShowNavigationBarToolTips = true;
```

### 2. Using VB.NET

[VB .NET]

```
Private Me.gridGroupingControl1.ShowNavigationBar = True
Private Me.gridGroupingControl1.ShowNavigationBarToolTips = True
```

### Through Designer

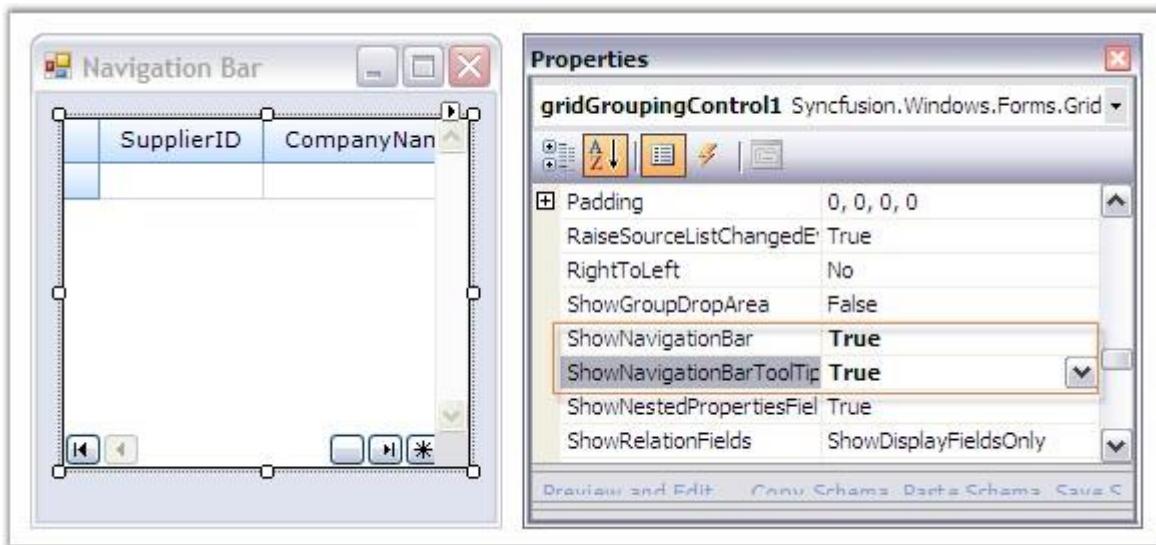


Figure 381: ShowNavigationBar = "True" and ShowNavigationBarToolTips = "True"

### Output



Figure 382: Navigation Bar and Navigation Bar ToolTip enabled for the Grid Grouping Control

#### 4.3.4.12 Print and Print Preview

Grid Grouping control supports printing and printing previews through the .NET Framework classes `System.Windows.Forms.PrintPreviewDialog` and `System.Windows.Forms.PrintDialog`. A derived **GridPrintDocument** which represents the print document is passed to these classes. This `GridPrintDocument` implements the printing logic that is needed to print multipage grids.

##### Code for Print Preview Dialog Box

[C#]

```
GridPrintDocument pd = new
GridPrintDocument(this.gridGroupingControl1.TableControl, true);
PrintPreviewDialog ppv = new PrintPreviewDialog();
ppv.Document = pd;
pd.DefaultPageSettings.Landscape = true;
ppv.ShowDialog();
```

[VB .NET]

```
Dim pd As New GridPrintDocument(Me.gridGroupingControl1.TableControl,
True)
Dim ppv As New PrintPreviewDialog()
ppv.Document = pd
```

```
pd.DefaultPageSettings.Landscape = True  
ppv.ShowDialog()
```

### Code for Print Dialog Box

[C#]

```
GridPrintDocument pd = new  
GridPrintDocument(this.gridGroupingControl1.TableControl);  
PrintDialog printDialog = new PrintDialog();  
printDialog.Document = pd;  
pd.DefaultPageSettings.Landscape = true;  
if (printDialog.ShowDialog() == DialogResult.OK)  
    pd.Print();
```

[VB .NET]

```
Dim pd As New GridPrintDocument(Me.gridGroupingControl1.TableControl)  
Dim printDialog As New PrintDialog()  
printDialog.Document = pd  
pd.DefaultPageSettings.Landscape = True  
If printDialog.ShowDialog() = Windows.Forms.DialogResult.OK Then  
    pd.Print()  
End If
```

Given below are sample screen shots.

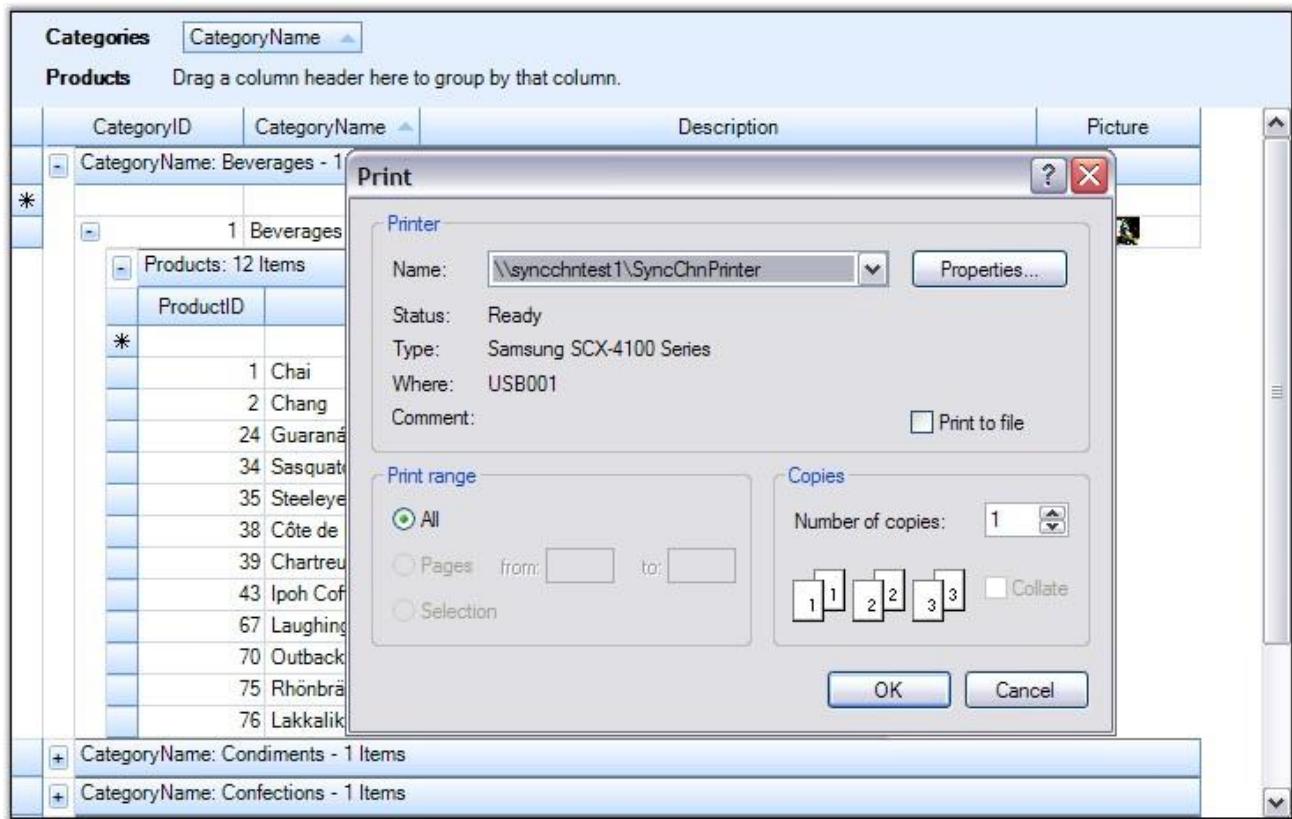


Figure 383: Grid with Print Dialog Box

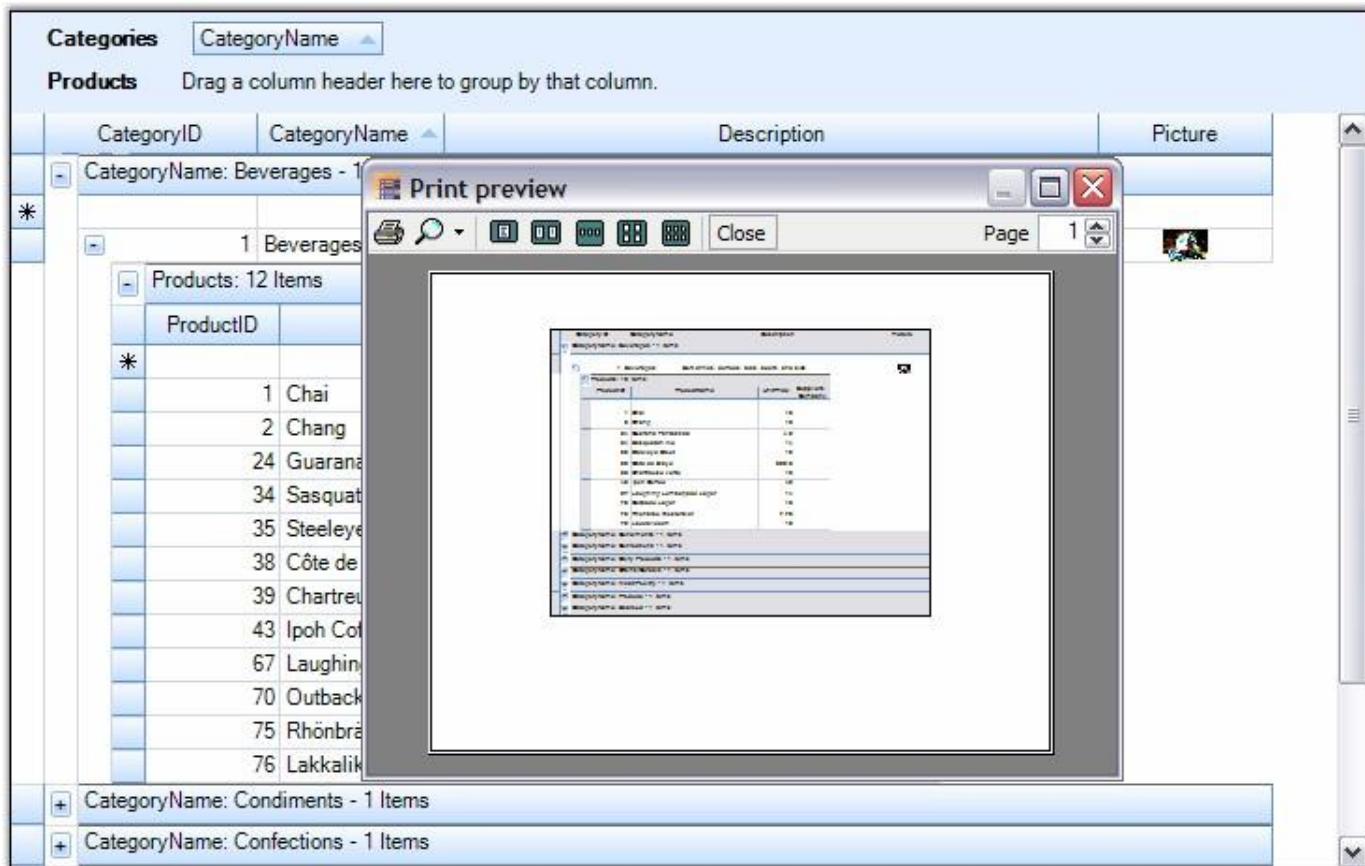


Figure 384: Grid with Print Preview Dialog Box



**Note:** For more details, refer the following browser sample:

<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Print\Print Grid Demo

### Advanced printing

Grid Grouping control supports printing of entire grid's column in a single page. Also, it allows the user to specify the header and footer for the page to be printed. This can be achieved by using the GridPrintDocumentAdv class. Column can be specified to fit in a single page by setting ScaleColumnsToFitPage property to true, header and footer can be added using the events DrawGridPrintHeader and DrawGridPrintFooter.

The following code example illustrates setting the header and footer for the page to be printed.

**[C#]**

```
Syncfusion.GridHelperClasses.GridPrintDocumentAdv pd = new
Syncfusion.GridHelperClasses.GridPrintDocumentAdv(this.gridGroupingCont
roll.TableControl);
pd.DefaultPageSettings.Margins = new
System.Drawing.Printing.Margins(25, 25, 25, 25);

// Set header and footer height.
pd.HeaderHeight = 70;
pd.FooterHeight = 50;

// Scale columns to fit page.
pd.ScaleColumnsToFitPage = true;

// Handle the following events to draw the header/footer.
pd.DrawGridPrintHeader += new
Syncfusion.GridHelperClasses.GridPrintDocumentAdv.DrawGridHeaderFooterE
ventHandler(pd_DrawGridPrintHeader);
pd.DrawGridPrintFooter += new
Syncfusion.GridHelperClasses.GridPrintDocumentAdv.DrawGridHeaderFooterE
ventHandler(pd_DrawGridPrintFooter); pd = new
GridPrintDocument(this.gridGroupingControll.TableControl);
PrintDialog printDialog = new PrintDialog();
printDialog.Document = pd;
pd.DefaultPageSettings.Landscape = true;
if (printDialog.ShowDialog() == DialogResult.OK)
pd.Print();
```

**[VB.NET]**

```
Dim pd As Syncfusion.GridHelperClasses.GridPrintDocumentAdv = New
Syncfusion.GridHelperClasses.GridPrintDocumentAdv(Me.gridGroupingContro
l1.TableControl)
pd.DefaultPageSettings.Margins = New
System.Drawing.Margins(25, 25, 25, 25)

' Set header and footer height.
pd.HeaderHeight = 70
pd.FooterHeight = 50

' Scale columns to fit page.
pd.ScaleColumnsToFitPage = True

' Handle the following events to draw the header/footer.
AddHandler pd.DrawGridPrintHeader, AddressOf pd_DrawGridPrintHeader
AddHandler pd.DrawGridPrintFooter, AddressOf pd_DrawGridPrintFooter
```

For more details, refer the following sample browser.

<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Print\Print Grid Demo

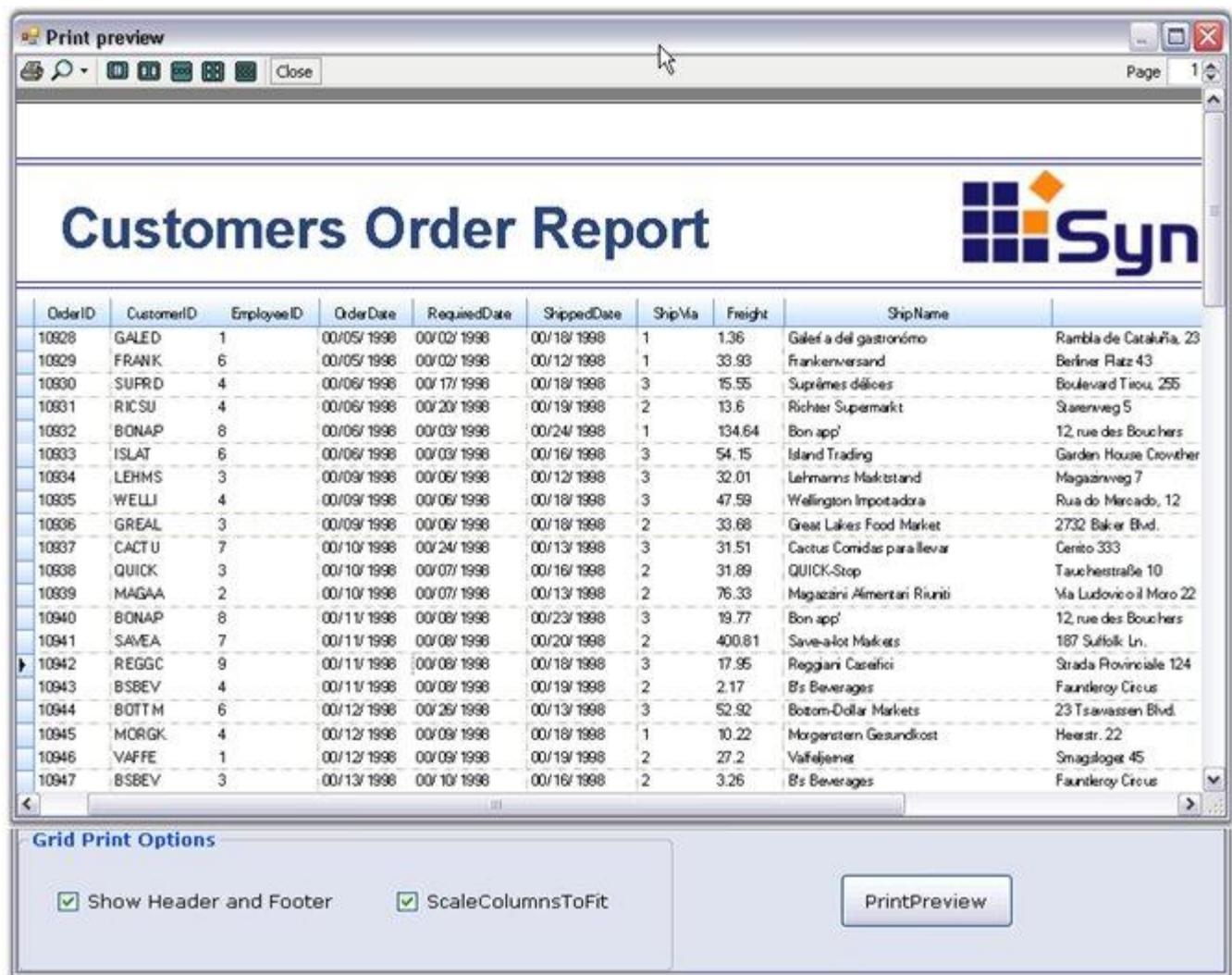


Figure 385: Header and Footer set for page to be Printed

#### 4.3.4.13 Advanced Features

This section discusses the following features:

#### 4.3.4.13.1 Custom Grouping

By default, the data rows having a matching value in a grouping column will be combined into a single group. If this does not suit your requirements, you can customize the grouping logic so that you can categorize the records as you need.

**Custom Grouping** can be achieved by adding a custom **Categorizer** object to **SortColumnDescriptor** that defines the group. You can also add a custom **Comparer** object to the **SortColumnDescriptor**. When a column is grouped, it is first sorted. The **Comparer** object allows you to control how the sorting is done on your column. Once the column is sorted, the custom **Categorizer** is used to determine which adjacent records in the sorted column belong to the same group. To create custom **Comparer** and **Categorizer** objects, you define classes that implement either **IComparer** (one method) or **ICategorizer** (two methods).

#### Example

Say you have a column with values for 0 to 50, and you want to have them grouped so that values less than 10 are in one group, values 10-20 are in another, values 20-30 in another, and so on. The following code example illustrates how to achieve this by using a **Custom Categorizer**.

1. Create a datasource and bind it to a grid grouping control.

[C#]

```
// Create the data source.
private DataTable GetDataTable()
{
    DataTable dt = new DataTable("MyTable");

    int nCols = 4;
    int nRows = 50;

    for(int i = 0; i < nCols; i++)
        dt.Columns.Add(new DataColumn(string.Format("Col{0}", i)));

    Random r = new Random();
    for(int i = 0; i < nRows; ++i)
    {
        DataRow dr = dt.NewRow();
```

```
        for(int j = 0; j < nCols; j++)
            dr[j] = r.Next(100).ToString();
            dt.Rows.Add(dr);
    }
    return dt;
}

// Set up a grouping grid.
this.gridGroupingControl1.DataSource = GetDataTable();
this.gridGroupingControl1.ShowGroupDropArea = true;
```

**[VB.NET]**

```
' Create the data source.
Private Function GetDataTable() As DataTable

    Dim dt As New DataTable("MyTable")

    Dim nCols As Integer = 4
    Dim nRows As Integer = 50

    Dim i As Integer
    For i = 0 To nCols - 1
        dt.Columns.Add(New DataColumn(String.Format("Col{0}", i)))
    Next i
    Dim r As New Random()

    i = 0
    While i < nRows
        Dim dr As DataRow = dt.NewRow()
        Dim j As Integer
        For j = 0 To nCols - 1
            dr(j) = r.Next(100).ToString()
        Next j
        dt.Rows.Add(dr)
        i += 1
    End While

    Return dt
End Function

' Set up a grouping grid.
Me.gridGroupingControl1.DataSource = GetDataTable()
Me.gridGroupingControl1.ShowGroupDropArea = True
```

2. Define a custom **Categorizer** by implementing the **IGroupByColumnCategorizer** interface. It writes a method named GetCategory where it defines a group and returns a category object which in turn is used by the interface members **GetGroupByCategoryKey** and **CompareCategoryKey** methods. The **GetGroupByCategoryKey** is used to return the key for the specified column and the record and the **CompareCategoryKey** determines if the current record belongs to the same category as the previous record.

[C#]

```
// Defines custom categorizer.
public class CustomCategorizer : Syncfusion.Grouping.IGroupByColumnCategorizer
{
    // Defines a group and returns a group category object (here
    returns 1 through 5).
    public static int GetCategory(int i)
    {
        int ret = 0;
        if(i < 10)
            ret = 1;
        else if(i >= 10 && i < 20)
            ret = 2;
        else if(i >= 20 && i < 30)
            ret = 3;
        else if(i >= 30 && i < 40)
            ret = 4;
        else
            ret = 5;

        return ret;
    }

    public object GetGroupByCategoryKey(SortColumnDescriptor column,
bool isForeignKey, Record record)
    {
        return GetCategory(int.Parse(
record.GetValue(column).ToString()));
    }

    public int CompareCategoryKey(SortColumnDescriptor column, bool
isForeignKey, object category, Record record)
    {
        return GetCategory(
int.Parse(record.GetValue(column).ToString())) - (int)category;
    }
}
```

[VB.NET]

```
' Defines custom categorizer.
Public Class CustomCategorizer
    Implements Syncfusion.Grouping.IGroupByColumnCategorizer

        ' Defines a group and returns a group category object (here returns
1 through 5).
        Public Shared Function GetCategory(ByVal i As Integer) As Integer
            Dim ret As Integer = 0
            If i < 10 Then
                ret = 1
            ElseIf i >= 10 AndAlso i < 20 Then
                ret = 2
            ElseIf i >= 20 AndAlso i < 30 Then
                ret = 3
            ElseIf i >= 30 AndAlso i < 40 Then
                ret = 4
            Else
                ret = 5
            End If
            Return ret
        End Function

        Public Function GetGroupByCategoryKey(ByVal column As
SortColumnDescriptor, ByVal isForeignKey As Boolean,
        ByVal record As Record) As Object Implements
IGroupByColumnCategorizer.GetGroupByCategoryKey
        Return
GetCategory(Integer.Parse(record.GetValue(column).ToString()))
    End Function

        Public Function CompareCategoryKey(ByVal column As
SortColumnDescriptor, ByVal isForeignKey As Boolean, ByVal category As
Object,      ByVal record As Record) As Integer Implements
IGroupByColumnCategorizer.CompareCategoryKey
        Return
GetCategory(Integer.Parse(record.GetValue(column).ToString())) -
Fix(category)
    End Function

End Class
```

3. Define a custom **Comparer** to ensure that the integer values are sorted as integers instead of strings.

[C#]

```
public class CustomComparer : IComparer
{
    public int Compare(object x, object y)
    {
        if(x == null)
            return -1;
        else if(y == null)
            return 100;
        else
        {
            int i = int.Parse(x.ToString());
            int j = int.Parse(y.ToString());
            return i - j;
        }
    }
}
```

**[VB.NET]**

```
Public Class CustomComparer Implements IComparer
    Public Function [Compare](ByVal x As Object, ByVal y As Object) As Integer Implements IComparer.Compare
        If x Is Nothing Then
            Return -1
        ElseIf y Is Nothing Then
            Return 100
        Else
            Dim i As Integer = Integer.Parse(x.ToString())
            Dim j As Integer = Integer.Parse(y.ToString())
            Return i - j
        End If
    End Function
End Class
```

4. Group the column **Col2** using a custom categorizer and comparer.

**[C#]**

```
// Group "Col2" by using a Custom Categorizer and Comparer.
SortColumnDescriptor cd = new SortColumnDescriptor("Col2");
cd.Categorizer = new CustomCategorizer();
cd.Comparer = new CustomComparer();
this.gridGroupingControl1.TableDescriptor.GroupedColumns.Add(cd);
```

**[VB .NET]**

```
' Group "Col2" by using a Custom Categorizer and Comparer.  
Dim cd As New Syncfusion.Grouping.SortColumnDescriptor("Col2")  
cd.Categorizer = New CustomCategorizer()  
cd.Comparer = New CustomComparer()  
Me.gridGroupingControl1.TableDescriptor.GroupedColumns.Add(cd)
```

5. Handle **QueryCellStyleInfo** event to display the custom group caption.

**[C#]**

```
// Subscribe to QueryCellStyleInfo event to display custom group  
// caption.  
this.gridGroupingControl1.QueryCellStyleInfo += new  
GridTableCellStyleInfoEventHandler(gridGroupingControl1_QueryCellStyleI  
nfo);  
  
private void gridGroupingControl1_QueryCellStyleInfo(object sender,  
GridTableCellStyleInfoEventArgs e)  
{  
    if (e.TableCellIdentity.GroupedColumn != null &&  
e.TableCellIdentity.DisplayElement.ParentGroup != null  
    && e.TableCellIdentity.DisplayElement.ParentGroup.Category is int)  
    {  
        if (e.TableCellIdentity.DisplayElement is CaptionRow  
        && e.TableCellIdentity.GroupedColumn.Name == "Col2")  
        {  
            int cat = (int)e.TableCellIdentity.DisplayElement.ParentGroup.Category;  
            string ret = "";  
            switch (cat)  
            {  
                case 1:  
                    ret = " < 10";  
                    break;  
                case 2:  
                    ret = "10 - 19";  
                    break;  
                case 3:  
                    ret = "20 - 29";  
                    break;  
                case 4:  
                    ret = "30 - 39";  
                    break;  
                case 5:  
            }  
        }  
    }  
}
```

```
        ret = " >= 40";
        break;
    }
    e.Style.CellValue = String.Format("{0}: {1} Items.", ret,
e.TableCellIdentity.DisplayElement.ParentGroup.GetChildCount());
}
}
```

**[VB.NET]**

```
' Subscribe to QueryCellStyleInfo event to display custom group
caption.
AddHandler Me.gridGroupingControl1.QueryCellStyleInfo, AddressOf
gridGroupingControl1_QueryCellStyleInfo

Private Sub gridGroupingControl1_QueryCellStyleInfo(ByVal sender As
Object, ByVal e As GridTableViewCellEventArgs)
    If Not (e.TableCellIdentity.GroupedColumn Is Nothing) AndAlso Not
(e.TableCellIdentity.DisplayElement.ParentGroup Is Nothing) AndAlso
TypeOf e.TableCellIdentity.DisplayElement.ParentGroup.Category Is
Integer Then

        If TypeOf e.TableCellIdentity.DisplayElement Is CaptionRow
        AndAlso e.TableCellIdentity.GroupedColumn.Name = "Col2" Then
            Dim cat As Integer =
Fix(e.TableCellIdentity.DisplayElement.ParentGroup.Category)
            Dim ret As String = ""
            Select Case cat
                Case 1
                    ret = " < 10"
                Case 2
                    ret = "10 - 19"
                Case 3
                    ret = "20 - 29"
                Case 4
                    ret = "30 - 39"
                Case 5
                    ret = " >= 40"
            End Select
            e.Style.CellValue = String.Format("{0}: {1} Items.", ret,
e.TableCellIdentity.DisplayElement.ParentGroup.GetChildCount()
)
        End If
    End If
End Sub
```

Given below is a sample screen shot.

	Col0	Col1	Col2	Col3
[+]	< 10: 4 Items.			
	67	72	0	74
	48	89	2	95
	67	85	4	82
	20	88	6	17
[+]	10 - 19: 4 Items.			
	24	36	11	83
	86	90	11	42
	28	74	16	91
	2	46	17	47
[+]	20 - 29: 6 Items.			
[+]	30 - 39: 5 Items.			
[+]	>= 40: 31 Items.			

Figure 386: Custom Grouping Illustrated

#### 4.3.4.13.2 Custom Sorting

Custom Sorting allows you to implement custom sorting logic when the standard sorting techniques do not meet your needs. To support custom sorting in Grid Grouping control, the user needs to add an **IComparer** object and handle one event.

The **Comparer** object allows you to control how the sorting is done on the column. This is the place where you can define your own sorting logic. After customizing the sorting logic through **IComparer**, you can make the grouping grid use this special **IComparer** by handling an event.

#### Example

Consider a scenario where one of the data columns of your datasource consists of **Date-Numeric** combination values. In this case, the default sorting will not produce the results we desire. The sorting has to be done first on date values and then on numeric values. Hence we need to write a custom sorting logic by defining a special **IComparer** object. The grouping grid could then be made to use this custom comparer by handling an event that gets fired while sorting the columns.

The example illustrates this process in a step by step manner.

1. Setup a datasource and bind it to a grid grouping control.

**[C#]**

```
// Create a Data Source.  
DataTable dt = new DataTable("MyTable");  
int nCols = 4;  
int nRows = 30;  
for (int i = 0; i < nCols; i++)  
    dt.Columns.Add(new DataColumn(string.Format("Col{0}", i)));  
Random r = new Random();  
for (int i = 0; i < nRows; ++i)  
{  
    DataRow dr = dt.NewRow();  
    for (int j = 0; j < nCols; j++)  
        dr[j] = string.Format("row{0} col{1}", i, j);  
    DateTime d = DateTime.Now.AddDays(r.Next(i < 20 ? 700 : 1));  
    dr[nCols - 1] = string.Format("{0:MMM}{1:00} - {2:0.00}", d,  
d.Year, r.Next(1000000) / 100d);  
    dt.Rows.Add(dr);  
}  
  
// Bind the data source to the grouping grid.  
this.gridGroupingControl1.DataSource = new DataView(dt);
```

**[VB.NET]**

```
' Create a Data Source.  
Dim dt As DataTable = New DataTable("MyTable")  
  
Dim nCols As Integer = 4  
Dim nRows As Integer = 30  
  
Dim i As Integer = 0  
Do While i < nCols  
    dt.Columns.Add(New DataColumn(String.Format("Col{0}", i)))  
    i += 1  
Loop  
  
Dim r As Random = New Random()  
i = 0  
Do While i < nRows  
    Dim dr As DataRow = dt.NewRow()
```

```
Dim j As Integer = 0
Do While j < nCols
    dr(j) = String.Format("row{0} col{1}", i, j)
    j += 1
Loop
Dim d As DateTime = DateTime.Now.AddDays(r.Next(IIf(i < 20, 700,
1)))
dr(nCols - 1) = String.Format("{0:MMM}{1:00} - {2:0.00}", d, d.Year,
r.Next(1000000) / 100.0R)
dt.Rows.Add(dr)
i += 1
Loop

' Bind the data source to the grouping grid.
Me.gridGroupingControll.DataSource = New DataView(dt)
```

2. Define a Custom Comparer by implementing the **IComparer** interface.

[C#]

```
// Sorts first on date and then on value.
public class DateComparer : IComparer
{
    // IComparer Members.
    public int Compare(object x, object y)
    {
        if (x == null && y == null)
            return 0;
        else if (x == null)
            return -1;
        else if (y == null)
            return 1;
        else
        {
            DateTime xdate = Convert.ToDateTime(x.ToString());
            DateTime ydate = Convert.ToDateTime(y.ToString());
            int c = xdate.CompareTo(ydate);
            return c;
        }
    }
}
```

[VB.NET]

```
' Sorts first on date and then on value.
```

```
Public Class DateComparer : Implements IComparer

    ' IComparer Members.

    Public Function Compare(ByVal x As Object, ByVal y As Object) As Integer Implements IComparer.Compare
        If x Is Nothing AndAlso y Is Nothing Then
            Return 0
        Else If x Is Nothing Then
            Return -1
        Else If y Is Nothing Then
            Return 1
        Else
            Dim c As Integer =
                GetDate(x.ToString()).CompareTo(GetDate(y.ToString()))
            If c = 0 Then
                c = GetDouble(x.ToString()).CompareTo(GetDouble(y.ToString()))
            End If
            Return c
        End If
    End Function

    Private Function GetDate(ByVal s As String) As DateTime
        Dim dt As DateTime = DateTime.MinValue
        Dim pos As Integer = s.IndexOf("-"c)
        If pos > -1 Then
            DateTime.TryParse(s.Substring(0, pos), dt)
        End If
        Return dt
    End Function

    Private Function GetDouble(ByVal s As String) As Double
        Dim d As Double = Double.NaN
        Dim pos As Integer = s.IndexOf("-"c)
        If pos > -1 Then
            Double.TryParse(s.Substring(pos + 1), d)
        End If
        Return d
    End Function
End Class
```

3. Handle the SortColumnsChanging event to make the grid use the custom comparer.

[C#]

```
private string specialDateColName = "Col3";
private DateComparer specialDateComparer = new DateComparer();
```

```
// Set up support for custom sort on Grid Grouping control.  
this.gridGroupingControl1.TableDescriptor.SortedColumns.Changing += new  
ListPropertyChangedEventHandler(SortedColumns_Changing);  
  
// Make the Grid Grouping control use the special IComparer.  
void SortedColumns_Changing(object sender, ListPropertyChangedEventArgs  
e)  
{  
    SortColumnDescriptor scd = e.Item as SortColumnDescriptor;  
    if (e.Action == ListPropertyChangedType.Add && scd != null &&  
scd.Name == specialDateColName)  
    {  
        ((SortColumnDescriptor)e.Item).Comparer = specialDateComparer;  
    }  
}
```

**[VB.NET]**

```
Private specialDateColName As String = "Col3"  
Private specialDateComparer As DateComparer = New DateComparer()  
  
' Set up support for custom sort on Grid Grouping control.  
AddHandler gridGroupingControl1.TableDescriptor.SortedColumns.Changing,  
AddressOf SortedColumns_Changing  
  
' Make the Grid Grouping control use the special IComparer.  
Private Sub SortedColumns_Changing(ByVal sender As Object, ByVal e As  
ListPropertyChangedEventArgs)  
    Dim scd As SortColumnDescriptor = CType(IIf(TypeOf e.Item Is  
SortColumnDescriptor, e.Item, Nothing), SortColumnDescriptor)  
    If e.Action = ListPropertyChangedType.Add AndAlso Not scd Is Nothing  
AndAlso scd.Name = specialDateColName Then  
        CType(e.Item, SortColumnDescriptor).Comparer =  
        specialDateComparer  
    End If  
End Sub
```

4. When you run the sample, click on Col3 to sort it. You will see the effect of the custom sorting logic. The screen shots given below shows the sorted grid with/without Custom Comparer.

	Col0	Col1	Col2	Col3
	row4 col0	row4 col1	row4 col2	Apr2008 - 8.82
	row9 col0	row9 col1	row9 col2	Aug2007 - 3274.23
	row10 col0	row10 col1	row10 col2	Aug2007 - 6214.15
	row12 col0	row12 col1	row12 col2	Aug2007 - 8595.03
	row11 col0	row11 col1	row11 col2	Dec2007 - 5467.78
	row2 col0	row2 col1	row2 col2	Dec2007 - 9296.92
	row13 col0	row13 col1	row13 col2	Dec2008 - 7074.72
	row5 col0	row5 col1	row5 col2	Jul2007 - 2455.41
	row14 col0	row14 col1	row14 col2	Jul2007 - 4249.49
	row7 col0	row7 col1	row7 col2	Jul2007 - 6591.97
	row19 col0	row19 col1	row19 col2	Jun2007 - 764.79
	row6 col0	row6 col1	row6 col2	Jun2008 - 1988.47
	row15 col0	row15 col1	row15 col2	Mar2008 - 2800.65
	row0 col0	row0 col1	row0 col2	Mar2008 - 9333.82
	row21 col0	row21 col1	row21 col2	May2007 - 1368.73

Figure 387: Sorted Grid without Custom Comparer

	Col0	Col1	Col2	Col3
	row29 col0	row29 col1	row29 col2	May2007 - 994.58
	row28 col0	row28 col1	row28 col2	May2007 - 1606.26
	row20 col0	row20 col1	row20 col2	May2007 - 2803.29
	row24 col0	row24 col1	row24 col2	May2007 - 3668.01
	row21 col0	row21 col1	row21 col2	May2007 - 4512.78
	row23 col0	row23 col1	row23 col2	May2007 - 5093.65
	row15 col0	row15 col1	row15 col2	May2007 - 6858.04
	row27 col0	row27 col1	row27 col2	May2007 - 8767.58
	row26 col0	row26 col1	row26 col2	May2007 - 9541.53
	row25 col0	row25 col1	row25 col2	May2007 - 9825.07
	row22 col0	row22 col1	row22 col2	May2007 - 9941.11
	row5 col0	row5 col1	row5 col2	Jun2007 - 3215.14
	row10 col0	row10 col1	row10 col2	Jun2007 - 7670.99
	row3 col0	row3 col1	row3 col2	Aug2007 - 3385.26
	row17 col0	row17 col1	row17 col2	Sep2007 - 1313.98

Figure 388: Sorted Grid with Custom Comparer

#### 4.3.4.13.3 Custom Summary

This section deals with the implementation of **Custom Summaries**. You can go for custom summaries when the standard summaries do not meet your needs.

Custom Summaries can be created by subclassing the **SummaryBase** class and by specifying the **SummaryType** as **Custom**. In the derived class, you can customize the **CreateSummaryMethod** in the way you want. It is the right place where you can define the way how summaries need to be calculated. You can then make this method to be called each time the summary is queried, by assigning the Custom **CreateSummaryDelegate** to the **CreateSummaryMethod** property of the **SummaryDescriptor** inside the **QueryCustomSummary** event handler. Optionally, you can also specify **CreateSummaryFromElementMethod** if summaries need to be calculated also for elements other than records.

### Example

Here is an example that calculates the summaries for both Minimum and Maximum values of the Date field. This can be achieved by using a custom summary field. For Custom summary, the summary Base class is derived and the combine method is made use of to get the new summary value by comparing it with other summary values.

The following code example illustrates how the Combine method compares and gets the maximum date value.

#### [C#]

```
public override SummaryBase Combine(SummaryBase other)
{
    return Combine((DateMaxSummary) other);
}

public DateMaxSummary Combine(DateMaxSummary other)
{
    if (Max > other.Max)
        return new DateMaxSummary(this.Max);
    else
        return new DateMaxSummary(other.Max);
}
```

#### [VB .NET]

```
Public Overloads Overrides Function Combine(ByVal other As SummaryBase)
As SummaryBase
    Return Combine(CType(other, DateMaxSummary))
End Function

Public Overloads Function Combine(ByVal other As DateMaxSummary) As
```

```
DateMaxSummary  
If (Max > other.Max) Then  
    Return New DateMaxSummary(Me.Max)  
Else  
    Return New DateMaxSummary(other.Max)  
End If  
End Function
```

The following are the classes and events that are used to create the date summary field.

1. Creating a GridSummaryColumnDescriptor instance.

**[C#]**

```
GridSummaryColumnDescriptor sd1 = new GridSummaryColumnDescriptor();  
sd1.Name = "MaxDate";  
sd1.DataMember = "Date";  
sd1.DisplayColumn = "Date";  
sd1.Format = "{Max}";  
sd1.SummaryType = SummaryType.Custom;  
this.gridGroupingControl1.TableDescriptor.SummaryRows.Add(new  
GridSummaryRowDescriptor("Row 1", "Max", sd1));
```

**[VB.NET]**

```
Dim sd1 As GridSummaryColumnDescriptor = New  
GridSummaryColumnDescriptor  
sd1.Name = "MaxDate"  
sd1.DataMember = "Date"  
sd1.DisplayColumn = "Date"  
sd1.Format = "{Max}"  
sd1.SummaryType = SummaryType.Custom  
Me.gridGroupingControl1.TableDescriptor.SummaryRows.Add(New  
GridSummaryRowDescriptor("Row 1", "Max", sd1))
```

2. Using the QueryCustomSummary event to instantiate the custom summary for maximum and minimum date.

**[C#]**

```
private void gridGroupingControl1_QueryCustomSummary(object sender,
GridQueryCustomSummaryEventArgs e)
{
    switch (e.SummaryColumn.Name)
    {
        case "MaxDate":
        {
            e.SummaryDescriptor.CreateSummaryMethod = new
CreateSummaryDelegate(DateMaxSummary.CreateSummaryMethod);
            break;
        }
        case "MinDate":
        {
            e.SummaryDescriptor.CreateSummaryMethod = new
CreateSummaryDelegate(DateMinSummary.CreateSummaryMethod);
            break;
        }
    }
}
```

**[VB .NET]**

```
Private Sub gridGroupingControl1_QueryCustomSummary(ByVal sender As
Object, ByVal e As GridQueryCustomSummaryEventArgs)
Select Case (e.SummaryColumn.Name)
Case "MaxDate"
    e.SummaryDescriptor.CreateSummaryMethod = New
CreateSummaryDelegate(AddressOf
DateMaxSummary.CreateSummaryMethod)
Exit Select
Case "MinDate"
    e.SummaryDescriptor.CreateSummaryMethod = New
CreateSummaryDelegate(AddressOf
DateMinSummary.CreateSummaryMethod)
Exit Select
End Select
End Sub
```

3. The CurrentRecordContextChange event is used to refresh the grid so that new values will be updated in the summary row.

**[C#]**

```
private void gridGroupingControl1_CurrentRecordContextChange(object
sender, CurrentRecordContextChangeEvent e)
```

```
{  
    if (e.Action == CurrentRecordAction.EndEditComplete)  
    {  
        e.Record.InvalidateCounterBottomUp();  
        this.gridGroupingControl1.Refresh();  
    }  
}
```

**[VB.NET]**

```
Private Sub gridGroupingControl1_CurrentRecordContextChange(ByVal  
sender As Object, ByVal e As CurrentRecordContextChangeEventArgs)  
    If (e.Action = CurrentRecordAction.EndEditComplete) Then  
        e.Record.InvalidateCounterBottomUp()  
        Me.gridGroupingControl1.Refresh()  
    End If  
End Sub
```

#### 4.3.4.13.4 Custom Filter Dialog

When you have a grouping grid with filter bar enabled, the drop down in a filter bar cell shows a 'Custom' option too along with other filtering options. This option allows you to customize the filtering mechanism. By default, the custom option pops up the RecordFieldDescriptor collection editor for the table. This default behavior must be overridden when a custom defined filter dialog has to be displayed as a result of clicking the custom option. It can be achieved by replacing the default FilterBarCell with custom defined filter bar cell.

This section best demonstrates merging of user defined filter dialog with the grid filter bar and also explores the code to implement it.

#### Implementation

To adopt our own filter dialog with respect to the 'Custom' filtering option, we must define a **Custom Filter CellType** which should then replace the filter bar cell that is shown by default.

In general, a celltype requires a model class and a renderer class. The model class handles the serialization requirements for the control and creates the renderer class. The renderer class handles the UI requirements of the cell, such as drawing it, handling mouse actions, and so on.

For our FilterBarCell, we have to create the model and renderer classes by inheriting the **GridTableFilterBarCellModel** and **GridTableFilterBarCellRenderer** classes. In the renderer, you should override **ListBoxClick** or **ListBoxMouseUp** event handlers (the event that gets fired upon clicking a filter option) in order to show up our own filter dialog. Once the dialog is closed, retrieve the filter string that has been setup using the dialog and add this filter string into the **RecordFilters** collection of the **TableDescriptor**. This would set up a filter using the custom option.

Following steps illustrate how to add a custom filter dialog.

1. Set up a grouping grid and bind a data source into it. Enable the grid filter bar to show the filter options for all the columns.

**[C#]**

```
// Set up a data source.
DataTable dt = new DataTable("MyTable");

int nCols = 6;
int nRows = 20;
Random r = new Random(100);
for (int i = 0; i < nCols; i++)
    dt.Columns.Add(new DataColumn(string.Format("Col{0}", i),
typeof(int)));

for (int i = 0; i < nRows; ++i)
{
    DataRow dr = dt.NewRow();
    for (int j = 0; j < nCols; j++)
        dr[j] = r.Next(10);
    dt.Rows.Add(dr);
}

// Bind it to the grouping grid.
this.gridGroupingControl1.DataSource = dt;

// Enable the filter bar.
this.gridGroupingControl1.TopLevelGroupOptions.ShowFilterBar = true;

for (int i = 0; i < gridGroupingControl1.TableDescriptor.Columns.Count;
i++)
gridGroupingControl1.TableDescriptor.Columns[i].AllowFilter = true;
```

**[VB .NET]**

```
' Setup a data source.
Dim dt As DataTable = New DataTable("MyTable")
```

```
Dim nCols As Integer = 6
Dim nRows As Integer = 20
Dim r As Random = New Random(100)

Dim i As Integer = 0
Do While i < nCols
    dt.Columns.Add(New DataColumn(String.Format("Col{0}", i),
        GetType(Integer)))
    i += 1
Loop
i = 0

Do While i < nRows
    Dim dr As DataRow = dt.NewRow()
    Dim j As Integer = 0
    Do While j < nCols
        dr(j) = r.Next(10)
        j += 1
    Loop
    dt.Rows.Add(dr)
    i += 1
Loop

' Bind it to the grouping grid.
Me.gridGroupingControll1.DataSource = dt

' Enable the filter bar.
Me.gridGroupingControll1.TopLevelGroupOptions.ShowFilterBar = True
i = 0
Do While i < gridGroupingControll1.TableDescriptor.Columns.Count
    gridGroupingControll1.TableDescriptor.Columns(i).AllowFilter = True
    i += 1
Loop
```

2. Add another form say **FilterDialog** to the project. This form will accept two filter conditions combined by a logical operator. Add a label to display the name of the filter column, two comboboxes to display the compare operator choices and two text boxes to accept the compare values, two radio buttons to determine logical combination of the filter conditions, and two buttons 'Ok' to pass the filter string to the parent grid and 'Cancel' to cancel the filter action. Here is a sample image of the form's designer.

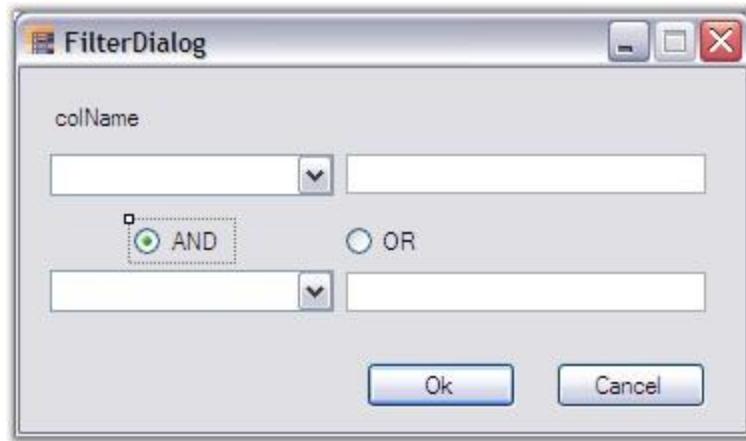


Figure 389: Creating FilterDialog Dialog Box

3. Add a method **SetupOptions** to fill the comboboxes with the operator choices. Inside the **Ok\_Button\_Click** event handler, form the filter string by writing the extracted values from the form in a valid syntax. Then close the form. The **CancelButton** would simply close the form.

[C#]

```

internal FilterString = "";

// Filling up the combo boxes with the operator options.
public void SetupOptions(string[] filteroptions)
{
    this.comboBox1.Items.AddRange(filteroptions);
    this.comboBox2.Items.AddRange(filteroptions);
}

// Code that is implemented on clicking the OK button.
private void button1_Click(object sender, EventArgs e)
{
    if (this.comboBox1.Text.Length > 0 && this.textBox1.Text.Length > 0)
    {
        FilterString = '[' + colLabel.Text + ']' +
GetFilter(this.comboBox1.SelectedIndex, this.textBox1.Text);
    }
    if (this.comboBox2.Text.Length > 0 && this.textBox2.Text.Length > 0)
    {
        FilterString += (this.radioButton1.Checked) ? " AND " : " OR ";
        FilterString += '[' + colLabel.Text + ']' +
GetFilter(this.comboBox2.SelectedIndex, this.textBox2.Text);
    }
}

```

```
        this.DialogResult = DialogResult.OK;
        this.Close();
    }

    private string GetFilter(int index, string text)
    {
        string s = "";

        switch (index)
        {
            case 0:
                s = " = '" + text + "'";
                break;
            case 1:
                s = " <> '" + text + "'";
                break;
            case 2:
                s = " > '" + text + "'";
                break;
            case 3:
                s = " >= '" + text + "'";
                break;
            case 4:
                s = " < '" + text + "'";
                break;
            case 5:
                s = " <= '" + text + "'";
                break;
        }
        return s;
    }

    // Code that is implemented on clicking the Cancel button.
    private void button2_Click(object sender, EventArgs e)
    {
        this.Close();
    }
}
```

**[VB.NET]**

```
Friend FilterString As String = ""

' Filling up the combo boxes with the operator options.
Public Sub SetupOptions(ByVal filteroptions As String())
    Me.comboBox1.Items.AddRange(filteroptions)
    Me.comboBox2.Items.AddRange(filteroptions)
End Sub

' Code that is implemented on clicking the OK button.
Private Sub button1_Click(ByVal sender As Object, ByVal e As EventArgs)
    If Me.comboBox1.Text.Length > 0 AndAlso Me.textBox1.Text.Length > 0
    Then
        FilterString = "[" + colLabel.Text & "] " + 
        GetFilter(Me.comboBox1.SelectedIndex, Me.textBox1.Text)
    End If
    If Me.comboBox2.Text.Length > 0 AndAlso Me.textBox2.Text.Length > 0
    Then
        If (Me.radioButton1.Checked) Then
            FilterString &= " AND "
        Else
            FilterString &= " OR "
        End If
        FilterString &= "[" + colLabel.Text & "] " + 
        GetFilter(Me.comboBox2.SelectedIndex, Me.textBox2.Text)
    End If
    Me.DialogResult = DialogResult.OK
    Me.Close()
End Sub

Private Function GetFilter(ByVal index As Integer, ByVal text As
String) As String
    Dim s As String = ""

    Select Case index
        Case 0
            s = " = '" + text + "'"
        Case 1
            s = " <> '" + text + "'"
        Case 2
            s = " > '" + text + "'"
        Case 3
            s = " >= '" + text + "'"
        Case 4
            s = " < '" + text + "'"
        Case 5
    End Select
    Return s
End Function
```

```
s = " <= '" + text + "'"
End Select
Return s
End Function

' Code that is implemented on clicking the Cancel button.
Private Sub button2_Click(ByVal sender As Object, ByVal e As EventArgs)
    Me.Close()
End Sub
```

4. Once a filter dialog has been designed, our next step is to create the required model and renderer classes in order to adopt the custom filter dialog. Inside the renderer, override the

ListBoxMouseUp event to make the custom filter dialog to pop up. When the dialog is closed, add the filter string into the RecordFilters collection to set up the filter.

**[C#]**

```
// Cell Model class.  
class CustomFilterCellModel : GridTableFilterBarCellModel  
{  
    internal GridFilterBar filterBar;  
  
    public CustomFilterCellModel(GridModel grid) : base(grid)  
    {  
        filterBar = new GridFilterBar();  
    }  
  
    public override GridCellRendererBase CreateRenderer(GridControlBase control)  
    {  
        return new CustomFilterCellRenderer(control, this);  
    }  
  
    public string[] FilterBarOptions = new string[]  
    {  
        "equals",  
        "does not equal",  
        "is greater than",  
        "is greater than or equal to",  
        "is less than",  
        "is less than or equal to"  
    };  
}  
  
// Cell Renderer class.  
class CustomFilterCellRenderer : GridTableFilterBarCellRenderer  
{  
    CustomFilterCellModel filterModel;  
  
    public CustomFilterCellRenderer(GridControlBase grid,  
        GridCellModelBase cellModel)  
        : base(grid, cellModel)  
    {  
        filterModel = cellModel as CustomFilterCellModel;  
    }  
  
    protected override void ListBoxMouseUp(object sender,  
        MouseEventArgs e)  
    {
```

```
    GridTableCellStyleInfo tableStyleInfo =
(GridTableCellStyleInfo) StyleInfo;
    GridTableCellStyleInfoIdentity tableCellIdentity =
tableStyleInfo.TableCellIdentity;
    if( this.ListBoxPart.SelectedIndex == 1)
    {
        FilterDialog dlg = new FilterDialog();
        dlg.colLabel.Text = tableCellIdentity.Column.MappingName;
        dlg.SetupOptions(filterModel.FilterBarOptions);
        if(dlg.ShowDialog() == DialogResult.OK)
        {
            // Apply the filter.

tableCellIdentity.Table.TableDescriptor.RecordFilters.Add(dlg.FilterString);
        }
    }
    else
    {
        if( this.ListBoxPart.SelectedIndex == 0)

tableCellIdentity.Table.TableDescriptor.RecordFilters.Clear();
        base.ListBoxMouseUp(sender, e);
    }
}
}
```

**[VB.NET]**

```
' Cell Model class.
Friend Class CustomFilterCellModel : Inherits
GridTableFilterBarCellModel
    Friend filterBar As GridFilterBar

    Public Sub New(ByVal grid As GridModel)
        MyBase.New(grid)
        filterBar = New GridFilterBar()
    End Sub

    Public Overrides Function CreateRenderer(ByVal control As
GridControlBase) As GridCellRendererBase
        Return New CustomFilterCellRenderer(control, Me)
    End Function

    Public FilterBarOptions As String() = New String() {"equals", "does
not equal", "is greater than", "is greater than or equal to",
"is less than", "is less than or equal to"}
```

```
End Class

' Cell Renderer class.
Friend Class CustomFilterCellRenderer : Inherits
GridTableFilterBarCellRenderer
    Private filterModel As CustomFilterCellModel

    Public Sub New(ByVal grid As GridControlBase, ByVal cellModel As
GridCellModelBase)
        MyBase.New(grid, cellModel)
        filterModel = CType(IIf(TypeOf cellModel Is
CustomFilterCellModel, cellModel, Nothing),
CustomFilterCellModel)
    End Sub

Protected Overrides Sub ListBoxMouseUp(ByVal sender As Object, ByVal
e As MouseEventArgs)
    Dim tableStyleInfo As GridTableCellStyleInfo = CType(StyleInfo,
GridTableCellStyleInfo)
    Dim tableCellIdentity As GridTableCellStyleInfoIdentity =
tableStyleInfo.TableCellIdentity
    If Me.ListBoxPart.SelectedIndex = 1 Then

        Dim dlg As FilterDialog = New FilterDialog()
        dlg.colLabel.Text = tableCellIdentity.Column.MappingName
        dlg.SetupOptions(filterModel.FilterBarOptions)
        If dlg.ShowDialog() = DialogResult.OK Then

            ' Apply the filter.
            tableCellIdentity.Table.TableDescriptor.RecordFilters.Add(d
lg.FilterString)

        End If
    Else
        If Me.ListBoxPart.SelectedIndex = 0 Then
            tableCellIdentity.Table.TableDescriptor.RecordFilters.Clear
()
        End If
        MyBase.ListBoxMouseUp(sender, e)
    End If
End Sub
End Class
```

5. Our final step is to replace the **default filter bar cell** with our **custom filter bar cell**. This can be done by registering our new cell model class into the existing cell model's collection by specifying a name for the new cell type. Then set the cell type of the FilterBarCell to this new cell type.

[C#]

```
this.gridGroupingControl1.TableModel.CellModels.Add("MyFilterCell", new  
CustomFilterCellModel(this.gridGroupingControl1.TableModel));  
this.gridGroupingControl1.Appearance.FilterBarCell.CellType =  
"MyFilterCell";
```

[VB.NET]

```
Me.gridGroupingControl1.TableModel.CellModels.Add("MyFilterCell", New  
CustomFilterCellModel(Me.gridGroupingControl1.TableModel))  
Me.gridGroupingControl1.Appearance.FilterBarCell.CellType =  
"MyFilterCell"
```

- When you run the sample, click the filter drop down and then select custom option to show up our filter dialog. This is shown in the below screenshots.

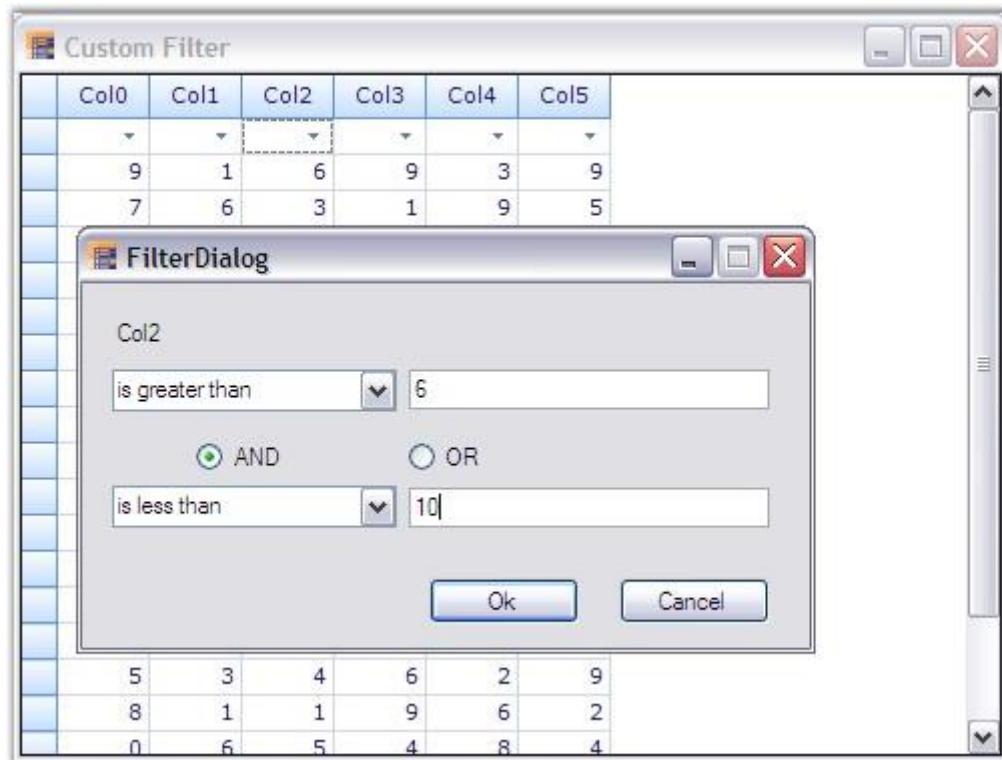


Figure 390: Image showing the Custom FilterDialog dialog box with Filter Options Entered

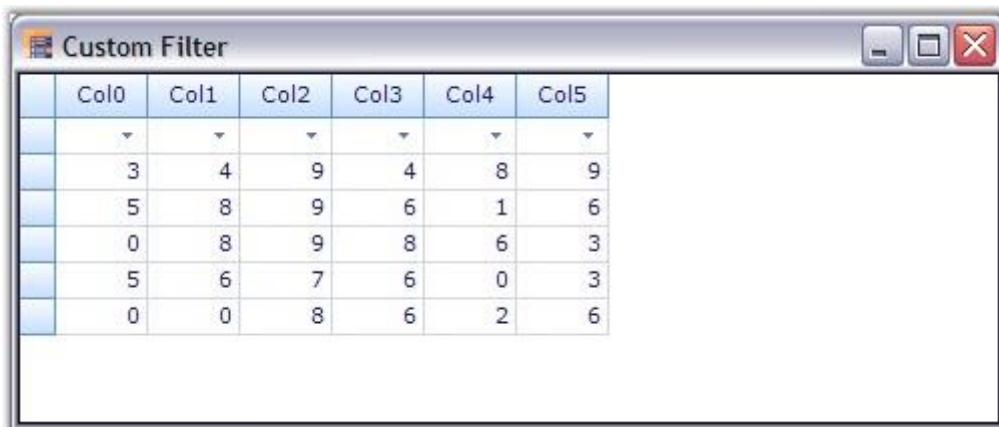


Figure 391: Filtered Grid

#### 4.3.4.13.5 Filter Optimization

The filter can be optimized for the performance over large data in **GridGroupingControl**.

Set **OptimizeFilterPerformance** to **true** to optimize the filter.

This boolean property enables the grid to optimize the filtering of data by delegating the summary to filter the grid accordingly.

The **OptimizeFilterPerformance** property can be assigned directly from the **GridGroupingControl** and can apply to any type of filter bar specified to the grid.

The following code illustrates how to optimize the filter.

[C#]

```
private void Form1_Load(object sender, EventArgs e)
{
    this.gridGroupingControl1.OptimizeFilterPerformance = true;
}
```

[VB]

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As
EventArgs)
    Me.gridGroupingControl1.OptimizeFilterPerformance = True
End Sub
```

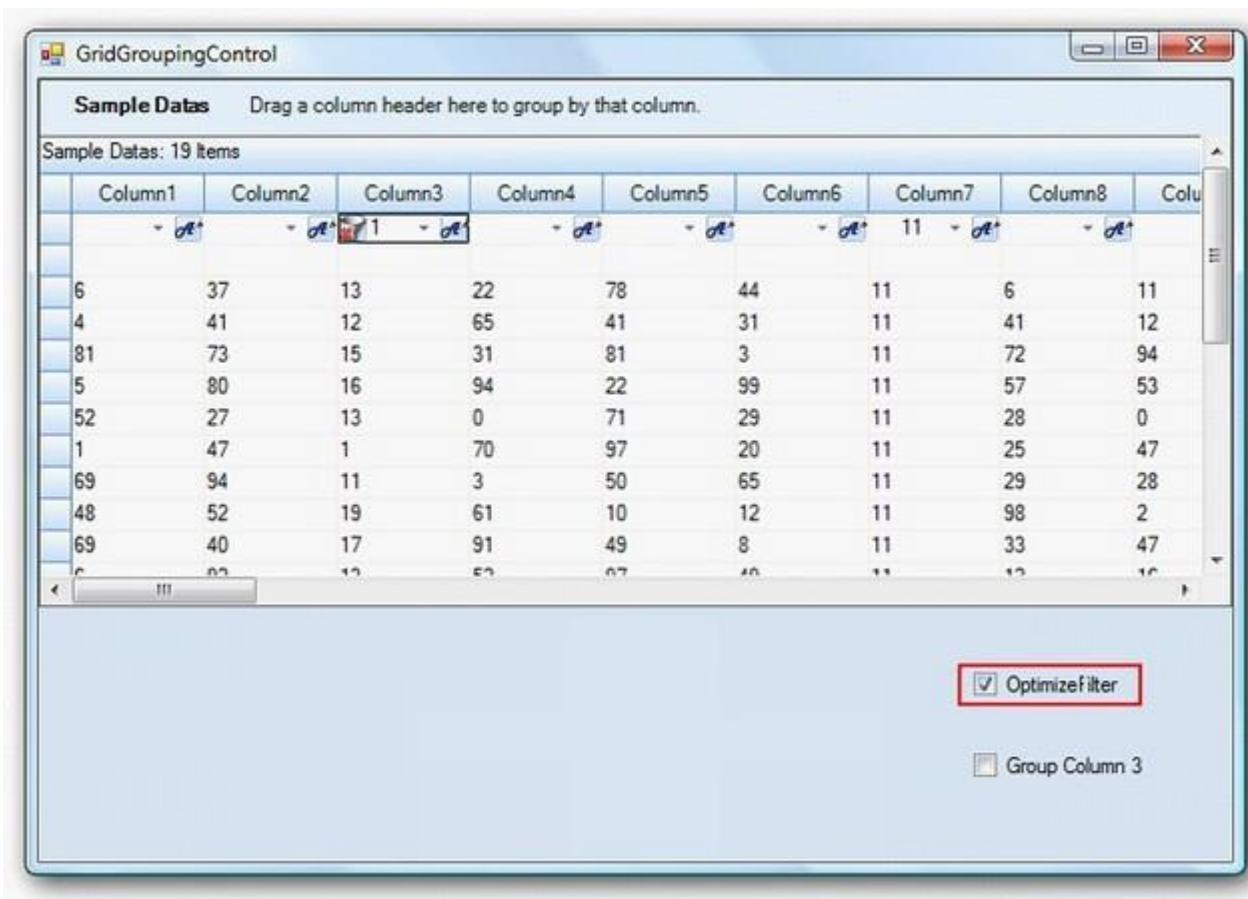


Figure 392: Optimize Filter

#### 4.3.4.14 Events

An Event is a message that is triggered to notify an object or a class of the occurrence of an action. When an event gets fired, all the event handlers will be notified i.e. the event handler functions will be invoked.

In order to receive the event notifications, the caller object must subscribe to the desired events. If you no longer want to listen to an event, then you can simply unsubscribe from the event notification. This section deals with some important events that are fired while performing group related actions. They are organized into different categories based primarily on the grid component they affect and the functionality it provides.

In this section, you will learn about the following events.

#### 4.3.4.14.1 Table Events

The Table Style events are as follows:

**BackColorChanged**-Occurs when the value of Control's BackColor property changes.

**[C#]**

```
this.groupingEngine.TableControl.BackColorChanged+=new  
EventHandler(TableControl_BackColorChanged);
```

**[VB .NET]**

```
AddHandler groupingEngine.TableControl.BackColorChanged, AddressOf  
TableControl_BackColorChanged
```

**BackgroundImageChanged**-Occurs when the value of Control's BackgroundImage property changes.

**[C#]**

```
this.groupingEngine.TableControl.BackgroundImageChanged+=new  
EventHandler(TableControl_BackgroundImageChanged);
```

**[VB .NET]**

```
AddHandler groupingEngine.TableControl.BackgroundImageChanged,  
AddressOf TableControl_BackgroundImageChanged
```

**BindingContextChanged**-Occurs when the value of Control's BindingContextChanged property.

**[C#]**

```
this.groupingEngine.TableControl.BindingContextChanged+=new  
EventHandler(TableControl_BindingContextChanged);
```

**[VB .NET]**

```
AddHandler groupingEngine.TableControl.BindingContextChanged, AddressOf  
TableControl_BindingContextChanged
```

**QueryAllowDragColumn**-Occurs when the user hovers the mouse over a column header or clicks on it. In your event handler, you can determine if the selected column can be dragged.

**[C#]**

```
this.groupingEngine.TableControl.QueryAllowDragColumn+=new  
GridQueryAllowDragColumnEventHandler(TableControl_QueryAllowDragColumn)  
;
```

**[VB .NET]**

```
AddHandler groupingEngine.TableControl.QueryAllowDragColumn, AddressOf  
TableControl_QueryAllowDragColumn
```

The event handler receives an argument of type **GridQueryAllowDragColumnEventArgs** containing data related to this event.

The following **GridQueryAllowDragColumnEventArgs** properties provide information specific to this event.

- **AllowDrag**-You can disallow dragging the column when you assign it to False.
- **Column**-Get the column Name.
- **Reason**-Provide reason why the event is raised.

**QueryAllowGroupByColumn**-Occurs when the user drags a column header over the GroupDropArea. In your event handler, you can determine if the grid can be grouped by the selected column.

**[C#]**

```
this.groupingEngine.TableControl.QueryAllowGroupByColumn+=new  
GridQueryAllowGroupByColumnEventHandler(TableControl_QueryAllowGroupByC  
olumn);
```

**[VB .NET]**

```
AddHandler groupingEngine.TableControl.QueryAllowGroupByColumn,  
AddressOf TableControl_QueryAllowGroupByColumn
```

The event handler receives an argument of type **GridQueryAllowGroupByColumnEventArgs** containing data related to this event.

The following GridQueryAllowGroupByColumnEventArgs properties provide information specific to this event.

- **AllowGroupByColumn**-You can disallow grouping by the column when you assign False to it.
- **Column**-Get the column Name.
- **Reason**-Provide reason why the event is raised.

**QueryAllowSortColumn**-Occurs when the user hovers the mouse over a column header or clicks on it. In your event handler you can determine if the selected column can be sorted.

**[C#]**

```
this.groupingEngine.TableControl.QueryAllowSortColumn+=new  
GridQueryAllowSortColumnEventHandler(TableControl_QueryAllowSortColumn)  
;
```

**[VB .NET]**

```
AddHandler groupingEngine.TableControl.QueryAllowSortColumn, AddressOf  
TableControl_QueryAllowSortColumn
```

The event handler receives an argument of type GridQueryAllowSortColumnEventArgs containing data related to this event.

The following GridQueryAllowSortColumnEventArgs properties provide information specific to this event.

- **AllowSort**-You can disallow sorting by the column when you assign False to it.
- **CellClickEventArgs**-You can check the CellClickEventArgs to find out which mouse button was clicked or the exact position of the mouse pointer.
- **Column**-Get the column Name.

**Resize**-Occurs when control is resized.

**[C#]**

```
this.groupingEngine.TableControl.Resize+=new  
EventHandler(TableControl_Resize);
```

**[VB .NET]**

```
AddHandler groupingEngine.TableControl.Resize, AddressOf
```

TableControl\_Resize

**ResizingColumns** – Occurs when the user resizes a selected range of columns.

The event handler receives an argument of type GridResizingColumnsEventArgs containing data related to this event.

The following **GridResizingColumnsEventArgs** properties provide information specific to this event.

- **Width**-The new width of the columns.
- **Reason**-The originating reason for this event.
- **SizeIndicatorBorder**-The appearance of the line that indicates the new size.
- **Point**-The mouse location.

**TextChanged**-Occurs when value of Control's text property changes.

[C#]

```
this.groupingEngine.TableControl.TextChanged+=new  
EventHandler(TableControl_TextChanged);
```

[VB .NET]

```
AddHandler groupingEngine.TableControl.TextChanged, AddressOf  
TableControl_TextChanged
```

**TopRowChanged** - Occurs after the grid has been scrolled when the top row index is changed.

[C#]

```
this.groupingEngine.TableControl.TopRowChanged+=new  
Syncfusion.Windows.Forms.Grid.GridRowColIndexChangedEventHandler(TableC  
ontrol_TopRowChanged);
```

[VB .NET]

```
AddHandler groupingEngine.TableControl.TopRowChanged, AddressOf  
TableControl_TopRowChanged
```

The event handler receives an argument of type **GridRowCollIndexChangedEventArgs** containing data related to this event.

The following **GridRowCollIndexChangedEventArgs** properties provide information specific to this event.

- **SavedValue**-The saved `Syncfusion.Windows.Forms.Grid.GridControlBase.TopRowIndex` or `Syncfusion.Windows.Forms.Grid.GridControlBase.LeftCollIndex` value.
- **Success**-Indicates if operation was ended successfully or was aborted.

In this section, you will learn about the following events.

#### *4.3.4.14.1.1 CurrentRecordContextChangeEvent*

It occurs before and after the status of the current record, when it is changed.

The event handler receives an argument of type **CurrentRecordContextChangeEventArgs** containing data related to this event.

Property	Description
Action	Specifies how the current record context changes.
Cancel	Gets or sets a value indicating whether the event should be canceled.
Record	Specifies the affected record.
Success	Specifies if the operation was completed successfully or failed.
Table	Specifies the table that <code>Grouping.Table.SelectedRecords</code> collection belongs to.

You can handle this event and get the updated expression field value. The following code example illustrates this.

[C#]

```
private void gridGroupingControll1_CurrentRecordContextChange(object
sender, CurrentRecordContextChangeEvent e)
{
    if (e.Action == CurrentRecordAction.EndEditComplete)
        Console.WriteLine(e.Record);
}
```

**[VB.NET]**

```
Private Sub gridGroupingControll1_CurrentRecordContextChange(ByVal
sender As Object, ByVal e As CurrentRecordContextChangeEvent)
    If e.Action = CurrentRecordAction.EndEditComplete Then
        Console.WriteLine(e.Record)
    End If
End Sub
```

**4.3.4.14.1.2 GroupAdded Event**

It occurs when a new group is added in the table after the table was categorized and when a record is changed. This event does not occur during the categorization of the table.

The event handler receives an argument of type **GroupEventArgs** containing data related to this event.

Property	Description
Cancel	Gets or sets a value indicating whether the event should be canceled.
Group	Gets the affected group.

You can handle this event to make the child groups that are not initially expanded. The following code example illustrates this.

**[C#]**

```
private void gridGroupingControll1_GroupAdded(object sender,
GroupEventArgs e)
{
    e.Group.IsExpanded = false;
}
```

**[VB.NET]**

```
Private Sub gridGroupingControll_GroupAdded(ByVal sender As Object,
 ByVal e As GroupEventArgs)
    e.Group.IsExpanded = False
End Sub
```

**4.3.4.14.1.3 GroupCollapsing Event**

It occurs before a group is collapsed.

The event handler receives an argument of type **GroupEventArgs** containing data related to this event.

Property	Description
Cancel	Gets or sets a value indicating whether the event should be canceled.
Group	Gets the affected group.

**[C#]**

```
this.gridGroupingControll.GroupCollapsing += new
GroupEventHandler(this.gridGroupingControll_GroupCollapsing);

private void gridGroupingControll_GroupCollapsing(object sender,
GroupEventArgs e)
{
    foreach(Record r in e.Group.Records)
        Console.WriteLine("Collapsing event "+r.Info);
}
```

**[VB.NET]**

```
AddHandler gridGroupingControll.GroupCollapsing, AddressOf
gridGroupingControll_GroupCollapsing

Private Sub gridGroupingControll_GroupCollapsing(ByVal sender As
Object, ByVal e As GroupEventArgs)
For Each r As Record In e.Group.Records
```

```
Console.WriteLine("Collapsing event "+r.Info)
Next r
End Sub
```

#### 4.3.4.14.1.4 GroupCollapsed Event

It occurs when a group is collapsed.

The event handler receives an argument of type **GroupEventArgs** containing data related to this event.

Property	Description
Cancel	Gets or sets a value indicating whether the event should be canceled.
Group	Gets the affected group.

##### [C#]

```
this.gridGroupingControl1.GroupCollapsed += new
GroupEventHandler(this.gridGroupingControl1_GroupCollapsed);

private void gridGroupingControl1_GroupCollapsed(object sender,
GroupEventArgs e)
{
    foreach(Record r in e.Group.Records)
        Console.WriteLine("Collapsed event "+r.Info);
}
```

##### [VB .NET]

```
AddHandler gridGroupingControl1.GroupCollapsed, AddressOf
gridGroupingControl1_GroupCollapsed

Private Sub gridGroupingControl1_GroupCollapsed(ByVal sender As Object,
ByVal e As GroupEventArgs)
    For Each r As Record In e.Group.Records
        Console.WriteLine("Collapsed event "+r.Info)
Next r
End Sub
```

#### 4.3.4.14.1.5 GroupExpanding Event

It occurs before a group is expanded.

The event handler receives an argument of type **GroupEventArgs** containing the data related to this event.

Property	Description
Cancel	Gets or sets a value indicating whether the event should be canceled.
Group	Gets the affected group.

You can allow the user to expand the group, by clicking the plus/minus button. The following code example illustrates this.

##### [C#]

```
bool IsClickExpand = false;

private void gridGroupingControll1_GroupExpanding(object sender,
GroupEventArgs e)
{
    if (IsClickExpand)
        IsClickExpand = false;
    else
        e.Cancel = true;
}
```

##### [VB.NET]

```
Private IsClickExpand As Boolean = False

Private Sub gridGroupingControll1_GroupExpanding(ByVal sender As Object,
ByVal e As GroupEventArgs)
    If IsClickExpand Then
        IsClickExpand = False
    Else
        e.Cancel = True
    End If

```

```
End Sub
```

#### 4.3.4.14.1.6 GroupExpanded Event

It occurs when a group is expanded.

The event handler receives an argument of type **GroupEventArgs** containing data related to this event.

Property	Description
Cancel	Gets or sets a value indicating whether the event should be canceled.
Group	Gets the affected group.

#### [C#]

```
this.gridGroupingControl1.GroupExpanded += new
GroupEventHandler(this.gridGroupingControl1_GroupExpanded);

private void gridGroupingControl1_GroupExpanded(object sender,
GroupEventArgs e)
{
    foreach(Record r in e.Group.Records)
        Console.WriteLine("Expanded event "+ r.Info);
}
```

#### [VB .NET]

```
AddHandler gridGroupingControl1.GroupExpanded, AddressOf
gridGroupingControl1_GroupExpanded

Private Sub gridGroupingControl1_GroupExpanded(ByVal sender As Object,
ByVal e As GroupEventArgs)
    For Each r As Record In e.Group.Records
        Console.WriteLine("Expanded event "+ r.Info)
    Next r
End Sub
```

#### 4.3.4.14.1.7 GroupRemoving Event

It occurs before a group is removed after the table was categorized and when a record is changed. This event does not occur during the categorization of the table.

The event handler receives an argument of type **GroupEventArgs** containing data related to this event.

Property	Description
Cancel	Gets or sets a value indicating whether the event should be canceled.
Group	Gets the affected group.

**[C#]**

```
this.gridGroupingControl1.GroupExpanded += new  
GroupEventHandler(this.gridGroupingControl1_GroupExpanded);  
  
private void gridGroupingControl1_GroupExpanded(object sender,  
GroupEventArgs e)  
{  
    foreach(Record r in e.Group.Records)  
        Console.WriteLine("Expanded event "+ r.Info);  
}
```

**[VB .NET]**

```
AddHandler gridGroupingControl1.GroupRemoving, AddressOf  
gridGroupingControl1_GroupRemoving  
  
Private Sub gridGroupingControl1_GroupRemoving(ByVal sender As Object,  
ByVal e As GroupEventArgs)  
    For Each r As Record In e.Group.Records  
        Console.WriteLine("Expanded event "+ r.Info)  
    Next r  
End Sub
```

#### 4.3.4.14.1.8 GroupSummaryInvalidate Event

It occurs when a summary has been marked dirty.

The event handler receives an argument of type **GroupEventArgs** containing data related to this event.

Property	Description
Cancel	Gets or sets a value indicating whether the event should be canceled.
Group	Gets the affected group.

You can handle this event to get the updated summary value in a grid. The following code example illustrates this.

#### [C#]

```
private void gridGroupingControl1_GroupSummaryInvalidated(object
sender, GroupEventArgs e)
{
    e.Group.ParentTable.SummariesDirty = true;
    GridGroupingControl grid = sender as GridGroupingControl;
    GridTableControl tc =
grid.GetTableControl(e.Group.ParentTableDescriptor.Name);
    tc.RefreshRange(GridRangeInfo.Cell(8, 3));

    // Get the updated summary in a Text Box.
    textbox1.Text = tc.Model[8, 3].Text;
}
```

#### [VB .NET]

```
Private Sub gridGroupingControl1_GroupSummaryInvalidated(ByVal sender
As Object, ByVal e As GroupEventArgs)
    e.Group.ParentTable.SummariesDirty = True
    Dim grid As GridGroupingControl = CType(IIf(TypeOf sender Is
GridGroupingControl, sender, Nothing), GridGroupingControl)
    Dim tc As GridTableControl =
grid.GetTableControl(e.Group.ParentTableDescriptor.Name)
    tc.RefreshRange(GridRangeInfo.Cell(8, 3))

    ' Get the updated summary in a Text Box.
    textbox1.Text = tc.Model(8, 3).Text
End Sub
```

#### 4.3.4.14.1.9 RecordExpanding Event

It occurs before a record with nested table is expanded.

The event handler receives an argument of type **RecordEventArgs** containing data related to this event.

Property	Description
Cancel	Gets or sets a value indicating whether the event should be canceled.
Record	Gets the affected record.

The parent records that have no child record can be prevented from expanding by handling the Table.RecordExpanding event and by setting the e.Cancel to true, when there is no child records for a parent record.

##### [C#]

```
private void Table_RecordExpanding(object sender, RecordEventArgs e)
{
    if(e.Record.GetRelatedChildTable(new
RelationDescriptor("ParentToChild")).GetRecordCount() == 0)
    e.Cancel = true;
}
```

##### [VB .NET]

```
Private Sub Table_RecordExpanding(ByVal sender As Object, ByVal e As
RecordEventArgs)
    If e.Record.GetRelatedChildTable(New
RelationDescriptor("ParentToChild")).GetRecordCount() = 0 Then
        e.Cancel = True
    End If
End Sub
```

#### 4.3.4.14.1.10 RecordValueChanging Event

It occurs when a RecordFieldCell cell's value is changed and before Record.SetValue is returned.

The event handler receives an argument of type **RecordValueChangingEventArgs** containing data related to this event.

Property	Description
Cancel	Gets or sets a value indicating whether the event should be canceled.
Column	Specifies the column.
FieldDescriptor	Specifies the field descriptor.
NewValue	Specifies the new value to save.
Record	Specifies the record.

You can handle this event to compare the old and new values of the cell when cursor leaves the cell. The following code example illustrates this.

#### [C#]

```
private void gridGroupingControll1_RecordValueChanging(object sender,
RecordValueChangingEventArgs e)
{
    Console.WriteLine(" New Value: {0} ", e.NewValue);
    Console.WriteLine(" Old Value: {0} ", e.Record.GetValue(e.Column));
}
```

#### [VB.NET]

```
Private Sub gridGroupingControll1_RecordValueChanging(ByVal sender As
Object, ByVal e As RecordValueChangingEventArgs)
    Console.WriteLine(" New Value: {0} ", e.NewValue)
    Console.WriteLine(" Old Value: {0} ", e.Record.GetValue(e.Column))
End Sub
```

#### 4.3.4.14.1.11 RecordValueChanged Event

It occurs when a RecordFieldCell cell's value is changed and after Record.SetValue is returned.

The event handler receives an argument of type **RecordValueChangedEventArgs** containing data related to this event.

Property	Description
Action	Specifies how the current record context changes.
Cancel	Gets or sets a value indicating whether the event should be canceled.
Record	Specifies the affected record.
Success	Specifies if the operation was completed successfully or failed.
Table	Specifies the affected table.

You can handle this event to save the current record value. The following code example illustrates this.

#### [C#]

```
private void gridGroupingControl1_RecordValueChanged(object sender,
RecordValueChangedEventArgs e)
{
    Record r = e.Record ;
    if(e.Column != "Column2")
        r.SetValue("Column2", "");
}
```

#### [VB .NET]

```
Private Sub gridGroupingControl1_RecordValueChanged(ByVal sender As
Object, ByVal e As RecordValueChangedEventArgs)
    Dim r As Record = e.Record
    If e.Column <> "Column2" Then
        r.SetValue("Column2", "")
    End If
End Sub
```

#### 4.3.4.14.1.12 *SortingItemsInGroup* Event

It occurs before the records for a group are sorted.

The event handler receives an argument of type **GroupEventArgs** containing data related to this event.

Property	Description
Cancel	Gets or sets a value indicating whether the event should be canceled.
Group	Gets the affected group.

##### [C#]

```
void Table_SortingItemsInGroup(object sender, GroupEventArgs e)
{
    Console.WriteLine("Before Sorting the records in the group: "+
        e.Group.Info);
}
```

##### [VB.NET]

```
Private Sub Table_SortingItemsInGroup(ByVal sender As Object, ByVal e
As GroupEventArgs)
    Console.WriteLine("Before Sorting the records in the group: "+
        e.Group.Info)
End Sub
```

#### 4.3.4.14.1.13 *SortedItemsInGroup* Event

It occurs after the records for a group are sorted.

The event handler receives an argument of type **GroupEventArgs** containing data related to this event.

Property	Description
Cancel	Gets or sets a value indicating whether the event should be canceled.

Group	Gets the affected group.
-------	--------------------------

**[C#]**

```
void Table_SortedItemsInGroup(object sender, GroupEventArgs e)
{
    Console.WriteLine("After Sorting the records in the group: " +
e.Group.Info);
}
```

**[VB .NET]**

```
Private Sub Table_SortedItemsInGroup(ByVal sender As Object, ByVal e As
GroupEventArgs)
    Console.WriteLine("After Sorting the records in the group: " +
e.Group.Info)
End Sub
```

**4.3.4.14.1.14 SourceListListChanged Event**

It occurs before the table processes the System.ComponentModel.IBindingList.ListChanged event of attached source list.

The event handler receives an argument of type **TableListChangedEventArgs** containing data related to this event.

Property	Description
ListChangedType	Gets the type of change.
NavigateCurrentRecordWhenDeleted	Specifies whether the current record should be moved to the previous visible record when ItemDeleted notification is received for the current record.
NewIndex	Gets the index of the item affected by change.
OldIndex	Gets the old index of an item that has been moved.
PropertyDescriptor	Gets the PropertyDescriptor that was added,

	changed or deleted.
ShouldIgnoreReset	Specifies whether ListChangedType.Reset notification should be ignored.
ShouldInvalidateCounters	Specifies whether counters need to be marked dirty when ListChanged event is handled.
ShouldInvalidateGroupSortOrder	Specifies whether changes to the current record affect the sort order of the current group.
ShouldInvalidateScreen	Specifies whether grid should repaint the record when changes to the record were made in the underlying data source and ListChangedType.ItemChanged notification was raised.
ShouldInvalidateSummaries	Specifies whether summaries need to be marked dirty when ListChanged event is handled.
ShouldReevaluateSortPosition	Specifies whether changes to the current record affects the sort order of the current group.
ShouldResetCurrentRecord	Specifies whether current record should be reset when ItemChanged notification is received for the current record.

You can handle this event to retrieve the ListChangedType. The following code example illustrates this.

**[C#]**

```
private void gridGroupingControll1_SourceListListChanged(object sender,
TableListChangedEventArgs e)
{
    Console.WriteLine("ListChangedType :" + e.ListChangedType.ToString());
}
```

**[VB .NET]**

```
Private Sub gridGroupingControll1_SourceListListChanged(ByVal sender As
Object, ByVal e As TableListChangedEventArgs)
    Console.WriteLine("ListChangedType :" &
```

```
e.ListChangedType.ToString())
End Sub
```

#### 4.3.4.14.2 Cell Events

The Cell events are as follows:

**CellButtonClicked**-Occurs when the user has clicked on a child button element inside a cell renderer.

[C#]

```
this.groupingEngine.TableControl.CellButtonClicked+=new
Syncfusion.Windows.Forms.Grid.GridCellButtonClickedEventHandler(TableCo
ntrol_CellButtonClicked);
```

[VB .NET]

```
AddHandler groupingEngine.TableControl.CellButtonClicked, AddressOf
TableControl_CellButtonClicked
```

The event handler receives an argument of type GridCellButtonClickedEventArgs containing data related to this event.

The following GridCellButtonClickedEventArgs properties provide information specific to this event.

- **RowIndex**-Gets the row index.
- **ColIndex**-Gets the column index.
- **ButtonIndex**-The index of the clicked cell button element.

**CellClick**-Occurs when the user clicks inside a cell.

[C#]

```
this.groupingEngine.TableControl.CellClick+=new
Syncfusion.Windows.Forms.Grid.GridCellClickEventHandler(TableControl_Ce
llClick);
```

**[VB .NET]**

```
AddHandler groupingEngine.TableControl.CellClick, AddressOf  
TableControl_CellClick
```

The event handler receives an argument of type GridCellEventArgs containing data related to this event.

The following GridCellEventArgs properties provide information specific to this event.

- **RowIndex**-Gets the row index.
- **ColumnIndex**-Gets the column index.
- **MouseEventArgs**-The System.Windows.Forms.MouseEventArgs originating this event.
- **IsOverImage**-Indicates if the mouse was over a image in static cell when the mouse was released.

**CellDoubleClick**-Occurs when the user double-clicks inside a cell.

**[C#]**

```
this.groupingEngine.TableControl.CellDoubleClick+=new  
Syncfusion.Windows.Forms.Grid.GridCellClickEventHandler(TableControl_Ce  
llDoubleClick);
```

**[VB .NET]**

```
AddHandler groupingEngine.TableControl.CellDoubleClick, AddressOf  
TableControl_CellDoubleClick
```

The event handler receives an argument of type GridCellEventArgs containing data related to this event.

The following GridCellEventArgs properties provide information specific to this event.

- **RowIndex**-Gets the row index.
- **ColumnIndex**-Gets the column index.
- **MouseEventArgs**-The System.Windows.Forms.MouseEventArgs originating this event.
- **IsOverImage**-Indicates if the mouse was over a image in static cell when the mouse was released.

**CellDrawn**-Occurs for every cell after the grid has drawn the specified cell.

**[C#]**

```
this.groupingEngine.TableControl.CellDrawn+=new  
Syncfusion.Windows.Forms.Grid.GridDrawCellEventHandler(TableControl_Cel  
lDrawn);
```

**[VB .NET]**

```
AddHandler groupingEngine.TableControl.CellDrawn, AddressOf  
TableControl_CellDrawn
```

The event handler receives an argument of type **GridDrawCellEventArgs** containing data related to this event.

The following **GridDrawCellEventArgs** properties provide information specific to this event.

- **Graphics**-Gets the Graphics context.
- **Renderer**-Gets the cell renderer.
- **Bounds**-Gets the Cell boundaries.
- **RowIndex**-Gets the row index.
- **ColIndex**-The column index.
- **Style**-The **Syncfusion.Windows.Forms.Grid.GridStyleInfo** object that holds cell information.
- **IsBackgroundErased**-True if the cell background has already been drawn with the interior as specified in the style object.

In this section, you will learn about the following events.

#### *4.3.4.14.2.1 CellButtonClicked Event*

It occurs when a user has clicked on a child button element inside the cell renderer. The event handler receives an argument of type **GridCellButtonClickedEventArgs** containing data related to this event. This event can be invoked as follows.

**[C#]**

```
this.gridGroupingControl1.TableControlCellButtonClicked+=new  
GridCellButtonClickedEventHandler(TableControl_CellButtonClicked);
```

**[VB .NET]**

```
AddHandler Me.gridGroupingControl1.TableControlCellButtonClicked,  
AddressOf TableControl_CellButtonClicked
```

The following **GridCellButtonClickedEventArgs** properties provide information specific to this event.

Properties	Description
Button	A reference to the GridCellButton for the clicked button.
ButtonIndex	The index of the cell button clicked.
ColIndex	Specifies the column index of the cell.
RowIndex	Specifies the row index of the cell.

**Example****[C#]**

```
private void TableControl_CellButtonClicked(object sender,  
GridCellButtonClickedEventArgs e)  
{  
    Console.WriteLine("CellButtonClicked");  
}
```

**[VB .NET]**

```
Private Sub TableControl_CellButtonClicked(ByVal sender As Object,  
ByVal e As GridCellButtonClickedEventArgs)  
    Console.WriteLine("CellButtonClicked")  
End Sub
```

#### 4.3.4.14.2.2 CellClick Event

It occurs when the user clicks inside a cell. The event handler receives an argument of type **GridCellClickEventArgs** containing data related to this event. This event can be invoked as follows.

**[C#]**

```
this.gridGroupingControl1.TableControlCellClick+=new  
Syncfusion.Windows.Forms.Grid.GridCellClickEventHandler(TableControl_Ce  
llClick);
```

**[VB .NET]**

```
AddHandler Me.gridGroupingControl1.TableControlCellClick, AddressOf  
TableControl_CellClick
```

The following **GridCellEventArgs** properties provide information specific to this event.

Properties	Description
IsOverImage	Indicates if the mouse was over the image in a static cell when the mouse was released.
MouseEventArgs	The MouseEventArgs originating this event.
ColIndex	Specifies the column index of the cell.
RowIndex	Specifies the row index of the cell.
Cancel	Specifies a value to indicate if this event should be canceled.

**Example**

**[C#]**

```
private void TableControl_CellClick(object sender,  
GridCellEventArgs e)  
{  
    Console.WriteLine("CellClicked");  
}
```

**[VB .NET]**

```
Private Sub TableControl_CellClick(ByVal sender As Object, ByVal e As  
GridCellEventArgs)  
    Console.WriteLine("CellClicked")  
End Sub
```

#### 4.3.4.14.2.3 CellDrawn Event

It occurs for every cell after the grid has drawn the specified cell. The event handler receives an argument of type **GridDrawCellEventArgs** containing data related to this event. This event can be invoked as follows.

**[C#]**

```
this.gridGroupingControl1.TableControlCellDrawn+=new
Syncfusion.Windows.Forms.Grid.GridDrawCellEventHandler(TableControl_Cel
lDrawn);
```

**[VB .NET]**

```
AddHandler Me.gridGroupingControl1.TableControlCellDrawn, AddressOf
TableControl_CellDrawn
```

The following GridTableControlCellDrawnEventArgs properties provide information specific to this event.

Properties	Description
Bounds	Cell boundaries including borders and margins.
Renderer	The GridCellRendererBase associated with the cell.
Graphics	The Graphics context.
IsBackgroundErased	Specifies if the cell background has already been drawn with the interior as specified in the style object.
ColIndex	Specifies the column index of the cell.
RowIndex	Specifies the row index of the cell.
Cancel	Specifies a value to indicate if this event should be canceled.

You can handle the TableControlCellDrawn to draw the error icon in an invalid cell as follows.

**[C#]**

```
private void TableControl_CellDrawn(object sender,
GridDrawCellEventArgs e)
{
    Console.WriteLine("Cell Drawn");
}
```

**[VB.NET]**

```
Private Sub TableControl_CellDrawn(ByVal sender As Object, ByVal e As
GridDrawCellEventArgs)
    Console.WriteLine("Cell Drawn")
End Sub
```

#### *4.3.4.14.2.4 CellDoubleClick Event*

It occurs when the user double-clicks inside a cell. The event handler receives an argument of type **GridCellClickEventArgs** containing data related to this event. This event can be invoked as follows.

**[C#]**

```
this.gridGroupingControl1.TableControlCellDoubleClick+=new
Syncfusion.Windows.Forms.Grid.GridCellClickEventHandler(TableControl_Ce
llDoubleClick);
```

**[VB.NET]**

```
AddHandler Me.gridGroupingControl1.TableControlCellDoubleClick,
AddressOf TableControl_CellDoubleClick
```

The following **GridCellClickEventArgs** properties provide information specific to this event.

Properties	Description
IsOverImage	Indicates if the mouse was over the image in a static cell when the mouse was released.
MouseEventArgs	The MouseEventArgs originating this event.
ColIndex	Specifies the column index of the cell.

RowIndex	Specifies the row index of the cell.
Cancel	Specifies a value to indicate if this event should be canceled.

[C#]

```
private void TableControl_CellDoubleClick(object sender,  
GridCellEventArgs e)  
{  
    Console.WriteLine("Cell DoubleClicked");  
}
```

[VB .NET]

```
Private Sub TableControl_CellDoubleClick(ByVal sender As Object, ByVal  
e As GridCellEventArgs)  
    Console.WriteLine("Cell DoubleClicked")  
End Sub
```

#### 4.3.4.14.3 Current Cell Events

The Current Cell events are as follows:

**CurrentCellActivated**-Occurs after the grid activates the specified cell as current cell.

[C#]

```
this.groupingEngine.TableControl.CurrentCellActivated+=new  
EventHandler(TableControl_CurrentCellActivated);
```

[VB .NET]

```
AddHandler groupingEngine.TableControl.CurrentCellActivated, AddressOf  
TableControl_CurrentCellActivated
```

**CurrentCellChanged**-Occurs when the user changes contents of the current cell.

[C#]

```
this.groupingEngine.TableControl.CurrentCellChanged+=new  
EventHandler(TableControl_CurrentCellChanged);
```

**[VB .NET]**

```
AddHandler groupingEngine.TableControl.CurrentCellChanged, AddressOf  
TableControl_CurrentCellChanged
```

**CurrentCellControlGotFocus**-Occurs when the current cell has switched to in-place editing and the control associated with the current cell has received the focus.

**[C#]**

```
this.groupingEngine.TableControl.CurrentCellControlGotFocus+=new  
ControlEventHandler(TableControl_CurrentCellControlGotFocus);
```

**[VB .NET]**

```
AddHandler groupingEngine.TableControl.CurrentCellControlGotFocus,  
AddressOf TableControl_CurrentCellControlGotFocus
```

The event handler receives an argument of type ControlEventArgs containing data related to this event.

The following ControlEventArgs properties provide information specific to this event.

- **Control**-Gets the control object used by this event.

**CurrentCellControlLostFocus**-Occurs when the current cell is in in-place editing mode and the control associated with the current cell has lost the focus.

**[C#]**

```
this.groupingEngine.TableControl.CurrentCellControlLostFocus+=new  
ControlEventHandler(TableControl_CurrentCellControlLostFocus);
```

**[VB .NET]**

```
AddHandler groupingEngine.TableControl.CurrentCellControlLostFocus,  
AddressOf TableControl_CurrentCellControlLostFocus
```

The event handler receives an argument of type ControlEventArgs containing data related to this event.

The following ControlEventArgs properties provide information specific to this event.

- **Control**-Gets the control object used by this event.

**CurrentCellDeactivated**-Occurs after the grid deactivates current cell.

**[C#]**

```
this.groupingEngine.TableControl.CurrentCellDeactivated+=new  
Syncfusion.Windows.Forms.Grid.GridCurrentCellDeactivatedEventHandler(Ta  
bleControl_CurrentCellDeactivated);
```

**[VB .NET]**

```
AddHandler groupingEngine.TableControl.CurrentCellDeactivated,  
AddressOf TableControl_CurrentCellDeactivated
```

The event handler receives an argument of type GridCurrentCellDeactivatedEventArgs containing data related to this event.

The following GridCurrentCellDeactivatedEventArgs properties provide information specific to this event.

- **RowIndex**-Gets the row index.
- **ColIndex**-Gets the column index.

**CurrentCellStartEditing**-Occurs before the current cell switches into editing mode.

**[C#]**

```
this.groupingEngine.TableControl.CurrentCellStartEditing+=new  
CancelEventHandler(TableControl_CurrentCellStartEditing);
```

**[VB .NET]**

```
AddHandler groupingEngine.TableControl.CurrentCellStartEditing,  
AddressOf TableControl_CurrentCellStartEditing
```

The event handler receives an argument of type CancelEventArgs containing data related to this event.

The following CancelEventArgs properties provide information specific to this event.

- **Cancel**-Gets or sets a value indicating whether the event should be canceled.

**CurrentCellValidated**-Occurs when the grid has successfully validated the contents of the active current cell.

**[C#]**

```
this.groupingEngine.TableControl.CurrentCellValidated+=new  
EventHandler(TableControl_CurrentCellValidated);
```

**[VB .NET]**

```
AddHandler groupingEngine.TableControl.CurrentCellValidated, AddressOf  
TableControl_CurrentCellValidated
```

#### 4.3.4.14.4 Control Events

Following are the control events:

**Click**-Occurs when control is clicked.

**[C#]**

```
this.groupingEngine.TableControl.Click+=new  
EventHandler(TableControl_Click);
```

**[VB .NET]**

```
AddHandler groupingEngine.TableControl.Click, AddressOf  
TableControl_Click
```

**DoubleClick**-Occurs when control is double clicked.

**[C#]**

```
this.groupingEngine.TableControl.DoubleClick+=new  
EventHandler(TableControl_DoubleClick);
```

**[VB .NET]**

```
AddHandler groupingEngine.TableControl.DoubleClick, AddressOf  
TableControl_DoubleClick
```

**DragDrop**-Occurs when a drag and drop operation is completed.

**[C#]**

```
this.groupingEngine.TableControl.DragDrop+=new  
DragEventHandler(TableControl_DragDrop);
```

**[VB .NET]**

```
AddHandler groupingEngine.TableControl.DragDrop, AddressOf  
TableControl_DragDrop
```

**GotFocus**-Occurs when the control receives focus.

**[C#]**

```
this.groupingEngine.TableControl.GotFocus+=new  
EventHandler(TableControl_GotFocus);
```

**[VB .NET]**

```
AddHandler groupingEngine.TableControl.GotFocus, AddressOf  
TableControl_GotFocus
```

**GridControlMouseDown**-Occurs before a MouseDown is raised and allows you to cancel the mouse event.

**[C#]**

```
this.groupingEngine.TableControl.GridControlMouseDown+=new  
Syncfusion.Windows.Forms.CancelMouseEventHandler(TableControl_GridContr  
olMouseDown);
```

**[VB .NET]**

```
AddHandler groupingEngine.TableControl.GridControlMouseDown, AddressOf  
TableControl_GridControlMouseDown
```

The event handler receives an argument of type **CancelMouseEventArgs** containing data related to this event.

The following CancelMouseEventArgs properties provide information specific to this event.

- **Cancel**-Gets or sets a value indicating whether the event should be canceled.

**GridControlMouseUp**-Occurs before a MouseUp is raised and allows you to cancel the mouse event.

**[C#]**

```
this.groupingEngine.TableControl.GridControlMouseUp+=new  
Syncfusion.Windows.Forms.CancelEventHandler(TableControl_GridContr  
olMouseUp);
```

**[VB .NET]**

```
AddHandler groupingEngine.TableControl.GridControlMouseUp, AddressOf  
TableControl_GridControlMouseUp
```

The event handler receives an argument of type **CancelMouseEventArgs** containing data related to this event.

The following CancelMouseEventArgs properties provide information specific to this event.

- **Cancel**-Gets or sets a value indicating whether the event should be canceled.

#### 4.3.4.14.5 Mouse Events

Following are the mouse events:

**MouseDown**-Occurs when the mouse pointer is over the control and a mouse button is pressed.

**[C#]**

```
this.groupingEngine.TableControl.MouseDown+=new  
MouseEventHandler(TableControl_MouseDown);
```

**[VB .NET]**

```
AddHandler groupingEngine.TableControl.MouseDown, AddressOf  
TableControl_MouseDown
```

The event handler receives an argument of type MouseEventArgs containing data related to this event. The following MouseEventArgs properties provide information specific to this event.

- **Button**-Indicates which mouse button was pressed.
- **Clicks**-The number of times a mouse button was pressed.
- **X**-The x-coordinate of a mouse click, in pixels.
- **Y**-The y-coordinate of a mouse click, in pixels.
- **Point**-Gets the location of the mouse during the generating mouse event.

**MouseLeave**-Occurs when mouse pointer leaves the control.

**[C#]**

```
this.groupingEngine.TableControl.MouseLeave+=new MouseLeave  
EventHandler(TableControl_MouseLeave);
```

**[VB .NET]**

```
AddHandler groupingEngine.TableControl.MouseLeave, AddressOf  
TableControl_MouseLeave
```

**MouseUp**-Occurs when the mouse pointer is over the control and a mouse button is released.

**[C#]**

```
this.groupingEngine.TableControl.MouseUp+=new  
MouseEventHandler(TableControl_MouseUp);
```

**[VB .NET]**

```
AddHandler groupingEngine.TableControl.MouseUp, AddressOf  
TableControl_MouseUp
```

The event handler receives an argument of type MouseEventArgs containing data related to this event. The following MouseEventArgs properties provide information specific to this event.

- **Button**-Indicates which mouse button was pressed.

- **Clicks**-The number of times a mouse button was pressed.
- **X**-The x-coordinate of a mouse click, in pixels.
- **Y**-The y-coordinate of a mouse click, in pixels.
- **Point**-Gets the location of the mouse during the generating mouse event.

In this section, you will learn about the following events.

#### *4.3.4.14.5.1 MouseDown Event*

It occurs when the mouse pointer is over the control and a mouse button is pressed. It receives an argument of type **GridTableControlMouseEventArgs** that provides data related to this event. This event can be invoked as follows.

##### **[C#]**

```
this.gridGroupingControl1.TableControlMouseDown+=new  
GridTableControlMouseEventArgs(TableControl_MouseDown);
```

##### **[VB .NET]**

```
Private Me.gridGroupingControl1.TableControlMouseDown+= New  
GridTableControlMouseEventArgs(TableControl_MouseDown)
```

The following **GridTableControlMouseEventArgs** properties provide information specific to this event.

<b>Properties</b>	<b>Description</b>
Inner	Holds the event data of the underlying original event.
TableControl	The GridTableControl that initiated the original event.

##### **[C#]**

```
private void Table Control_MouseDown(object sender,  
GridTableControlMouseEventArgs e)  
{
```

```
Console.WriteLine("MouseDown: No. of  
Clicks"+e.Inner.Clicks.ToString());  
}
```

**[VB.NET]**

```
Private Sub TableControl_MouseDown(ByVal sender As Object, ByVal e As  
GridTableControlMouseEventArgs)  
    Console.WriteLine("MouseDown: No. of  
Clicks"+e.Inner.Clicks.ToString())  
End Sub
```

#### 4.3.4.14.5.2 *MouseLeave Event*

It occurs when the mouse pointer leaves the control. It receives an argument of type **EventArgs** that provides data, related to this event. This event can be invoked as follows.

**[C#]**

```
this.gridGroupingControll.TableControl.MouseLeave+=new  
MouseEventHandler(TableControl_MouseDown);
```

**[VB.NET]**

```
Private Me.gridGroupingControll.TableControl.MouseLeave+= New  
MouseEventHandler(TableControl_MouseLeave)
```

#### Example

**[C#]**

```
private void TableControl_MouseLeave(object sender, EventArgs e)  
{  
    Console.WriteLine("MouseLeave");  
}
```

**[VB.NET]**

```
Private Sub TableControl_MouseLeave(ByVal sender As Object, ByVal e As  
EventArgs)  
    Console.WriteLine("MouseLeave")  
End Sub
```

#### 4.3.4.14.5.3 MouseUp Event

It occurs when the mouse pointer leaves the control and a mouse button is pressed. It receives an argument of type **MouseEventArgs** that provides data related to this event. This event can be invoked as follows.

[C#]

```
this.gridGroupingControl1.TableControlMouseUp+=new
MouseEventHandler(TableControl_MouseUp);
```

[VB .NET]

```
Private Me.gridGroupingControl1.TableControlMouseUp+= New
MouseEventHandler(TableControl_MouseUp)
```

The following **MouseEventArgs** properties provide information specific to this event.

Properties	Description
Button	Gets which mouse button(left or right) is pressed.
Clicks	Gets the number of times the mouse button was pressed and released.
Delta	Gets a signed count of the number of detent's the mouse wheel has rotated. A detent is one notch of the mouse wheel.
Location	Gets the mouse location during generating this event.
X	Gets X co-ordinate of the mouse during generating this event.
Y	Gets Y co-ordinate of the mouse during generating this event.

#### Example

**[C#]**

```
private void TableControl_MouseUp(object sender, MouseEventArgs e)
{
    Console.WriteLine("MouseUp: No. of Clicks"+e.Clicks.ToString());
}
```

**[VB .NET]**

```
Private Sub TableControl_MouseUp(ByVal sender As Object, ByVal e As MouseEventArgs)
    Console.WriteLine("MouseUp: No. of Clicks"+e.Clicks.ToString())
End Sub
```

#### 4.3.4.14.6 Key Events

Following are the key events:

**KeyDown**-Occurs when a key is pressed when control has focus.

**[C#]**

```
this.groupingEngine.TableControl.KeyDown+=new
KeyEventHandler(TableControl_KeyDown);
```

**[VB .NET]**

```
AddHandler groupingEngine.TableControl.KeyDown, AddressOf
TableControl_KeyDown
```

The event handler receives an argument of type `KeyEventArgs` containing data related to this event.

The following `KeyEventArgs` properties provide information specific to this event.

**KeyData**-Gets the key data for a `System.Windows.Forms.Control.KeyDown` or `System.Windows.Forms.Control.KeyUp` event.

- **Alt**-Gets a value indicating whether the ALT key was pressed.
- **Control**-Gets a value indicating whether the CTRL key was pressed.
- **Handled**-Gets or sets a value indicating whether the event was handled.

- **KeyCode**-Gets the keyboard code for a System.Windows.Forms.Control.KeyDown or System.Windows.Forms.Control.KeyUp.
- **KeyValue**-Gets the keyboard value for a System.Windows.Forms.Control.KeyDown or System.Windows.Forms.Control.KeyUp event.
- **Modifiers**-Gets the modifier flags for a System.Windows.Forms.Control.KeyDown or System.Windows.Forms.Control.KeyUp.
- **Shift**-Gets a value indicating whether the SHIFT key was pressed.
- **SuppressKeyPress**-Gets or sets a value indicating whether the key event should be passed on to the underlying control.

**KeyPress**:Occurs when a key is pressed when control has focus.

**[C#]**

```
this.groupingEngine.TableControl.KeyPress+=new  
KeyPressEventHandler(TableControl_KeyPress);
```

**[VB .NET]**

```
AddHandler groupingEngine.TableControl.KeyPress, AddressOf  
TableControl_KeyPress
```

The event handler receives an argument of type KeyPressEventArgs containing data related to this event. The following KeyPressEventArgs properties provide information specific to this event.

- **KeyChar**-The ASCII character corresponding to the key the user pressed.
- **Handled**-Gets or sets a value indicating whether the System.Windows.Forms.Control.KeyPress

**KeyUp**:Occurs when a key is released when control has focus.

**[C#]**

```
this.groupingEngine.TableControl.KeyUp+=new  
KeyEventHandler(TableControl_KeyUp);
```

**[VB .NET]**

```
AddHandler groupingEngine.TableControl.KeyUp, AddressOf  
TableControl_KeyUp
```

The event handler receives an argument of type KeyEventArgs containing data related to this event. The following KeyEventArgs properties provide information specific to this event.

- **KeyData**-Gets the key data for a System.Windows.Forms.Control.KeyDown or System.Windows.Forms.Control.KeyUp event.
- **Alt**-Gets a value indicating whether the ALT key was pressed.
- **Control**-Gets a value indicating whether the CTRL key was pressed.
- **Handled**-Gets or sets a value indicating whether the event was handled.
- **KeyCode**-Gets the keyboard code for a System.Windows.Forms.Control.KeyDown or System.Windows.Forms.Control.KeyUp
- **KeyValue**-Gets the keyboard value for a System.Windows.Forms.Control.KeyDown or System.Windows.Forms.Control.KeyUp event.
- **Modifiers**-Gets the modifier flags for a System.Windows.Forms.Control.KeyDown or System.Windows.Forms.Control.KeyUp event.
- **Shift**-Gets a value indicating whether the SHIFT key was pressed.
- **SuppressKeyPress**-Gets or sets a value indicating whether the key event should be passed on to the underlying control.

#### 4.3.4.14.7 Selection Events for Filter Bar

Two Selection events for filter bar, namely *FilterBarSelectedItemChanging* and *FilterBarSelectedItemChanged* are implemented for Essential Grid. These events enable you to cancel or customize the filter value. This support is not available in MS-DataGridView.

#### Use Case Scenarios

The *FilterBarSelectedItemChanging* event enables you to customize or cancel filter value for specific column. *FilterBarSelectedItemChanged* event enable you to retrieve the selected column information, selected index and selected text.

#### Events Table

Table 13: Event Table

Event	Description	Arguments	Type	Reference links
<i>FilterBarSelect edItemChangi ng</i>	This event will be triggered while filtering a column.  This can be canceled.  We can modify the selected index or selected text as required.	public FilterBarSelect edItemChangi ngEventArgs( GridColumnD escriptor column, int selectedIndex, string selectedText)	Event	NA

<i>FilterBarSelect edItemChange d</i>	This event will be triggered after a column is filtered.  We can retrieve the selected column, selected index and selected text with this event.	public FilterBarSelect edItemChange dEventArgs(G ridColumnDes criptor column, int selectedIndex, string selectedText)	Event	NA
---	--	--	-------	----

**Triggering Selection Events**

Refer to the following code, to trigger the Selection events:

```
//Events hooked in the form load...

this.gridGroupingControl1.FilterBarSelectedIndexChanged += new
FilterBarSelectedIndexChangedEventHandler(gridGroupingControl1_FilterB
arSelectedIndexChanged);

this.gridGroupingControl1.FilterBarSelectedIndexChanged += new
FilterBarSelectedIndexChangedEventHandler(gridGroupingControl1_Filter
BarSelectedIndexChanged);

void gridGroupingControl1_FilterBarSelectedIndexChanged(object
sender, FilterBarSelectedIndexChangedEventArgs e)
{
    //You can cancel the filtering like this.
    if (e.Column.Name == "Department")
    {
        e.Cancel = true;
    }
}

void gridGroupingControl1_FilterBarSelectedIndexChanged(object
sender, FilterBarSelectedIndexChangedEventArgs e)
{
    //you can retreive the selected column, selected index and selected
}
```

```
text.  
        MessageBox.Show(e.SelectedIndex.ToString());  
    }  
}
```

```
[VB]  
'Events hooked in the form load...  
  
    Private  
Me.gridGroupingControl1.FilterBarSelectedIndexChanged += New  
FilterBarSelectedIndexChangedEventHandler(AddressOf  
gridGroupingControl1_FilterBarSelectedIndexChanged)  
  
    Private  
Me.gridGroupingControl1.FilterBarSelectedItemChanging += New  
FilterBarSelectedItemChangingEventHandler(AddressOf  
gridGroupingControl1_FilterBarSelectedItemChanging)  
  
Private Sub gridGroupingControl1_FilterBarSelectedItemChanging(ByVal  
sender As Object, ByVal e As FilterBarSelectedItemChangingEventArgs)  
  
'You can cancel the filtering like this.  
    If e.Column.Name = "Department" Then  
        e.Cancel = True  
    End If  
End Sub  
  
    Private Sub  
gridGroupingControl1_FilterBarSelectedIndexChanged(ByVal sender As  
Object, ByVal e As FilterBarSelectedIndexChangedEventArgs)  
'you can retreive the selected column, selected index and selected  
text.  
    MessageBox.Show(e.SelectedIndex.ToString())  
End Sub
```

#### 4.3.4.15 Delete Collection of Records in **GridGroupingControl**

Essential **GridGroupingControl** now supports deletion of collection of Records from the **GridGroupingControl** instead of deleting records one by one.

**GridGroupingControl** supports three methods of deleting records.

- **DeleteAll** - Deletes all the records
- **Delete All[Selected]** - Delete selected records
- **Delete Selected Records** - Deletes Specified records

For these methods **DeleteAll** have been implemented to support delete all the records, **Delete All[Selected]** deletes selected records, and **Delete Selected Records** delete specified records from the **GridGroupingControl**.

### **Deleting All**

The following code illustrates deleting all the records.

[C#]

```
//This will delete all the records from the grid table.  
this.gridGroupingControl1.Table.Records.DeleteAll();
```

When the code runs, Deleting all records is bound to **Delete All** button.

### **Deleting Selected Records**

The following code illustrates deleting records manually selected.

[C#]

```
// this will delete selected records from the grid table.  
this.gridGroupingControl1.Table.SelectedRecords.DeleteAll();
```

When the code runs, Deleting manually selected record is bound to **Delete All[Selected]** button

### **Deleting Specified Records**

The following code illustrates deleting Specified records.

 **Note:** Parameter – Specify the collection of records need to be deleted.

[C#]

```
//Delete the specified records from the table
```

```
this.gridGroupingControl1.Table.Records.DeleteRecords(rec);
```

When the code runs, Deleting specified records are bound to **Delete Selected Records** button.

The screenshot shows a Windows Form with a grid control and a context menu. The grid has columns for CustomerID, CompanyName, and ContactName. The context menu on the right contains three items: 'Delete All', 'Delete All[Selected]', and 'Delete Selected Records'.

CustomerID	CompanyName	ContactName
ALFKI	Alfreds Futterkiste	Maria Anders
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo
ANTON	Antonio Moreno Taquería	Antonio Moreno
AROUT	Around the Horn	Thomas Hardy
BERGS	Berglunds snabbköp	Christina Berglund
BLAUS	Blauer See Delikatessen	Hanna Moos
BLONP	Blondesddsl père et fils	Frédérique Citeaux
BOLID	Bólido Comidas preparadas	Martín Sommer
BONAP	Bon app'	Laurence Lebihan
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln
BSBEV	B's Beverages	Victoria Ashworth
CACTU	Cactus Comidas para llevar	Patricia Simpson
CENTC	Centro comercial Moctezuma	Francisco Chang
CHOPS	Chop-suey Chinese	Yang Wang
COMMI	Comércio Mineiro	Pedro Afonso
CONSH	Consolidated Holdings	Elizabeth Brown
DRACD	Drachenblut Delikatessen	Sven Ottlieb
DUMON	Du monde entier	Janine Labrune
EASTC	Eastern Connection	Ann Devon
ERNSH	Ernst Handel	Roland Mendel
FAMIA	Familia Arquibaldo	Aria Cruz

Figure 393: Delete Collection of Records in GridGroupingControl

#### 4.3.4.16 Select Collection of Records In the GridGroupingControl

Essential **GridGroupingControl** now supports two methods for selecting records in grid table.

- **SelectAll**
- **Select Specified Record**

##### Selecting All

The following code illustrates how to select all record in grid table.

```
//This will select all the records from the grid table
this.gridGroupingControl1.Table.Records.SelectAll();
```

When the code runs, selecting all record is bound to **Select All** button.

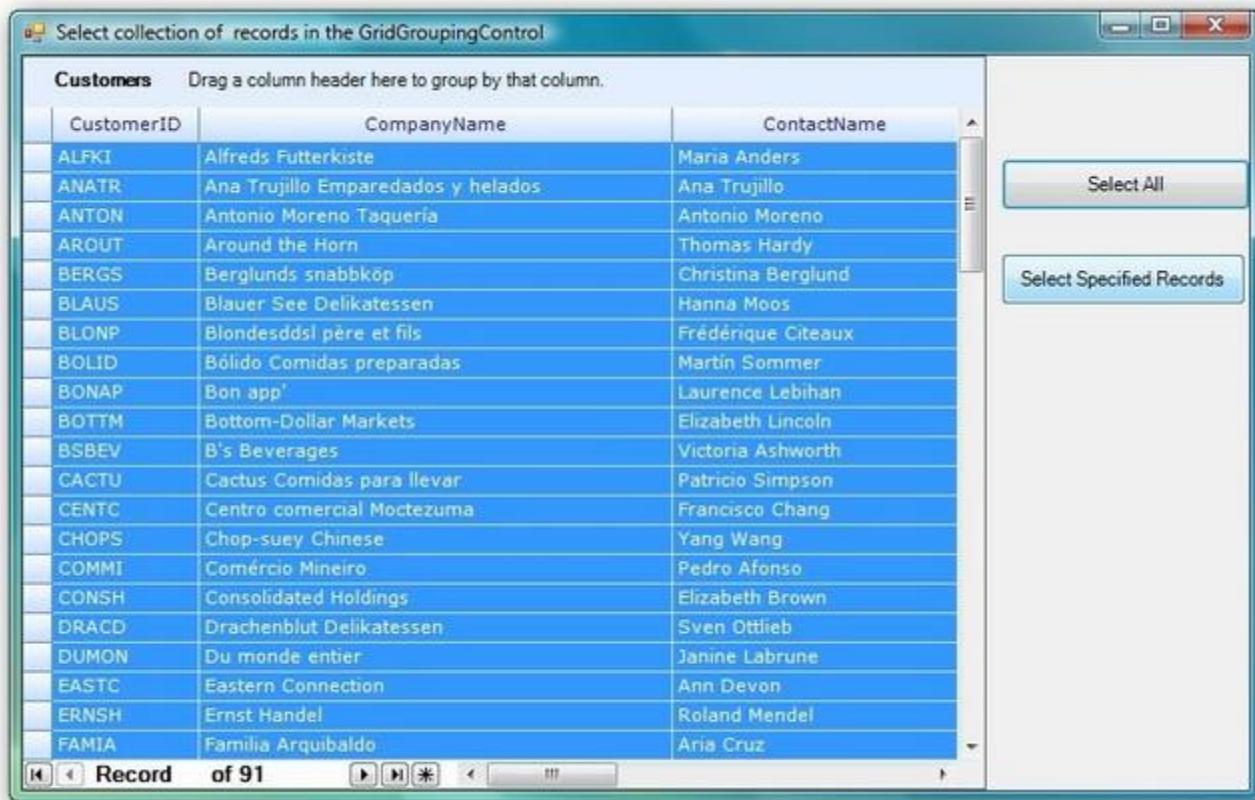


Figure 394: SelectAll

### Selecting Specified Records

The following code illustrates how to select specified records.



**Note: Method Name: AddRange**

**Parameter: Specify the Record collection to be selected**

```
//this will select the specified records in the grid table
this.gridGroupingControl1.Table.SelectedRecords.AddRange(rec);
```

When the code runs, selecting specified is bound to **Select Specified Records** button.

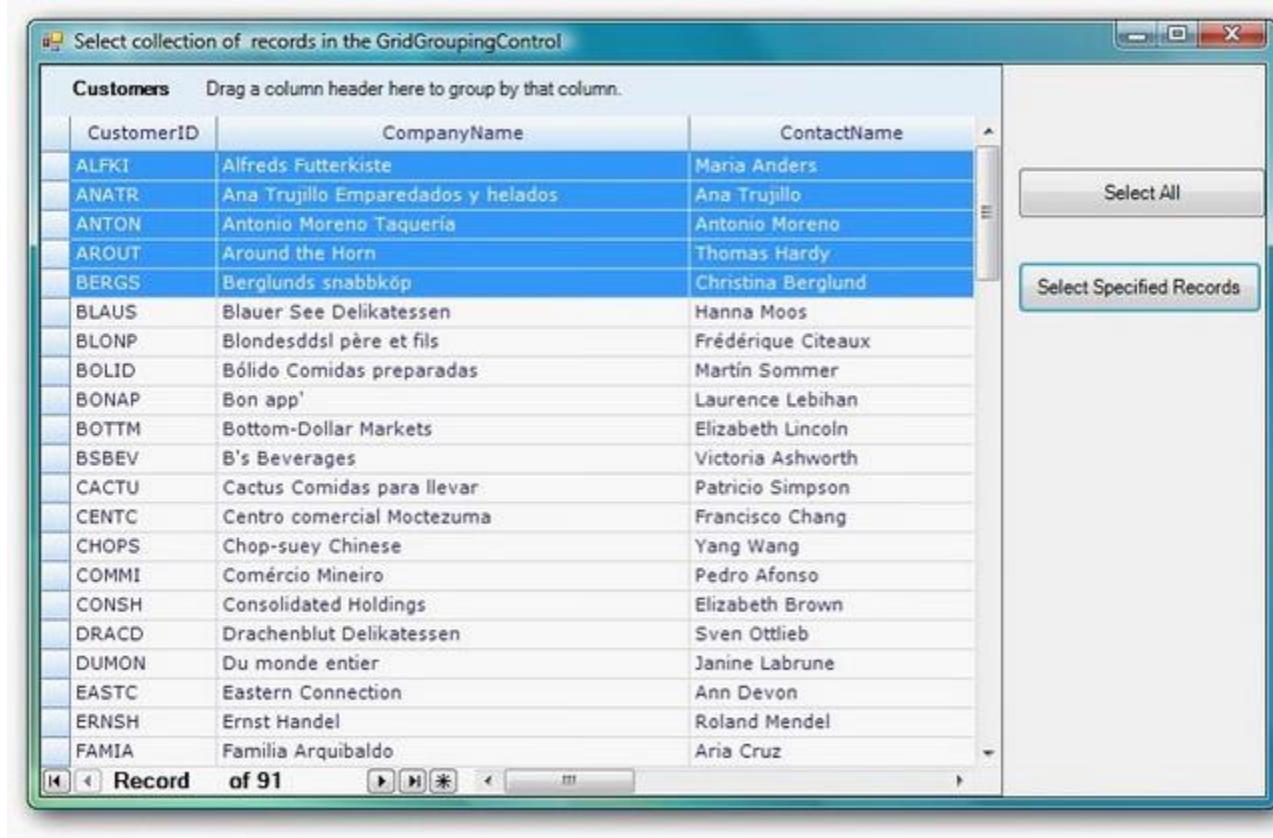


Figure 395: Illustrating AddRange

#### 4.3.4.17 Export Summary as Caption

This feature enables you to export the summary as caption, while exporting GridGroupingControl to Excel.

#### Properties

Table 14: Property Table

Property	Description	Type	Data Type	Reference links
ExportCaptionSummary	This property exports summary as caption.	GridGroupingControl	Boolean	NA

## Exporting Summary as Caption

You can export summary as caption using the *ExportCaptionSummary* property. The following code illustrates this:

[C#]

```
converter.ExportCaptionSummary = true;
```

	ParentID	ParentName	ParentCity	ParentState	Fees	
			Discount		134297.00	
			Department:		2563	21296
			FOR		2251.00	29479.00
			14	PREETHI	ECE	II
					2251	29479
			Discount		2251	29479
			Department:		4049.00	19418.00
			MARINE	RAM	MARINE	IV
					4049	19418
			Discount		4049	19418
			Department:		1682.00	14424.00
			TEXTILE	RISHI	TEXTILE	III
					1682	14424
			Discount		1682	14424

Figure 396: Summary as Caption

### 4.3.4.17.1 Searching for text in multiple columns

This feature can be used for performing searches either in specified columns or in all of the grid columns. Only the rows containing the search text will be filtered and displayed.

Searches can be performed in two modes:

1. Search for text in all columns.
2. Search for text in particular columns.

#### Method Table

Method	Prototypes	Description
Search()	<code>this.gridGroupingControl1.TableDescriptor.Search("SearchText", List&lt;GridColumnDescriptor&gt;);</code>	It is called for performing the searching operation in the columns.

	<code>this.gridGroupingControl1.TableDescriptor.Search("SearchText");</code>	
--	--	--

#### *4.3.4.17.1.1 Searching for text in all columns*

Here, text is provided as a parameter in the Search method in the TableDescriptor. It searches for the text in all the columns and displays the rows that contain the word.

The following code samples illustrate how to perform the search operation:

[C#]

```
this.gridGroupingControl1.TableDescriptor.Search("Duke");
```

[VB]

```
Me.gridGroupingControl1.TableDescriptor.Search("Duke");
```

#### *4.3.4.17.1.2 Searching for text in particular columns*

We can perform this operation by passing the columns that need to be searched into the Search() method.

In this case, we need to create a list that will hold the columns that need to be searched for that particular text. The search text and the list of columns that will be searched are passed as parameters for performing the operation. The following code illustrates this:

[C#]

```
List<GridColumnDescriptor> list= new List<GridColumnDescriptor>();  
list.Add(this.gridGroupingControl1.TableDescriptor.Columns[1]);  
list.Add(this.gridGroupingControl1.TableDescriptor.Columns[3]);  
  
this.gridGroupingControl1.TableDescriptor.Search("Duke", list);
```

[VB]

```
Dim list As New List(Of GridColumnDescriptor)()  
list.Add(Me.gridGroupingControl1.TableDescriptor.Columns(1))  
list.Add(Me.gridGroupingControl1.TableDescriptor.Columns(3))
```

```
Me.gridGroupingControl1.TableDescriptor.Search("Duke", list)
```

## 4.4 Grid List Control

The Grid List control is a System.Windows.Forms ListControl-derived object. It uses an embedded **GridControl** object to hold data. Using this GridListControl.Grid member, you can easily control the appearance of your list control through the GridControl object.

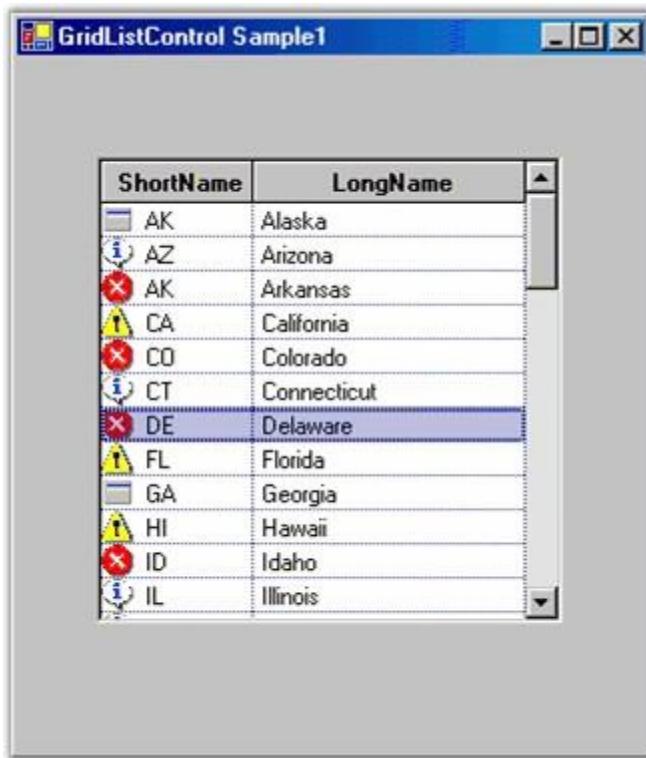


Figure 397: Grid List control holding United States Name Objects

### 4.4.1 Creating Grid List Control

This section will provide the step-by-step procedure to create a Grid List control through designer and through programmatical approach in a .NET application.

#### 4.4.1.1 Through Designer

With the designer, all you have to do is drag the control onto the form, size it and then set the desired properties, assuming you have a data source ready. If you do not have a data source ready, then create one by using the steps listed below.

##### To Create a Data Source

1. Drag an **SqlDataAdapter** from the **Data** tab of the Toolbox onto the form. Follow the steps in the wizard to select the database and SQL query used to generate the table.



Figure 398: Data Adapter Configuration Wizard

2. Click the **SqlAdapter** in the components tray, with the right mouse button and generate a dataset for this adapter, by just taking the defaults.

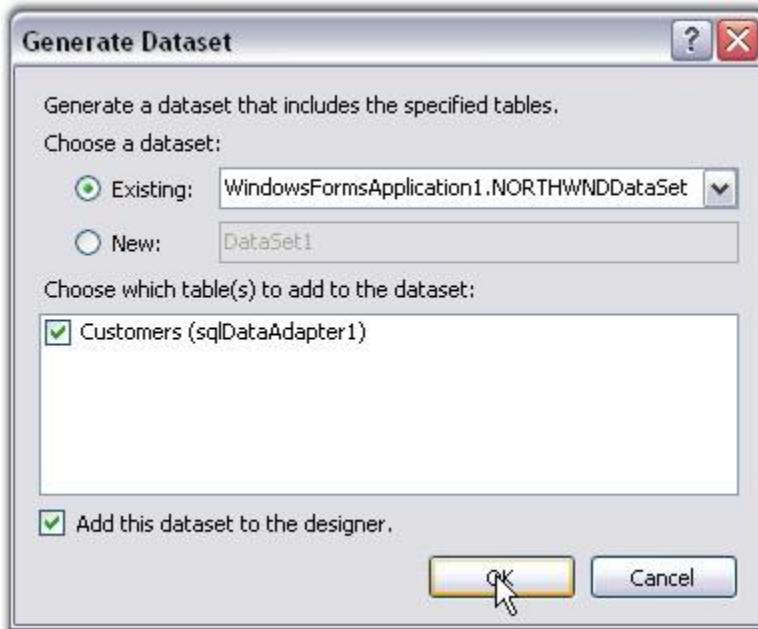


Figure 399: Generating Data Set

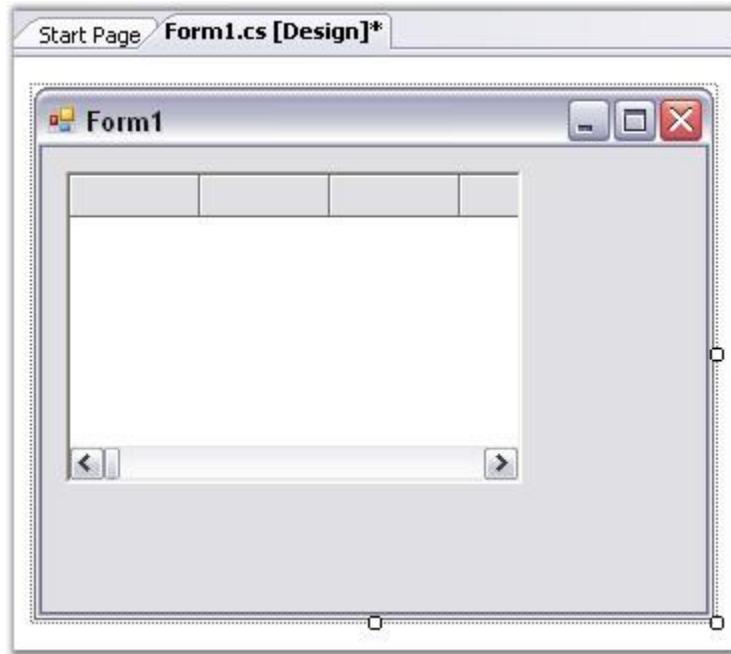
3. In the Form\_Load event handler, **Fill** method will be called automatically for this SqlDataAdapter, by passing the dataset that is generated in the previous step.

```
private void Form1_Load(object sender, EventArgs e)
{
    // TODO: This line of code loads data into the 'nORTHWNDDataset.Customers'
    this.customersTableAdapter.Fill(this.nORTHWNDDataset.Customers);

}
```

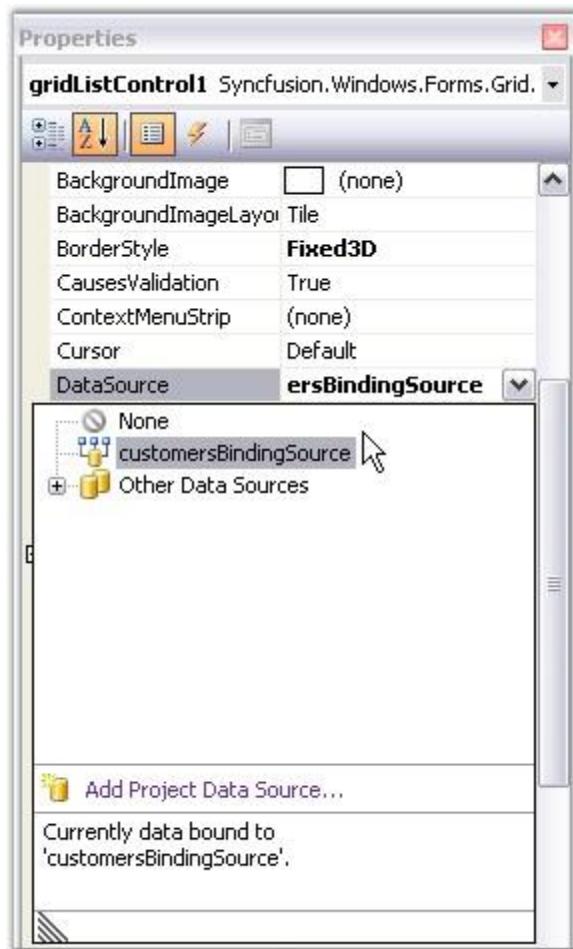
Figure 400: Fill method called automatically in the Form\_Load Event Handler

4. Drag a **GridListControl** object from your tool box and drop it onto the form.



*Figure 401: Grid List control dragged from the toolbox onto the Form*

5. Size and position it.
6. Go to Properties dialog of this Grid List control and set the **DataSource** property of this control to an appropriate object.



*Figure 402: Data Source set by using the DataSource Property*

7. Run the application. Following is the output.

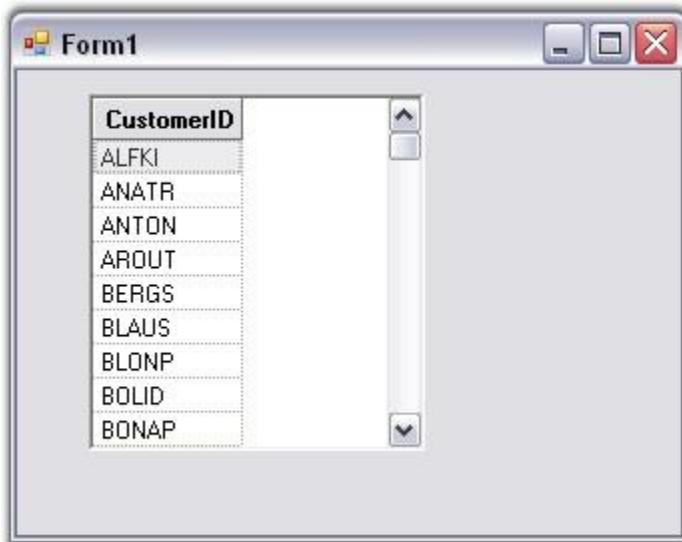


Figure 403: Grid List control created Through Code

This designer-created data source is now available for use as the data source member of the Grid List control. For a complete step-by-step tutorial on how to use the designer to create a data source, see the Grid Data Bound Grid tutorial.

#### 4.4.1.2 Through Code

The Grid List control sample that ships with Essential Grid does not use the designer. It creates an ArrayList of objects that serves as a data source for the Grid List control. Each state object has a **LongName**, **ShortName** and **ImageIndex** properties that can be displayed in the list control. Here are some code samples that illustrate the assignments of the major properties.

For the complete implementation details, refer to the sample in the below installation path.

**<Install Location>\Syncfusion\EssentialStudio\[Version Number]\Windows\Grid.Windows\Samples\2.0\Grid List Control**

```
[C#]

// Set to arraylist of states.
gridListBox1.DataSource = USStates;

// ImageList—the images displayed in the list.
gridListBox1.ImageList = imageList;
```

```
// Displays multiple columns.  
gridListBox1.MultiColumn = true;  
gridListBox1.ShowColumnHeader = true;  
gridListBox1.SelectionMode = SelectionMode.One;  
  
// Makes the last column wide enough to fill the client area.  
gridListBox1.FillLastColumn = true;
```

**[VB.NET]**

```
' Set to arraylist of states.  
gridListBox1.DataSource = USStates  
  
' ImageList-the images displayed in the list.  
gridListBox1.ImageList = ImageList  
  
' Displays multiple columns.  
gridListBox1.MultiColumn = True  
gridListBox1.ShowColumnHeader = True  
gridListBox1.SelectionMode = SelectionMode.One  
  
' Makes last column.  
gridListBox1.FillLastColumn = True
```

## 4.4.2 Concepts and Features

This section discusses the following concepts of Grid List control:

### 4.4.2.1 Data binding and Selection Modes

#### Data Binding

Data binding is used in Web pages that contain interactive components such as forms, calculators, tutorials, and games. Pages are displayed incrementally so that portions of a page can be used even before the entire page has finished downloading. Data binding helps in populating the Grid List control with large amounts of data. This can be achieved by using DataSource property which allows the system to acquire data from the Data Source Object (DSO).

The following code example illustrates data binding for Grid List control by using DataSource property.

**[C#]**

```
ArrayList array = new ArrayList();
array.Add(new MyClass(001, "John David"));
array.Add(new MyClass(002, "Tom"));
array.Add(new MyClass(003, "Bretney"));
array.Add(new MyClass(004, "Jessy"));
array.Add(new MyClass(005, "Bruch"));
array.Add(new MyClass(006, "Johny"));
this.gridlistControl1.DataSource = array;
```

**[VB .NET]**

```
Dim array As ArrayList = New ArrayList()
array.Add(New [MyClass](1, "John David"))
array.Add(New [MyClass](2, "Tom"))
array.Add(New [MyClass](3, "Bretney"))
array.Add(New [MyClass](4, "Jessy"))
array.Add(New [MyClass](5, "Bruch"))
array.Add(New [MyClass](6, "Johny"))
Me.gridlistControl1.DataSource = array
```

## Selection Modes

The selection behavior for Grid List control can be specified by using SelectionMode property. There are 3 types of selection behaviors:

- **One**—Allows the user to select only one item.
- **MultiSimple**—Allows the user to select multiple items.
- **MultiExtended**—Allows the user to select multiple items using SHIFT, CTRL, and arrow keys etc.

The following code example illustrates setting of various selection behaviors for the Grid List control.

**[C#]**

```
this.gridListControl1.SelectionMode = SelectionMode.One;
this.gridListControl1.SelectionMode = SelectionMode.MultiSimple;
this.gridListControl1.SelectionMode = SelectionMode.MultiExtended;
```

**[VB.NET]**

```
Me.gridListControl1.SelectionMode = SelectionMode.One
Me.gridListControl1.SelectionMode = SelectionMode.MultiSimple
Me.gridListControl1.SelectionMode = SelectionMode.MultiExtended
```

#### 4.4.2.2 ComboBoxBase Feature

The Grid List control can be coupled to a ComboBoxBase control by using the ListControl property of ComboBoxBase class. The ComboBoxBase is an advanced control provided by Syncfusion, which essentially separates the edit portion from the drop-down portion making. It displays Grid List control as a dropdown i.e. users can drop the Grid List control in the drop-down area to get a multi-column drop-down effect.

The following code example illustrates usage of Grid List control as a drop-down in Combo Box Base control.

**[C#]**

```
this.comboBoxBase1.ListControl = this.gridListControl1;
```

**[VB.NET]**

```
Me.comboBoxBase1.ListControl = Me.gridListControl1
```



Figure 404: Combo Box Base

#### 4.4.2.3 Customizing List control

The appearance Grid List control can be customized by customizing the background color, image, header background color etc. The following properties can be used for customization:

- **TransparentBackground**—This property can be used to set a transparent background for grid cells. If its value is set to true, no background color is displayed for the grid cells. If its value is set to false, the background is filled with the chosen color. The value is set to False by default. Refer BackColor property for setting the required background color.

The following code example illustrates setting of a transparent background for grid cells.

[C#]

```
this.gridListControl1.TransparentBackground = true;
```

[VB.NET]

```
Me.gridListControl1.TransparentBackground = True
```

	A	B	C	D	E
1	Row1:Col	Row1:Col	Row1:Col	Row1:Col	Row1:Col
2	Row2:Col	Row2:Col	Row2:Col	Row2:Col	Row2:Col
3	Row3:Col	Row3:Col	Row3:Col	Row3:Col	Row3:Col
4	Row4:Col	Row4:Col	Row4:Col	Row4:Col	Row4:Col
5	Row5:Col	Row5:Col	Row5:Col	Row5:Col	Row5:Col
6	Row6:Col	Row6:Col	Row6:Col	Row6:Col	Row6:Col
7	Row7:Col	Row7:Col	Row7:Col	Row7:Col	Row7:Col
8	Row8:Col	Row8:Col	Row8:Col	Row8:Col	Row8:Col
9	Row9:Col	Row9:Col	Row9:Col	Row9:Col	Row9:Col
10	Row10:C	Row10:C	Row10:C	Row10:C	Row10:C

Figure 405: Transparent Background set for Grid List Control

- **DisplayVertLines/DisplayHorzLines**—This property can be used to specify the display of vertical/horizontal lines on the grid. This property when set to true ensures display of vertical/horizontal grid lines.

The following code example illustrates the usage of the properties to display grid lines.

[C#]

```
this.gridListControl1.Properties.DisplayHorzLines = true;
this.gridListControl1.Properties.DisplayVertLines = true;
```

**[VB.NET]**

```
Me.gridListControl1.Properties.DisplayHorzLines = True
Me.gridListControl1.Properties.DisplayVertLines = True
```

	A	B	C	D	E
1	Row1:Col	Row1:Col	Row1:Col	Row1:Col	Row1:Col
2	Row2:Col	Row2:Col	Row2:Col	Row2:Col	Row2:Col
3	Row3:Col	Row3:Col	Row3:Col	Row3:Col	Row3:Col
4	Row4:Col	Row4:Col	Row4:Col	Row4:Col	Row4:Col
5	Row5:Col	Row5:Col	Row5:Col	Row5:Col	Row5:Col
6	Row6:Col	Row6:Col	Row6:Col	Row6:Col	Row6:Col
7	Row7:Col	Row7:Col	Row7:Col	Row7:Col	Row7:Col
8	Row8:Col	Row8:Col	Row8:Col	Row8:Col	Row8:Col
9	Row9:Col	Row9:Col	Row9:Col	Row9:Col	Row9:Col
10	Row10:C	Row10:C	Row10:C	Row10:C	Row10:C

Figure 406: Horizontal Lines displayed for Grid List Control

	A	B	C	D	E
1	Row1:Col	Row1:Col	Row1:Col	Row1:Col	Row1:Col
2	Row2:Col	Row2:Col	Row2:Col	Row2:Col	Row2:Col
3	Row3:Col	Row3:Col	Row3:Col	Row3:Col	Row3:Col
4	Row4:Col	Row4:Col	Row4:Col	Row4:Col	Row4:Col
5	Row5:Col	Row5:Col	Row5:Col	Row5:Col	Row5:Col
6	Row6:Col	Row6:Col	Row6:Col	Row6:Col	Row6:Col
7	Row7:Col	Row7:Col	Row7:Col	Row7:Col	Row7:Col
8	Row8:Col	Row8:Col	Row8:Col	Row8:Col	Row8:Col
9	Row9:Col	Row9:Col	Row9:Col	Row9:Col	Row9:Col
10	Row10:C	Row10:C	Row10:C	Row10:C	Row10:C

Figure 407: Vertical Lines displayed for Grid List Control

- **Buttons3D**-This property can be used to specify the appearance of row and column headers. This property when set to true, renders a three dimensional header providing the header a raised look.

The following code example illustrates the usage of the property to render a 3D header.

**[C#]**

```
this.gridListControl1.Properties.Buttons3D = true;
```

[VB.NET]

```
Me.gridListControl1.Properties.Buttons3D = True
```

	A	B	C	D	E
1	Row1:Col	Row1:Col	Row1:Col	Row1:Col	Row1:Col
2	Row2:Col	Row2:Col	Row2:Col	Row2:Col	Row2:Col
3	Row3:Col	Row3:Col	Row3:Col	Row3:Col	Row3:Col
4	Row4:Col	Row4:Col	Row4:Col	Row4:Col	Row4:Col
5	Row5:Col	Row5:Col	Row5:Col	Row5:Col	Row5:Col
6	Row6:Col	Row6:Col	Row6:Col	Row6:Col	Row6:Col
7	Row7:Col	Row7:Col	Row7:Col	Row7:Col	Row7:Col
8	Row8:Col	Row8:Col	Row8:Col	Row8:Col	Row8:Col
9	Row9:Col	Row9:Col	Row9:Col	Row9:Col	Row9:Col
10	Row10:C	Row10:C	Row10:C	Row10:C	Row10:C

Figure 408: Row and Column Headers with 3D Buttons Appearance

- **GridLineColor**-This property allows the user to specify a color for grid lines. Its value can be set to the required color.

The following code example illustrates the usage of this property to render blue grid lines.

[C#]

```
this.gridListControl1.Grid.Properties.GridLineColor = Color.Blue;
```

[VB.NET]

```
Me.gridListControl1.Grid.Properties.GridLineColor = Color.Blue
```

	A	B	C	D	E
1	Row1:Col	Row1:Col	Row1:Col	Row1:Col	Row1:Col
2	Row2:Col	Row2:Col	Row2:Col	Row2:Col	Row2:Col
3	Row3:Col	Row3:Col	Row3:Col	Row3:Col	Row3:Col
4	Row4:Col	Row4:Col	Row4:Col	Row4:Col	Row4:Col
5	Row5:Col	Row5:Col	Row5:Col	Row5:Col	Row5:Col
6	Row6:Col	Row6:Col	Row6:Col	Row6:Col	Row6:Col
7	Row7:Col	Row7:Col	Row7:Col	Row7:Col	Row7:Col
8	Row8:Col	Row8:Col	Row8:Col	Row8:Col	Row8:Col
9	Row9:Col	Row9:Col	Row9:Col	Row9:Col	Row9:Col
10	Row10:C	Row10:C	Row10:C	Row10:C	Row10:C

Figure 409: Grid Line Color set to "Blue"

- **BackColor**-This property allows the user to specify a background color for the Grid List control. It is mandatory to set the TransparentBackground to false to set the background color.

The following code example illustrates the usage of this property to render Beige background color.

**[C#]**

```
this.gridListControl1.BackColor = Color.Beige;
```

**[VB .NET]**

```
Me.gridListControl1.BackColor = Color.Beige
```

	A	B	C	D	E
1	Row1:Col	Row1:Col	Row1:Col	Row1:Col	Row1:Col
2	Row2:Col	Row2:Col	Row2:Col	Row2:Col	Row2:Col
3	Row3:Col	Row3:Col	Row3:Col	Row3:Col	Row3:Col
4	Row4:Col	Row4:Col	Row4:Col	Row4:Col	Row4:Col
5	Row5:Col	Row5:Col	Row5:Col	Row5:Col	Row5:Col
6	Row6:Col	Row6:Col	Row6:Col	Row6:Col	Row6:Col
7	Row7:Col	Row7:Col	Row7:Col	Row7:Col	Row7:Col
8	Row8:Col	Row8:Col	Row8:Col	Row8:Col	Row8:Col
9	Row9:Col	Row9:Col	Row9:Col	Row9:Col	Row9:Col
10	Row10:C	Row10:C	Row10:C	Row10:C	Row10:C

Figure 410: Grid List control with Back Color set to "Beige"

- **HeaderBackColor**-This property allows the user to specify the background color of headers.

The following code example illustrates the usage of this property to render a red background for the headers.

**[C#]**

```
this.gridListControl1.HeaderBackColor = Color.Red;
```

**[VB .NET]**

```
Me.gridListControl1.HeaderBackColor = Color.Red
```

- **HeaderTextColor**—This property allows the user to specify the header text color.

The following code example illustrates the usage of this property to render a blue header text color.

[C#]

```
this.gridListControl1.HeaderTextColor = Color.Blue;
```

[VB .NET]

```
Me.gridListControl1.HeaderTextColor = Color.Blue;
```

- **BackgroundImage**—This property allows the user to specify the background image used for the control.

The following code example illustrates the usage of this property to set the required image as background of the control.

[C#]

```
this.gridListControl1.BackgroundImage = Image.FromFile("Colud.jpg");
```

[VB .NET]

```
Me.gridListControl1.BackgroundImage = Image.FromFile("Colud.jpg")
```

	A	B	C	D	E
1	Row1:Col	Row1:Col	Row1:Col	Row1:Col	Row1:Col
2	Row2:Col	Row2:Col	Row2:Col	Row2:Col	Row2:Col
3	Row3:Col	Row3:Col	Row3:Col	Row3:Col	Row3:Col
4	Row4:Col	Row4:Col	Row4:Col	Row4:Col	Row4:Col
5	Row5:Col	Row5:Col	Row5:Col	Row5:Col	Row5:Col
6	Row6:Col	Row6:Col	Row6:Col	Row6:Col	Row6:Col
7	Row7:Col	Row7:Col	Row7:Col	Row7:Col	Row7:Col
8	Row8:Col	Row8:Col	Row8:Col	Row8:Col	Row8:Col
9	Row9:Col	Row9:Col	Row9:Col	Row9:Col	Row9:Col
10	Row10:C	Row10:C	Row10:C	Row10:C	Row10:C

Figure 411: Grid List control with Background Image Set

## 4.5 Grid Record Navigation Control

The Grid Record Navigation control will allow the user to move from row to row using a Microsoft Access-like navigation bar at the bottom of the grid.

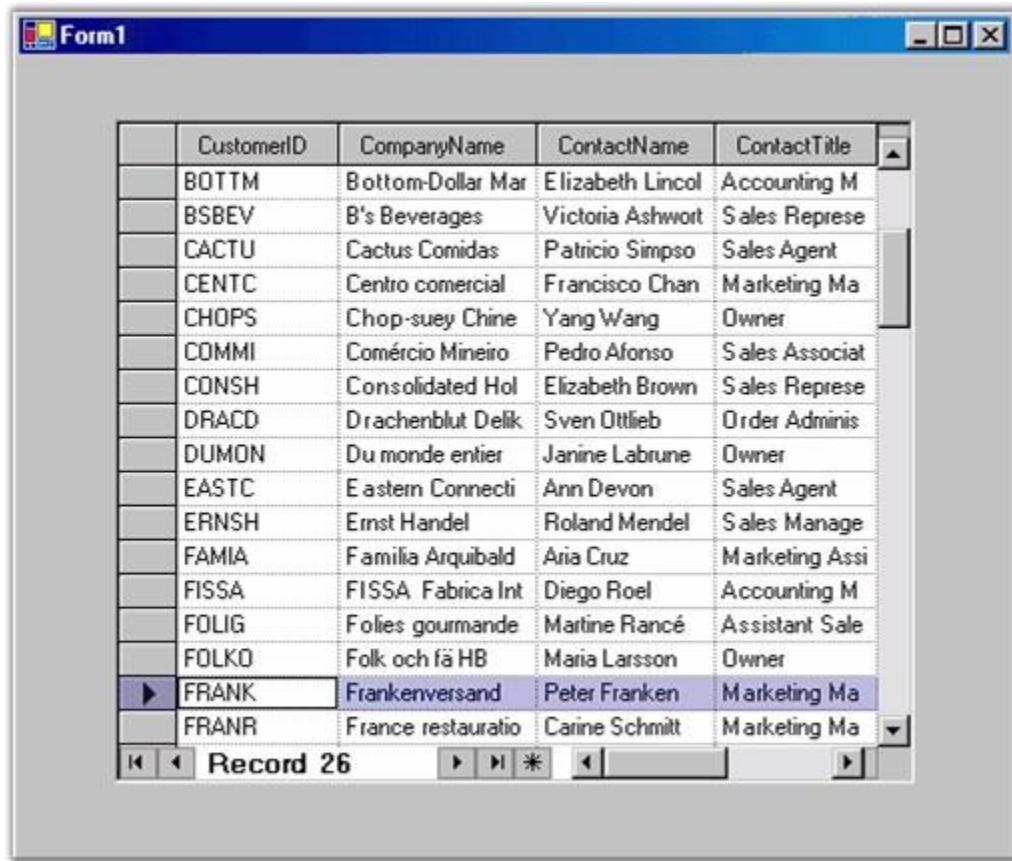


Figure 412: Grid Data Bound Grid Embedded in a Grid Record Navigation Control

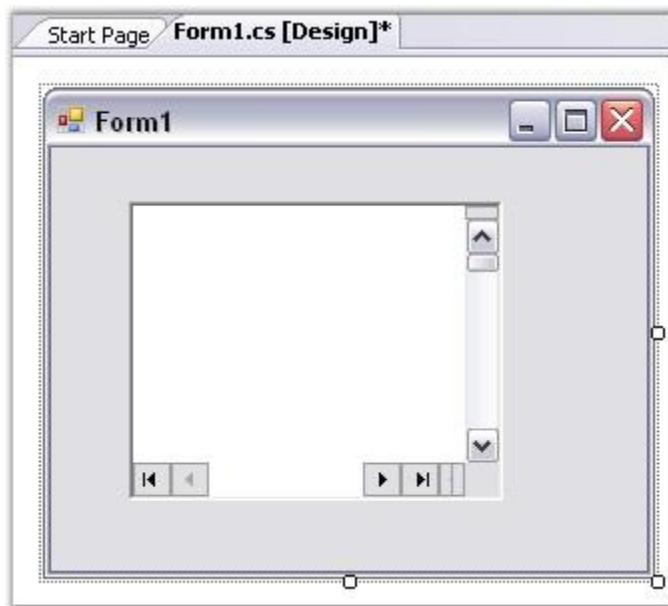
### 4.5.1 Creating Grid Record Navigation Control

This section will provide the step-by-step procedure to create a Grid Record Navigation control through designer and through programmatical approach in a .NET application.

#### 4.5.1.1 Through Designer

With the designer you can drag both the Grid Record Navigation control and a Grid control (either Grid control or Grid Data Bound Grid) from the toolbox onto your form. But, to get the proper initialization code generated by the designer, you must first drop and position the Grid Record Navigation control on your form and then drag the grid directly onto the Grid Record Navigation control. Do not drop the grid onto the form.

1. Drag a **GridRecordNavigationControl** object from your toolbox and drop it on the form.



*Figure 413: Grid Record Navigation control dragged from the Toolbox onto the Form*

2. Size and position it.
3. Drag a **GridDataBoundGrid** object from your toolbox and drop it on the Grid Record Navigation control (and not the form itself).

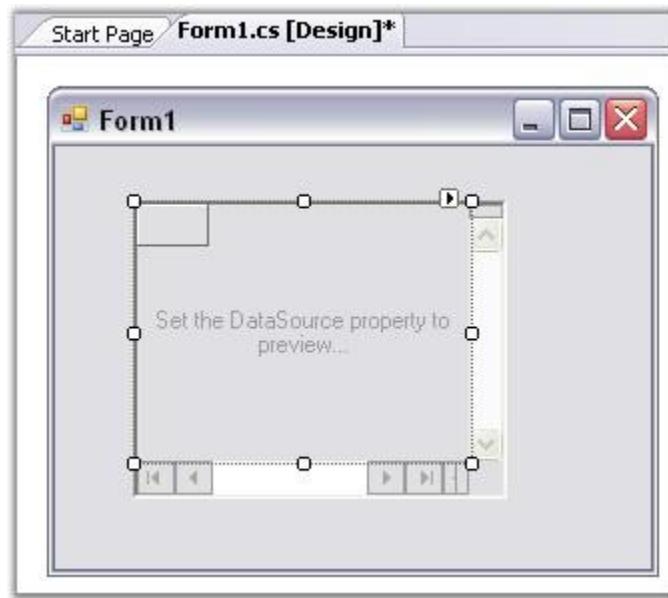


Figure 414: Grid Data Bound Grid dragged from the toolbox onto the Grid Record Navigation Control

4. Set the **GridDataBoundGrid.DataSource** property to an appropriate object.

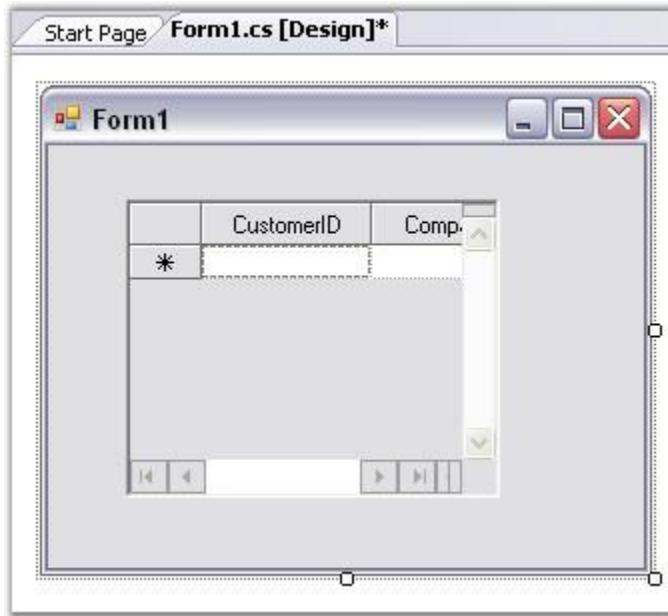


Figure 415: Data Source set for the Grid Data Bound Grid

5. Run the application. The following will be the output.



Figure 416: Grid Record Navigation Control created Through Designer

A Grid Record Navigation control is created.

#### 4.5.1.2 Through Code

Here is some minimal code that is necessary to create a Grid Record Navigation control.

[C#]

```
this.recordNavigationControl1 = new  
Syncfusion.Windows.Forms.Grid.GridRecordNavigationControl();  
this.recordNavigationControl1.Location = new System.Drawing.Point(32,  
48);  
this.recordNavigationControl1.MaxValue = "of 1000";  
this.recordNavigationControl1.MaxRecord = 1000;  
this.recordNavigationControl1.NavigationBarWidth = 237;  
this.recordNavigationControl1.Size = new System.Drawing.Size(520, 256);  
this.recordNavigationControl1.SplitBars =  
Syncfusion.Windows.Forms.DynamicSplitBars.Both;  
  
this.gridControl1 = new Syncfusion.Windows.Forms.Grid.GridControl();  
this.gridControl1.ColCount = 16;  
this.gridControl1.NumberedRowHeaders = false;  
this.gridControl1.RowCount = 1000;  
  
// Adds the grid to the Record Navigation control.
```

```
this.recordNavigationControl1.Controls.Add(this.gridControl1);

// Adds the Record Navigation control to the form.
this.Controls.Add(this.recordNavigationControl1);
```

**[VB.NET]**

```
Me.recordNavigationControl1 = New
Syncfusion.Windows.Forms.Grid.GridRecordNavigationControl()
Me.recordNavigationControl1.Location = New System.Drawing.Point(32, 48)
Me.recordNavigationControl1.MaxLabel = "of 1000"
Me.recordNavigationControl1.MaxRecord = 1000
Me.recordNavigationControl1.NavigationBarWidth = 237
Me.recordNavigationControl1.Size = New System.Drawing.Size(520, 256)
Me.recordNavigationControl1.SplitBars =
Syncfusion.Windows.Forms.DynamicSplitBars.Both

Me.gridControl1 = New Syncfusion.Windows.Forms.Grid.GridControl()
Me.gridControl1.ColCount = 16
Me.gridControl1.NumberedRowHeaders = False
Me.gridControl1.RowCount = 1000

' Adds the grid to the Record Navigation control.
Me.recordNavigationControl1.Controls.Add(Me.gridControl1)

' Adds the Record Navigation control to the form.
Me.Controls.Add(Me.recordNavigationControl1)
```

## 4.5.2 Built-in Navigation Support for RecordNavigationControl in GridGroupingControl

The GridGrouping control now provides four types of built-in navigation support, enabling users to navigate to the first record, last record, previous record, and next record.

### Use Case Scenarios

When you have lots of records in your application, this feature helps you easily navigate to the required record.

### Methods

Table 15: Methods Table

Method	Description	Parameters	Type	Return Type	Reference links
MoveFirst()	This Method is used to navigate to the first record.	N/A	method	void	N/A.
MoveLast()	This method is used to navigate to the last record.	N/A	<b>method</b>	void	N/A
MoveNext()	This method is used to navigate to the next record.	N/A	<b>method</b>	void	N/A
MovePrevious()	This method is used to navigate to the previous record.	N/A	<b>method</b>	void	N/A

**Sample Link**

A demo of this feature is available in the following location:

`..\\AppData\\Local\\Syncfusion\\EssentialStudio\\{Installed  
Version}\\Windows\\Grid.Grouping.Windows\\Samples\\2.0\\Selection\\Multi Record Selection  
Demo`

**Adding Navigation Bar to the RecordNavigationControl**

The following are steps to add navigation bar:

1. Enable navigation bar by setting the `ShowNavigationBar` property to true. The following code illustrates this:

[C#]

```
this.gridGroupingControl1.ShowNavigationBar = true;
```

[VB]

```
Me.gridGroupingControl1.ShowNavigationBar = True
```

2. Call the methods for navigation bar i.e., MoveFirst(), MoveLast(), MoveNext() and MovePrevious() methods. The following code illustrates this:

[C#]

```
//This property should set to true to show the navigation bar  
this.gridGroupingControl1.ShowNavigationBar = true;  
  
//This method is used to navigate the first record  
this.gridGroupingControl1.RecordNavigationBar.MoveFirst();  
  
//This method is used to navigate the last record  
this.gridGroupingControl1.RecordNavigationBar.MoveLast();  
  
//This method is used to navigate the next record  
this.gridGroupingControl1.RecordNavigationBar.MoveNext();  
  
//This method is used to navigate the previous record  
this.gridGroupingControl1.RecordNavigationBar.MovePrevious();
```

[VB]

```
'This property should set to true to show the navigation bar  
Me.gridGroupingControl1.ShowNavigationBar = True  
  
'This method is used to navigate the first record  
Me.gridGroupingControl1.RecordNavigationBar.MoveFirst()  
  
'This method is used to navigate the last record  
Me.gridGroupingControl1.RecordNavigationBar.MoveLast()  
  
'This method is used to navigate the next record
```

```
Me.gridGroupingControl1.RecordNavigationBar.MoveNext()
```

'This method is used to navigate the previous record

```
Me.gridGroupingControl1.RecordNavigationBar.MovePrevious()
```

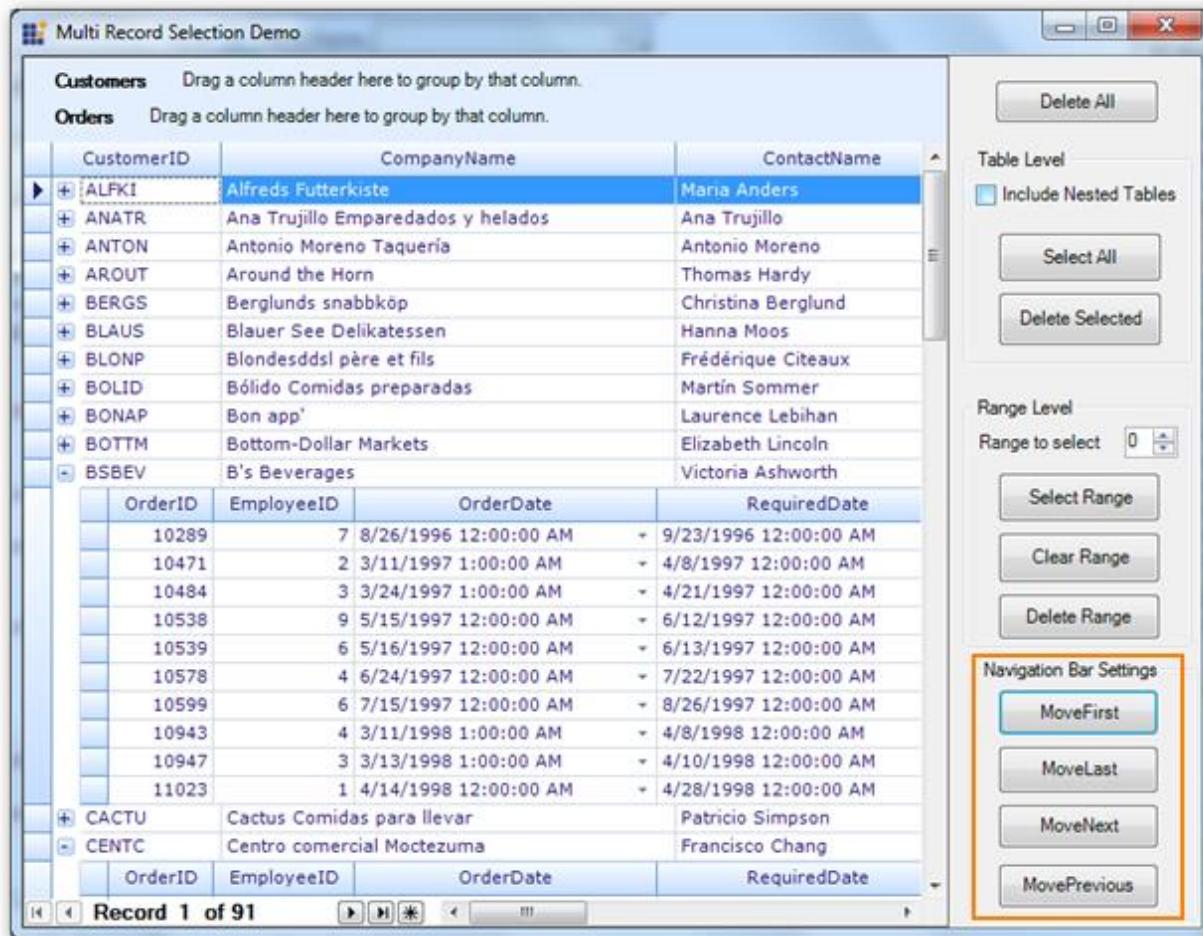


Figure 417: Navigation Bar

## 4.6 Grid Aware Text Box

This class is derived from the textbox and will allow you to bind it to the **CurrentCell** of either a Grid control or a Grid Data Bound Grid. The standard use for such a text box is to provide a special edit bar for editing grid cells. This is similar to the FormulaBar in Excel. In Essential Grid's Formula Grid sample, a similar Formula Bar is implemented by using a Grid Aware Text Box.

Drag the Grid Aware Text Box from your toolbox and drop it on a form along with either a Grid control or a Grid Data Bound Grid. Finally, in your Form\_Load handler, add a call to the GridAwareTextBox.WireGrid to bind the Grid Aware Text Box to the grid. Here are some code samples that will show you how this is done.

[C#]

```
private void Form1_Load(object sender, System.EventArgs e)
{
    // Bind Grid Aware Text Box to grid.
    this.gridAwareTextBox1.WireGrid(this.gridControl1);
}
```

[VB .NET]

```
Private Sub Form1_Load(sender As Object, e As System.EventArgs)

    ' Bind Grid Aware Text Box to grid.
    Me.gridAwareTextBox1.WireGrid(Me.gridControl1)
End Sub
```

## 4.7 Grid Helper Classes

Grid control provides a collection of grid helper classes to achieve specific functionalities. You need to refer the **Syncfusion.GridHelperClasses.Windows** assembly in your Windows application to make use of these helper classes. The various helper classes and their usage are described in the following topics.

### 4.7.1 Grid to PDF Conversion

Grid control provides support to convert the grid content to PDF format. You can convert the grid content into a PDF document for offline verification and/or computation. This is achieved by making use of the **GridPDFConverter** class.

PDF libraries support the conversion of grid content to a PDF page. The following dependent assemblies need to be referenced for this purpose along with the default assemblies that are present in the **References** folder of your Windows application: **Syncfusion.Pdf.Base** and **Syncfusion.GridHelperClasses.Windows**.

Following are the properties, methods and events that are made available to users as part of the GridPDFConverter class.

## Properties

- **ShowHeader**-This property gets or sets a value indicating whether the header should be displayed in the PDF document. Default value is set to *false*.

The following code example illustrates how to set this property.

### 1. Using C#

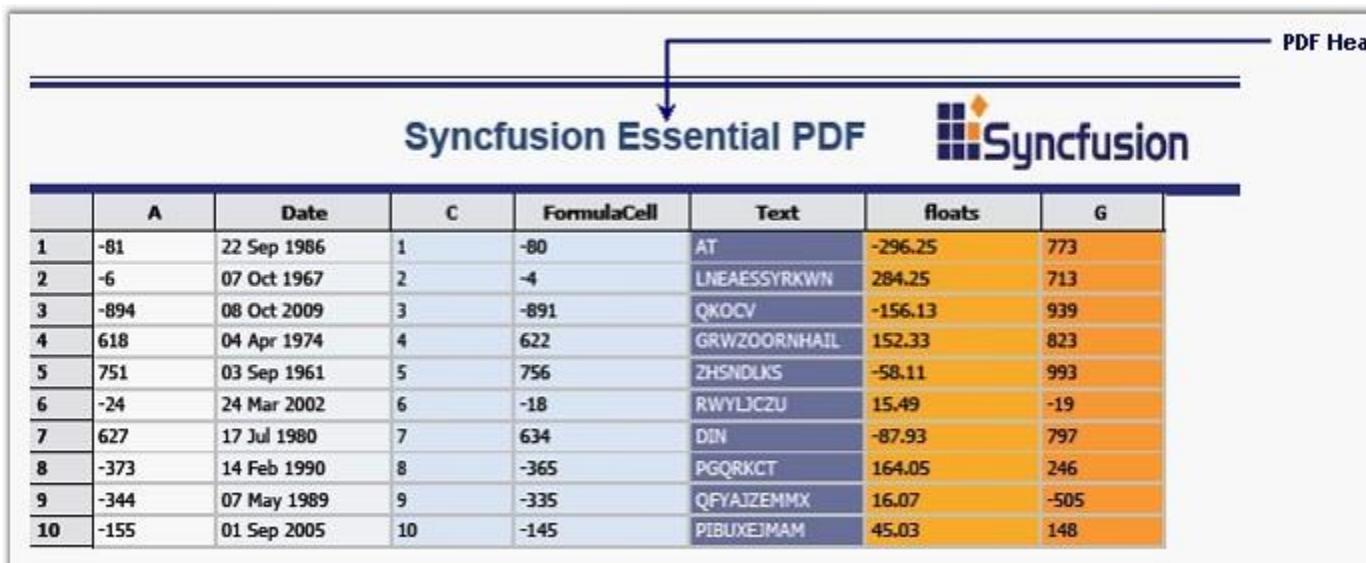
[C#]

```
pdfConvertor.ShowHeader = true;
```

### 2. Using VB.NET

[VB .NET]

```
pdfConvertor.ShowHeader = True
```



A	Date	C	FormulaCell	Text	Floats	G
1	-81	22 Sep 1986	1	-80	AT	-296.25
2	-6	07 Oct 1967	2	-4	LNEAESYRKWN	284.25
3	-894	08 Oct 2009	3	-891	QKOCV	-156.13
4	618	04 Apr 1974	4	622	GRWZOORNHAIL	152.33
5	751	03 Sep 1961	5	756	ZHSNDLKS	-58.11
6	-24	24 Mar 2002	6	-18	RWYLJCZU	15.49
7	627	17 Jul 1980	7	634	DIN	-87.93
8	-373	14 Feb 1990	8	-365	PGQRKCT	164.05
9	-344	07 May 1989	9	-335	QFYAJZEMMX	16.07
10	-155	01 Sep 2005	10	-145	PIBUXEJMAM	45.03

Figure 418: PDF Document displayed with Header

- **ShowFooter**-This property gets or sets a value indicating whether the footer should be displayed in the PDF document. Default value is set to *false*.

The following code example illustrates how to set this property.

1. Using C#

```
[C#]
```

```
pdfConvertor.ShowFooter = true;
```

2. Using VB.NET

```
[VB .NET]
```

```
pdfConvertor.ShowFooter = True
```

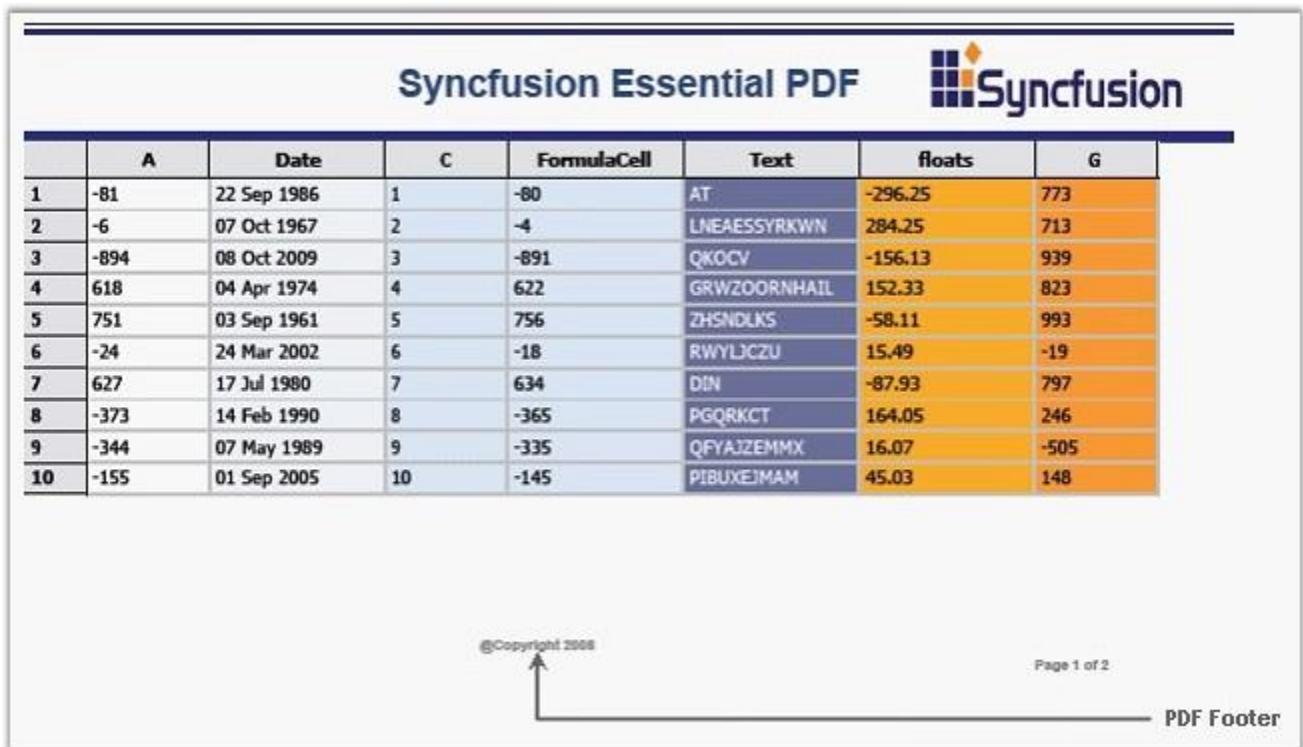


Figure 419: PDF Document displayed with Footer

- **HeaderHeight**-This property gets or sets the height of the header for the PDF document.

The following code example illustrates how to set this property.

1. Using C#

[C#]

```
pdfConvertor.HeaderHeight = 15;
```

2. Using VB.NET

[VB.NET]

```
pdfConvertor.HeaderHeight = 15
```

- **FooterHeight**-This property gets or sets the height of the footer for the PDF document.

The following code example illustrates how to set this property.

1. Using C#

[C#]

```
pdfConvertor.FooterHeight = 20;
```

2. Using VB.NET

[VB.NET]

```
pdfConvertor.FooterHeight = 20
```

- **Margins**-This property gets or sets margins for the PDF document.

The following code example illustrates how to set this property.

1. Using C#

[C#]

```
pdfConvertor.Margins.All = 40;
```

2. Using VB.NET

[VB.NET]

```
pdfConvertor.Margins.All = 40
```

**Methods**

- **ExportToPdf**-This method is used to export the grid to a PDF file.

The following code example illustrates how to use this method.

#### 1. Using C#

**[C#]**

```
GridPDFConverter pdfConvertor = new GridPDFConverter();
pdfConvertor.ExportToPdf("Sample1.pdf", this.gridControl1);
```

#### 2. Using VB.NET

**[VB .NET]**

```
Dim pdfConvertor As New GridPDFConverter()
pdfConvertor.ExportToPdf("Sample1.pdf", Me.gridControl1)
```

### Events

- **DrawPDFFooter**-This event lets you draw a footer for the PDF document.

The following code example illustrates how to handle this event.

#### 1. Using C#

**[C#]**

```
pdfConvertor.DrawPDFFooter += new
GridPDFConverter.DrawPDFHeaderFooterEventHandler(pdfConvertor_DrawPDFFooter);
```

#### 2. Using VB.NET

**[VB .NET]**

```
AddHandler pdfConvertor.DrawPDFFooter, AddressOf
pdfConvertor_DrawPDFFooter
```

- **DrawPDFHeader**-This event lets you draw a header for the PDF document.

The following code example illustrates how to handle this event.

#### 1. Using C#

[C#]

```
pdfConvertor.DrawPDFHeader += new  
GridPDFConverter.DrawPDFHeaderFooterEventHandler(pdfConvertor_DrawPDFHe  
ader);
```

2. Using VB.NET

[VB .NET]

```
AddHandler pdfConvertor.DrawPDFHeader, AddressOf  
pdfConvertor_DrawPDFHeader
```

## 4.7.2 Resizing Heights of Individual Rows in Grid

Grid Grouping control does not support resizing heights of individual rows in the grid. This feature has been newly added and can be implemented by initializing an instance of the **AllowResizingIndividualRows** class to the **GridEngineFactory** in the Form's constructor of your Windows application. The following code examples illustrate how to do this.

1. Using C#

[C#]

```
GridEngineFactory.Factory = new  
Syncfusion.GridHelperClasses.AllowResizingIndividualRows();
```

2. Using VB.NET

[VB .NET]

```
GridEngineFactory.Factory = New  
Syncfusion.GridHelperClasses.AllowResizingIndividualRows()
```

You can make use of the **AllowResizingIndividualRows** class by adding the dependent assembly, **Syncfusion.GridHelperClasses.Windows**, to the **References** folder in your application.

The following screen shot illustrates how the heights of individual rows in the grid have been resized.

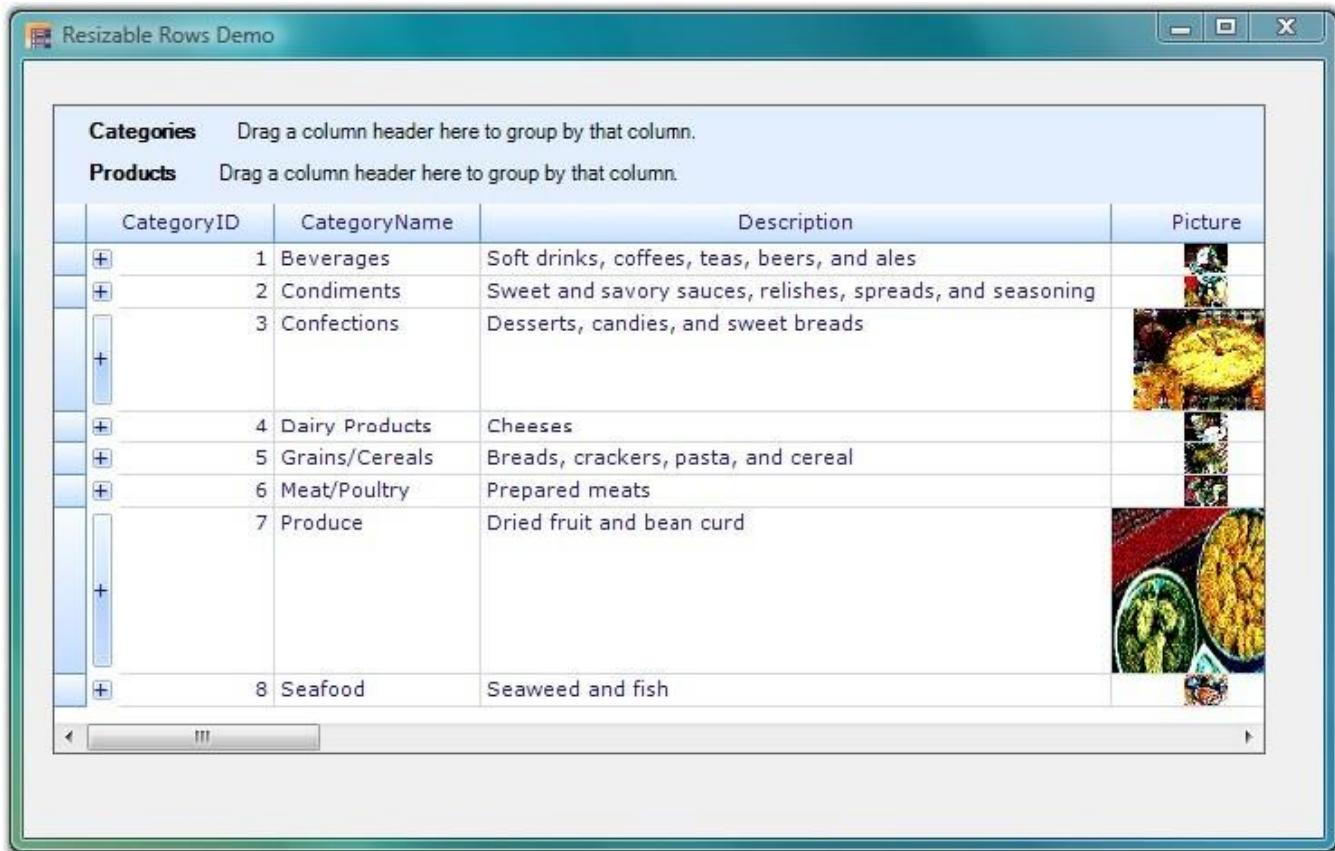


Figure 420:: Grid Grouping control with Row Heights Resized

### 4.7.3 Grid Dynamic Filter

The **GridDynamicFilter** class is used to wire a custom filter bar to the Grid Grouping control by replacing the default filter bar. The existing filter bar logic is extended to make the filter easy to use. This feature displays filtered results as you type each character.

The new filter bar adds two cell buttons, **Filter** button and **Clear Filter** button, inside every filter bar cell. The Filter button is used to display a list of the available Compare Operators in a drop down. The selected operator will then be associated with the value present in the filter bar cell to form a filter string. The Clear Filter button, as its name indicates, clears the record filters of the respective column. This button will be displayed for a filter bar cell, only when that particular cell is in focus.

The following code example illustrates how to invoke the Grid Dynamic Filter.

## 1. Using C#

[C#]

```
GridEngineFactory.Factory = new
Syncfusion.GridHelperClasses.AllowResizingIndividualRows();
```

## 2. Using VB.NET

[VB .NET]

```
GridEngineFactory.Factory = New
Syncfusion.GridHelperClasses.AllowResizingIndividualRows()
```

The following screen shot illustrates Grid Grouping control with filter drop down.

The screenshot shows a Windows application window containing a grid control. The grid has columns for Address, City, and CompanyName. A context menu is open over the first row of the grid, specifically over the 'Address' column. The menu is titled 'StartsWith' and lists several comparison operators: StartsWith, EndsWith, Equals, NotEquals, LessThan, LessThanOrEqualTo, GreaterThan, GreaterThanOrEqualTo, Like, Match, and Expression MATCH. The 'Address' column header also features a small dropdown arrow icon.

Address	City	CompanyName
Obere Str. 57	freds Futterkiste	
Avda. de la Constitución 2222	ma Trujillo Emparedados y helados	
Mataderos 2312	Antonio Moreno Taquería	
120 Hanover Sq.	ound the Horn	
Berguvsvägen 8	erglunds snabbköp	
Orders: 1 Items		
EmployeeID	Freight	OrderDate
		OrderID
		Region
Forsterstr. 57	Mannheim	Blauer See Delikatessen
24, place Kléber	Strasbourg	Blondel père et fils
C/ Araquil, 67	Madrid	Bólido Comidas preparadas
12, rue des Bouchers	Marseille	Bon app'
23 Tsawassen Blvd.	Tsawassen	Bottom-Dollar Markets

Figure 421: Grid Grouping control with Filter Drop Down

### Support to Save and Load Compare Operators State in Grid Dynamic Filter

**GridDynamicFilter** in **GridGroupingControl** is now enhanced a functionality to serialize/de-serialize the **compareoperator** images in button. This can be achieved by handling the following method calls.

```
<code>filter.LoadCompareOperator();</code>
```

```
<code>filter.SaveCompareOperator();</code>
```

When the code runs, the following output displays.

	Address	City	CompanyName
+	Obere Str. 57	Berlin	Alfreds Futterkiste

*Figure 422: CompareOperator states restored in respective columns*

#### Apply Filter Only on Lost Focus in GridDynamicFilter

ApplyFilterOnlyOnCellLostFocus property enables you to turn off/on the filtering on each key stroke in GridDynamicFilter.

Set ApplyFilterOnlyOnCellLostFocus property to true to filter only when the filter cell lost focus.

This disables filtering for each key stroke including Enter, arrow keys, and tab keys.

Defaults value is false and allows filtering for each key stroke.

The following code illustrates how to add ApplyFilterOnlyOnCellLostFocus property.

[C#]

```
GridDynamicFilter filter = new GridDynamicFilter();
filter.ApplyFilterOnlyOnCellLoseFocus= true;
```

when the code runs, the following output displays.

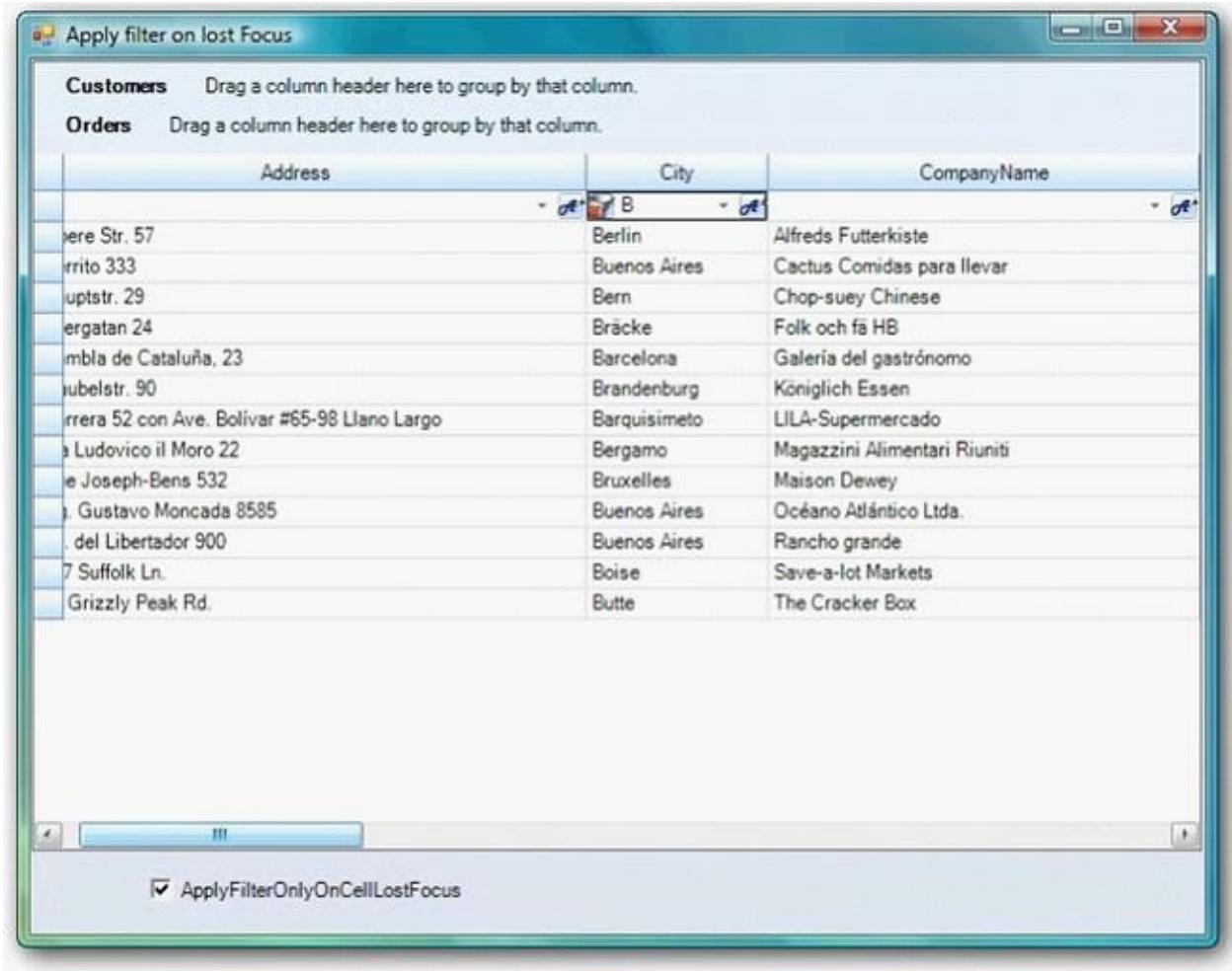


Figure 423: Filter on Lost Focus on lost focus

#### 4.7.4 Setting up Foreign Key Relations

**GridForeignKeyHelper** class is used to set up foreign key relations to perform foreign key lookups. With this class, you can easily set up a foreign table with a single method call instead of implementing numerous steps.

The following code example illustrates how to use this class.

1. Using C#

[C#]

```
GridForeignKeyHelper.SetupForeignTableLookUp(gridGroupingControll1,  
"Country", countries, "CountryCode", "CountryName");
```

2. Using VB.NET

[VB .NET]

```
GridForeignKeyHelper.SetupForeignTableLookUp(gridGroupingControll1,  
"Country", countries, "CountryCode", "CountryName")
```



**Note:**

***The first argument in this method is an instance of the Grid Grouping control.***

***The second argument is the column name of the Parent table's Value Member.***

***The third argument is the name of the Foreign table.***

***The fourth argument is the column name of the Child table's Value Member.***

***The fifth argument is the column name of the Child table's Display Member.***

The following screen shot illustrates Foreign Key Relations in the Grid Grouping control.

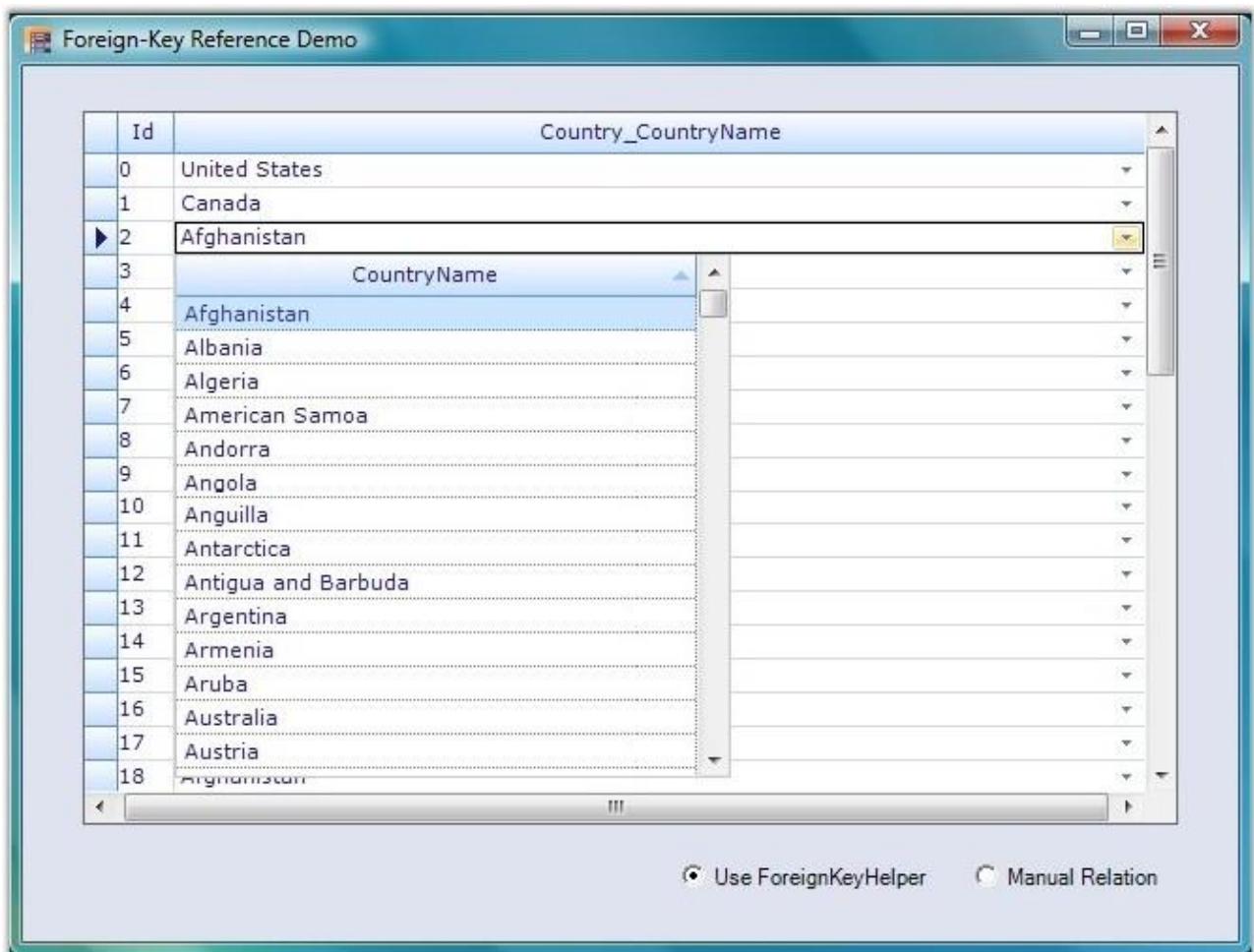


Figure 424: Grid Grouping control with Foreign Key Relations

#### 4.7.5 Custom Cell Types

Following are the custom cell types supported by the Grid Helper Library.

- a. ButtonEdit
- b. CalculatorTextBox
- c. Calendar
- d. DateTimePicker
- e. FNumericUpDown
- f. GridinCell
- g. LinkLabelCell
- h. PictureBox

## **ButtonEdit**

You can implement a Button Edit control in grid cells by using the **ButtonEdit** cell type. **ButtonEdit** cell types can be used by initializing the **ButtonEditTextProperties** class for the grid cells.

Following are the Button Edit cell types available in the Grid control.

- Browse
- Check
- Down
- Image
- Left
- Leftend
- None
- Redo
- Right
- Rightend
- Undo
- Up

The following code example illustrates how to set the grid cell type to **ButtonEdit**.

### 1. Using C#

```
[C#]

// Register the Cell Type with Grid control.
RegisterCellModel.GridCellType(gridControl1,
CustomCellTypes.ButtonEdit);
Syncfusion.GridHelperClasses.ButtonEditTextProperties sp;
sp = new
Syncfusion.GridHelperClasses.ButtonEditTextProperties(this.gridControl1
1[rowIndex, colIndex]);
sp.ButtonEditInfo.ButtonEditType =
Syncfusion.GridHelperClasses.ButtonType.Browse;

this.gridControl1[2, 2].CellType = "ButtonEdit";
```

### 2. Using VB.NET

**[VB.NET]**

```
' Register the Cell Type with Grid control.  
RegisterCellModel.GridCellType(gridControl1,  
CustomCellTypes.ButtonEdit)  
Dim sp As Syncfusion.GridHelperClasses.ButtonEditStyleProperties  
sp = New  
Syncfusion.GridHelperClasses.ButtonEditStyleProperties(Me.gridControl1(  
rowIndex, colIndex))  
sp.ButtonEditInfo.ButtonEditType =  
Syncfusion.GridHelperClasses.ButtonType.Browse  
gridControl1[2, 2].CellType = "ButtonEdit"
```

Following screen shots illustrate different Button Edit cell types.

	A	B
1		
2		...
3		
4		
5		

Figure 425: "Browse" Button Edit Cell Type

	A	B
1		
2		<input checked="" type="checkbox"/>
3		
4		
5		

Figure 426: "Check" Button Edit Cell Type

	A	B
1		
2		▼
3		
4		
5		

Figure 427: "Down" Button Edit Cell Type

	A	B
1		
2		
3		
4		
5		

Figure 428: "Image" Button Edit Cell Type

	A	B
1		
2		
3		
4		
5		

Figure 429: "Left" Button Edit Cell Type

	A	B
1		
2		
3		
4		
5		

Figure 430: "Leftend" Button Edit Cell Type

	A	B
1		
2		
3		
4		
5		

Figure 431: "None" Button Edit Cell Type

	A	B
1		
2		↶
3		
4		
5		

Figure 432: "Redo" Button Edit Cell Type

	A	B
1		
2		▶
3		
4		
5		

Figure 433: "Right" Button Edit Cell Type

	A	B
1		
2		▶
3		
4		
5		

Figure 434: "Rightend" Button Edit Cell Type

	A	B
1		
2		↶
3		
4		
5		

Figure 435: "Undo" Button Edit Cell Type

	A	B
1		
2		▲
3		
4		
5		

Figure 436: "Up" Button Edit Cell Type

### **CalculatorTextBox**

You can implement a Calculator control in grid cells by using the **CalculatorTextBox** cell type. This cell type is implemented as a drop-down container, embedded into the cell. The drop down contains the calculator which displays and stores the value in the cell.

The following code example illustrates how to set the grid cell type to **CalculatorTextBox**.

#### 1. Using C#

[C#]

```
RegisterCellModel.GridCellType(gridControl1,
CustomCellTypes.CalculatorTextBox);

CalculatorControl c1 = new CalculatorControl();
c1.BorderStyle = Border3DStyle.Flat;
c1.ButtonStyle = Syncfusion.Windows.Forms.ButtonAppearance.Office2007;
c1.UseVisualStyle = true;

GridStyleInfo style = gridControl1[4, 2];
style.CellType = "CalculatorTextBox";
style.Control = c1;
```

#### 2. Using VB.NET

[VB .NET]

```
RegisterCellModel.GridCellType(gridControl1, CustomCellTypes.CalculatorTextBox)

Dim c1 As New CalculatorControl()
c1.BorderStyle = Border3DStyle.Flat
c1.ButtonStyle = Syncfusion.Windows.Forms.ButtonAppearance.Office2007
c1.UseVisualStyle = True
```

```
Dim style As GridStyleInfo = gridControl1(4, 2)
style.CellType = "CalculatorTextBox"
style.Control = c1
```

Following screen shot illustrates CalculatorTextBox cell type in the Grid control.

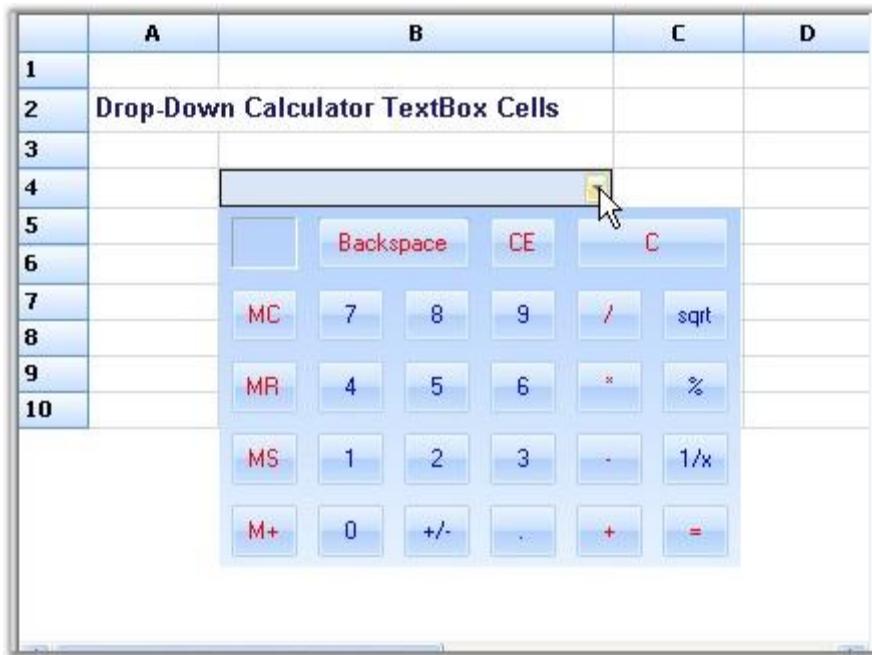


Figure 437: CalculatorTextBox cell type in Grid Control

## Calendar

You can implement a MonthCalendar control in a grid cell by enabling the **Calendar** cell type for that particular cell.

The following code example illustrates how to set the grid cell type to Calendar.

### 1. Using C#

```
[C#]

RegisterCellModel.GridCellType(gridControl1, CustomCellTypes.Calendar);
GridStyleInfo style;
style = gridControl1[row, 2];
style.CellType = "Calendar";

// Provide a Month Calendar control for drawing cell contents.
```

```
style.Control = new MonthCalendar();
```

## 2. Using VB.NET

### [VB.NET]

```
RegisterCellModel.GridCellType(gridControl1, CustomCellTypes.Calendar)
Dim style As GridStyleInfo
style = gridControl1(row, 2)
style.CellType = "Calendar"

' Provide a Month Calendar control for drawing cell contents.
style.Control = New MonthCalendar()
```

Following screen shot illustrates Calendar cell type in Grid control.

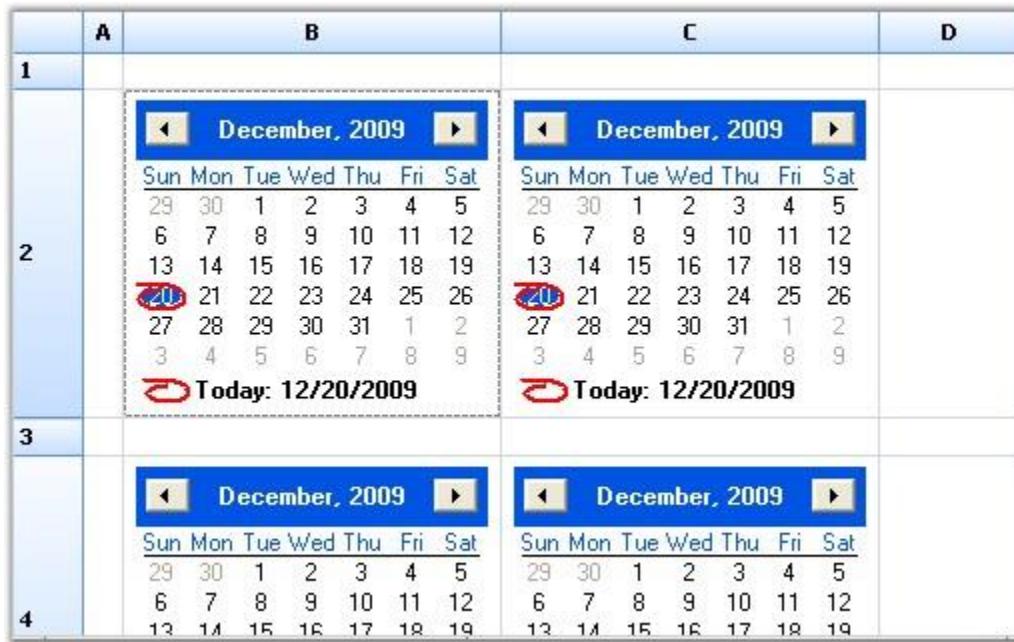


Figure 438: Calendar cell type in Grid Control

### DateTimePicker

You can implement a Date Time Picker control in grid cells by using the **DateTimePicker** cell type. This cell type is implemented as a drop-down container, embedded into the cell, where the date and time picker is added. The drop down contains the calendar which displays and stores the date value in the cell. Various formats for the date and time can be specified by using the **Format** style property.

The following code example illustrates how to set the grid cell type to DateTimePicker.

1. Using C#

[C#]

```
RegisterCellModel.GridCellType(gridControl1,
CustomCellTypes.DateTimePicker);

// Set up DateTimePicker Cells.
this.gridControl1[4, 2].CellType = "DateTimePicker";
this.gridControl1[4, 2].CellValueType = typeof(DateTime);
this.gridControl1[4, 2].CellValue = DateTime.Now;
this.gridControl1[4, 2].Format = "MM/dd/yyyy hh:mm";
```

2. Using VB.NET

[VB .NET]

```
RegisterCellModel.GridCellType(gridControl1,
CustomCellTypes.DateTimePicker)

' Set up DateTimePicker Cells.
Me.gridControl1(4, 2).CellType = "DateTimePicker"
Me.gridControl1(4, 2).CellValueType = GetType(DateTime)
Me.gridControl1(4, 2).CellValue = DateTime.Now
Me.gridControl1(4, 2).Format = "MM/dd/yyyy hh:mm"
```

Following screen shot illustrates DateTimePicker cell type in the Grid control.



Figure 439: DateTimePicker cell type in Grid Control

### FNumericUpDown

You can implement a Float Numeric Up Down control in grid cells by using the **FNumericUpDown** cell type. FNumericUpDown cell types can be used by initializing the **FloatNumericUpDownStyleProperties** class for the grid cells. This will allow you set the limitations of the numeric values and several other properties can also be added as follows.

The following code example illustrates how to set the grid cell type to FNumericUpDown.

#### 1. Using C#

```
[C#]

RegisterCellModel.GridCellType(gridControl1,
CustomCellTypes.FNumericUpDown);
GridStyleInfo style = this.gridControl1[2, 2];

// Set up FNumericUpDown Cell.
style.CellType = "FNumericUpDown";
style.Text = "0.5";
```

#### 2. Using VB.NET

**[VB.NET]**

```
RegisterCellModel.GridCellType(gridControl1,
CustomCellTypes.FNumericUpDown)
Dim style As GridStyleInfo = Me.gridControl1(2, 2)

' Set up FNumericUpDown Cell.
style.CellType = "FNumericUpDown"
style.Text = "0.5"
```

Following screen shot illustrates FNumericUpDown cell type in the Grid control.

	A	B	C	D	E	F
1	<b>Allow Decimal Increment and Decrement(step by 0.2,0.01,0.001)</b>					
2		0.5	10.10	15.000		
3						
4	<b>Option to decrease number beyond Zero(Negative numbers allowed)</b>					
5		-4				
6						
7	<b>Use Up/Down arrow keys from Keyboard to increment/decrement the value</b>					
8		100000				
9						
10	<b>Setting the Orientation property</b>					
11		5.5	◀ ▶	10.0		
12						

Figure 440: FNumericUpDown cell type in Grid Control

**GridinCell**

The **GridinCell** cell type provides a covered range of cells to embed the grid, which is added as a control to the cells. The registered cell model initializes the range by calculating the size of the grid control to be embedded, and adds styles such as borders and scroll bars to have the control within the range.

The following code example illustrates how to set the grid cell type to GridinCell.

1. Using C#

[C#]

```
RegisterCellModel.GridCellType(gridControl1,
CustomCellTypes.GridinCell);
GridControl grid;
this.gridControl1[3, 2].CellType = "GridinCell";
this.gridControl1.CoveredRanges.Add(GridRangeInfo.Cells(3, 2, 7, 4));
grid = new
Syncfusion.GridHelperClasses.CellEmbeddedGrid(this.gridControl1);
grid.BackColor = Color.FromArgb(0xb4, 0xe7, 0xf2);
grid.RowCount = 10;
grid.ColCount = 4;
grid[1, 1].Text = "this is a 10x4 grid";
grid.ThemesEnabled = true;
this.gridControl1[3, 2].Control = grid;
this.gridControl1.Controls.Add(grid);
```

## 2. Using VB.NET

[VB.NET]

```
RegisterCellModel.GridCellType(gridControl1,
CustomCellTypes.GridinCell)
Dim grid As GridControl
Me.gridControl1(3, 2).CellType = "GridinCell"
Me.gridControl1.CoveredRanges.Add(GridRangeInfo.Cells(3, 2, 7, 4))
grid = New
Syncfusion.GridHelperClasses.CellEmbeddedGrid(Me.gridControl1)
grid.BackColor = Color.FromArgb(&HB4, &HE7, &HF2)
grid.RowCount = 10
grid.ColCount = 4
grid(1, 1).Text = "this is a 10x4 grid"
grid.ThemesEnabled = True
Me.gridControl1(3, 2).Control = grid
Me.gridControl1.Controls.Add(grid)
```

Following screen shot illustrates GridinCell cell type in the Grid control.

A	B	C	D	E	F	G	H	I	
	A	B	C	D		A	B	C	D
1	this is a 10x4 grid					1	this is a 5x5 grid		
2						2			
3						3			
4									
	A	B	C	D	E	F	G	H	I
1	this is a 20x20 grid								J
2									
3									
4									

Figure 441: GridInCell cell type in Grid Control

### LinkLabelCell

The **LinkLabelCell** cell type displays text which can be hyperlinked to a specific location. The path to be hyperlinked is specified by using the **Tag** property.

The following code example illustrates how to set the grid cell type to LinkLabelCell.

#### 1. Using C#

[C#]

```
RegisterCellModel.GridCellType(gridControl1,
CustomCellTypes.LinkLabelCell);
int rowIndex = 5;
gridControl1[rowIndex, 2].CellType = "LinkLabelCell";
gridControl1[rowIndex, 2].Text = "Syncfusion, Inc.";
gridControl1[rowIndex, 2].Font.Bold = true;
gridControl1[rowIndex, 2].Tag = "http://www.syncfusion.com";
gridControl1[rowIndex, 2].HorizontalAlignment =
GridHorizontalAlignment.Center;
```

#### 2. Using VB.NET

**[VB.NET]**

```
RegisterCellModel.GridCellType(gridControl1,
CustomCellTypes.LinkLabelCell)
Dim rowIndex As Integer = 5
gridControl1(rowIndex, 2).CellType = "LinkLabelCell"
gridControl1(rowIndex, 2).Text = "Syncfusion, Inc."
gridControl1(rowIndex, 2).Font.Bold = True
gridControl1(rowIndex, 2).Tag = "http://www.syncfusion.com"
gridControl1(rowIndex, 2).HorizontalAlignment =
GridHorizontalAlignment.Center
```

Following screen shot illustrates LinkLabelCell cell type in the Grid control.

	A	B	C	D
1				
2				
3				
4				
5		<a href="#">Syncfusion, Inc.</a>		
6		<a href="#">Windows Forms FAQ</a>		
7		<a href="#">Microsoft Windows Forms</a>		
8		<a href="#">MSDN</a>		
9		<a href="#">Yahoo</a>		
10		<a href="#">Google</a>		
11				

Figure 442: LinkLabelCell cell type in Grid Control

### PictureBox

The **PictureBox** cell type can be embedded into a cell by calculating the size of the picture and extending the width and height of the cell accordingly. The **PictureBoxStyleProperties** class is used to specify the style for the Picture Box control.

The following code example illustrates how to set the grid cell type to PictureBox.

#### 1. Using C#

**[C#]**

```
RegisterCellModel.GridCellType(gridControl1,
CustomCellTypes.PictureBox);
```

```
Syncfusion.GridHelperClasses.PictureBoxStyleProperties tsp = new  
Syncfusion.GridHelperClasses.PictureBoxStyleProperties(new  
GridStyleInfo(gridControl1.TableStyle));  
tsp.SizeMode = PictureBoxSizeMode.AutoSize;  
  
Syncfusion.GridHelperClasses.PictureBoxStyleProperties sp;  
  
GridStyleInfo style;  
  
gridControl1.ColWidths[1] = 20;  
  
style = gridControl1[2, 2];  
style.CellType = "PictureBox";  
  
sp = new Syncfusion.GridHelperClasses.PictureBoxStyleProperties(style);  
sp.Image = GetImage("one.jpg");
```

## 2. Using VB.NET

### [VB.NET]

```
RegisterCellModel.GridCellType(gridControl1,  
CustomCellTypes.PictureBox)  
Dim tsp As New  
Syncfusion.GridHelperClasses.PictureBoxStyleProperties(New  
GridStyleInfo(gridControl1.TableStyle))  
tsp.SizeMode = PictureBoxSizeMode.AutoSize  
Dim sp As Syncfusion.GridHelperClasses.PictureBoxStyleProperties  
  
Dim style As GridStyleInfo  
  
gridControl1.ColWidths(1) = 20  
  
style = gridControl1(2, 2)  
style.CellType = "PictureBox"  
  
sp = New Syncfusion.GridHelperClasses.PictureBoxStyleProperties(style)  
sp.Image = GetImage("one.jpg")
```

Following screen shot illustrates PictureBox cell type in the Grid control.

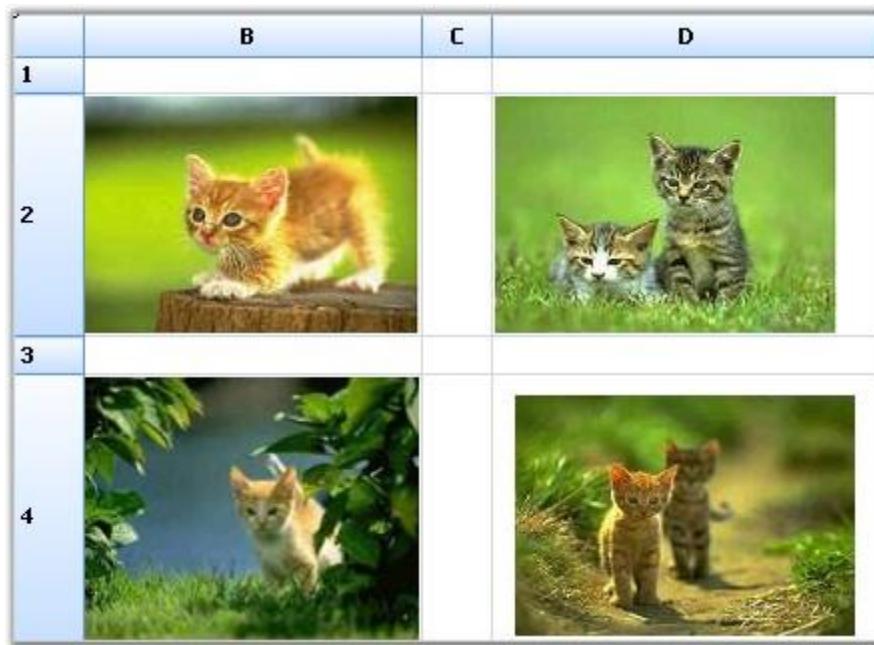


Figure 443: PictureBox cell type in Grid Control

#### 4.7.6 Grid Field Chooser

You can customize the column appearance of a Grid Grouping control by using a plug-in utility called the **Field Chooser**. The **FieldChooser** class can be associated with the Grid Grouping control to add/remove columns from the grid. The following code example illustrates this.

##### 1. Using C#

[C#]

```
FieldChooser fchooser = new FieldChooser(this.gridGroupingControl1);
```

##### 2. Using VB.NET

[VB .NET]

```
Dim fchooser As New FieldChooser(Me.gridGroupingControl1)
```

Following screen shot shows Grid Grouping control with the Field dialog box.

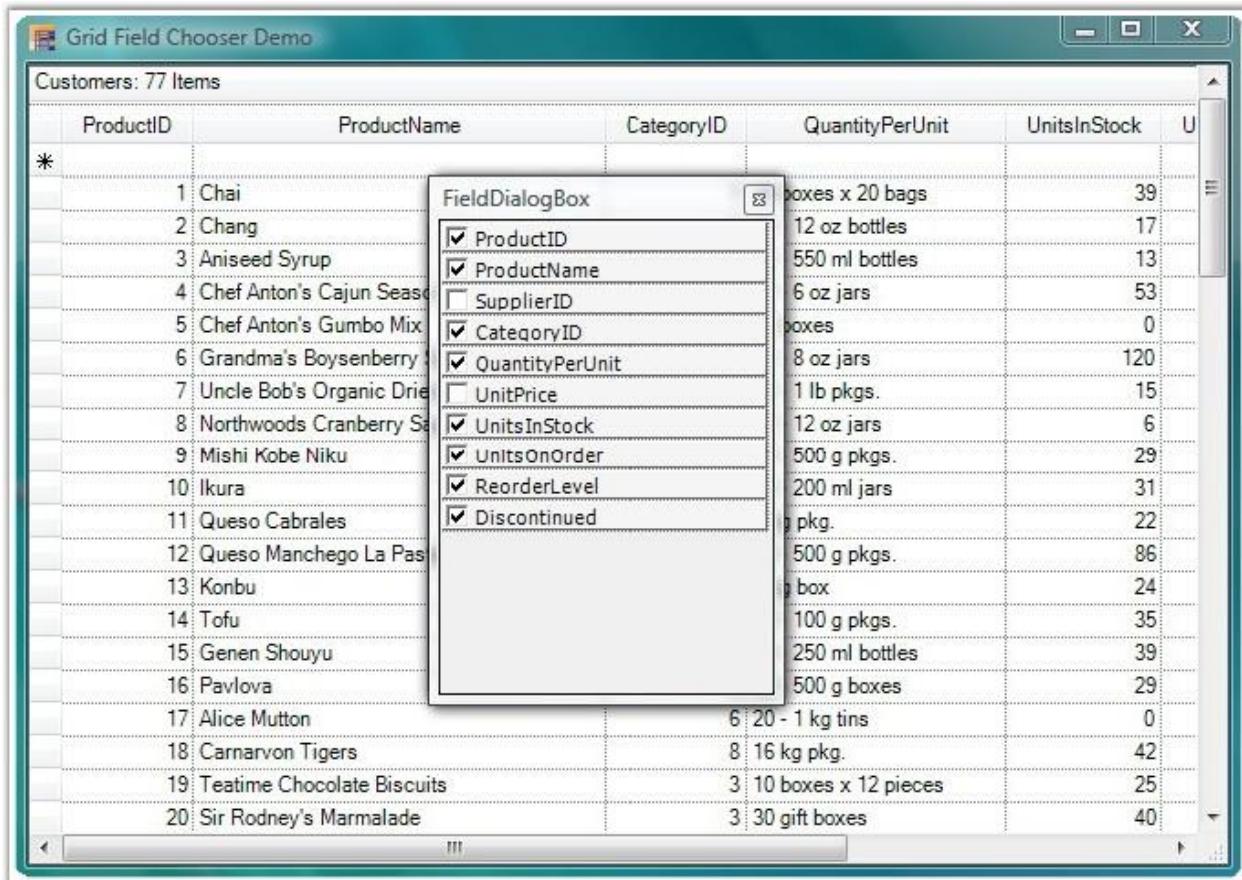


Figure 444: Grid Grouping control with Field Dialog Box

To add/remove columns by using the Field Chooser, right-click on a column header, and select the **Field Chooser** menu item to view the **Field** dialog box. This dialog box lists all the column names with check boxes. You can select/clear the check boxes to add/remove the respective columns from the Grid Grouping control.

A sample demonstration of the Grid Field Chooser feature is available in the following sample installation path.

`<Install Location>\Syncfusion\EssentialStudio\Version Number\Windows\Grid.Grouping.Windows\Samples\2.0\Grouping Grid Layout\Grid Field Chooser Demo`

#### 4.7.7 Filtering By Display Member

This topic elaborates on filtering columns in the Grid Data Bound Grid and Grid Grouping controls by their display member.

### Filtering Columns in Grid Data Bound Grid

The **GridDataBoundGridFilterBarExt** class provides support to filter a column in the Grid Data Bound Grid by its display member instead of the value member. This is accomplished by implementing a custom filter bar cell by replacing the default filter bar cell.

Following code example illustrates how to wire the **GridDataBoundGridFilterBarExt** to the Grid Data Bound Grid.

#### 1. Using C#

[C#]

```
private GridDataBoundGridFilterBarExt filterBar;  
filterBar = new GridDataBoundGridFilterBarExt();  
filterBar.WireGrid(this.gridBoundGrid1);
```

#### 2. Using VB.NET

[VB .NET]

```
Private filterBar As GridDataBoundGridFilterBarExt  
filterBar = New GridDataBoundGridFilterBarExt()  
filterBar.WireGrid(Me.gridBoundGrid1)
```

Following screen shot illustrates how to filter a column in the Grid Data Bound Grid by its display member.



Figure 445: Filtering a column in the Grid Data Bound Grid by its Display Member

### Filtering Columns in Grid Grouping Control

The **GroupingGridFilterBarExt** class provides support to filter a column in the Grid Grouping control by its display member instead of the value member. This is accomplished by implementing a custom filter bar cell by replacing the default filter bar cell.

Following code example illustrates how to wire the GroupingGridFilterBarExt to the Grid Grouping control.

#### 1. Using C#

```
[C#]

private GroupingGridFilterBarExt gGCFilter;
this.gGCFilter = new GroupingGridFilterBarExt();
this.gGCFilter.WireGrid(this.gridGroupingControl1);
```

#### 2. Using VB.NET

**[VB .NET]**

```
Private gGCFilter As GroupingGridFilterBarExt
Me.gGCFilter = New GroupingGridFilterBarExt()
Me.gGCFilter.WireGrid(Me.gridGroupingControl1)
```

Following screen shot illustrates how to filter a column in the Grid Grouping control by its display member.



Figure 446: Filtering a column in the Grid Grouping control by its Display Member

#### 4.7.8 Grid Format Cell Dialog

The **Grid Format Cell Dialog**, similar to the Excel-like Format Cell Dialog, enables users to format the cells dynamically. It provides options to customize the cell font family, font color, font size, font style, font effects, background, alignment, text format, and so on. You can instantiate the Grid Format Cell Dialog by using the following code.

1. Using C#

[C#]

```
GridFormatCellDialog f = new GridFormatCellDialog(this.gridControl1);
f.ShowDialog();
```

## 2. Using VB.NET

[VB .NET]

```
Dim f As New GridFormatCellDialog(Me.gridControl1)
f.ShowDialog()
```

Following screen shot illustrates the Format Cell Dialog of the Grid control.

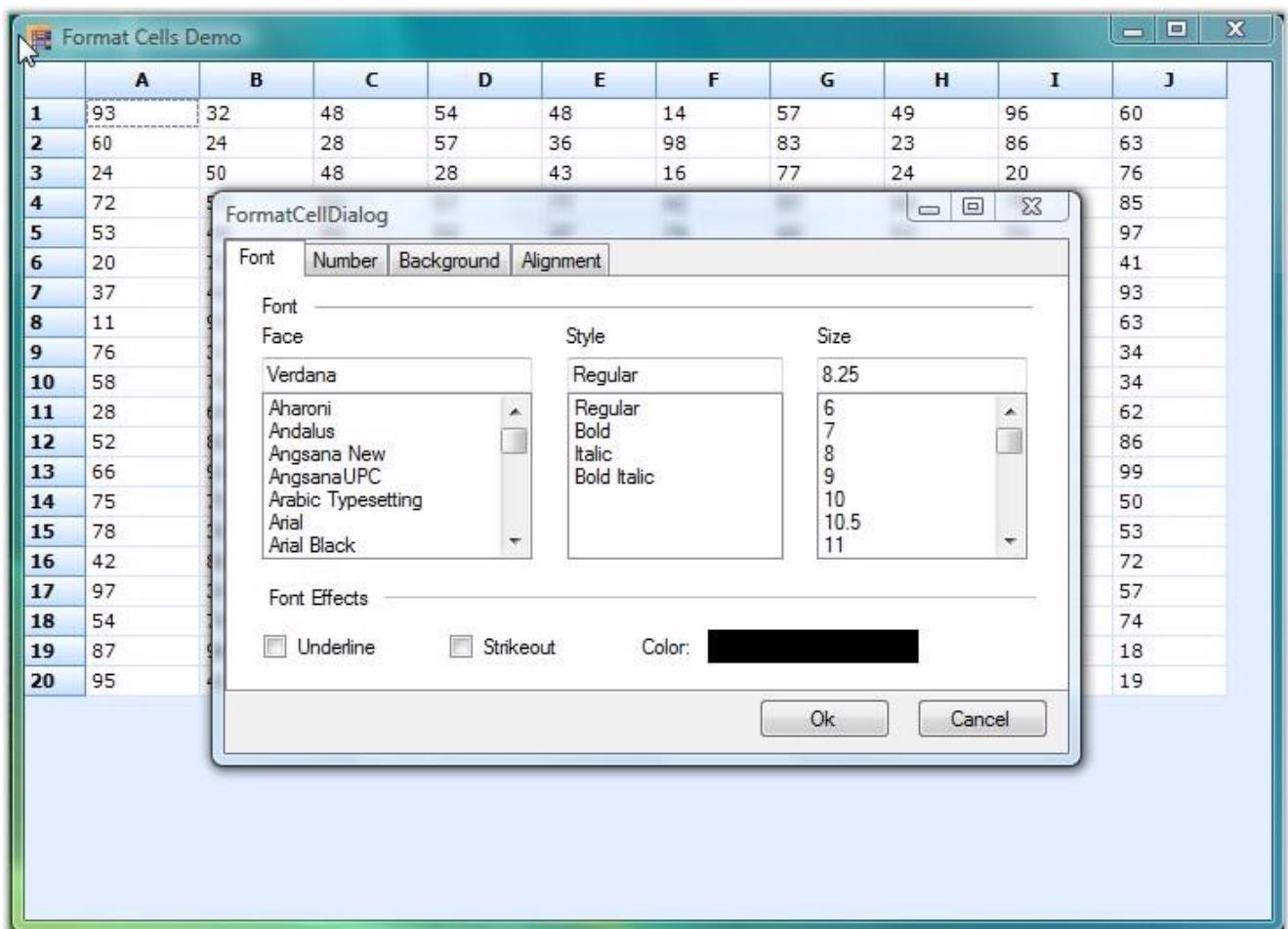


Figure 447: Grid control with Format Cell Dialog

## 4.7.9 Printing

This topic elaborates on printing options supported by the Grid control.

### Printing Multiple Grids

You can print multiple grids across various pages by using the **GridPrintDocumentAdv** helper class. This is achieved by drawing the full size grid to a large bitmap, and then drawing this bitmap, scaled to fit the output page.

By using the **ScaleColumnsToFitPage** property, columns can be scaled to fit on a single page. Headers and footers can also be added by using the **DrawGridPrintHeader** and **DrawGridPrintFooter** events. The following code examples illustrate how to do this.

#### 1. Using C#

[C#]

```
Syncfusion.GridHelperClasses.GridPrintDocumentAdv pd = new  
Syncfusion.GridHelperClasses.GridPrintDocumentAdv(this.gridControl1);  
pd.DefaultPageSettings.Margins = new  
System.Drawing.Printing.Margins(25, 25, 25, 25);  
pd.HeaderHeight = 70;  
pd.FooterHeight = 50;  
  
pd.ScaleColumnsToFitPage = true;  
  
PrintPreviewDialog previewDialog = new PrintPreviewDialog();  
previewDialog.Document = pd;  
previewDialog.Show();
```

#### 2. Using VB.NET

[VB .NET]

```
Dim pd As New  
Syncfusion.GridHelperClasses.GridPrintDocumentAdv(Me.gridControl1)  
pd.DefaultPageSettings.Margins = New  
System.Drawing.Printing.Margins(25, 25, 25, 25)  
pd.HeaderHeight = 70  
pd.FooterHeight = 50  
  
pd.ScaleColumnsToFitPage = True
```

```
Dim previewDialog As New PrintPreviewDialog()
previewDialog.Document = pd
previewDialog.Show()
```

The following screen shots illustrate the Print Preview feature of the Grid control.

	OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	ShippedDate	ShipVia	Freight	ShipName
20	10267	FRANK	4	00/29/1996	00/26/1996	00/06/1996	1	208.58	Frankenversand
21	10268	GROSR	8	00/30/1996	00/27/1996	00/02/1996	3	66.29	GROSELLA-Restaurante
22	10269	WHITC	5	00/31/1996	00/14/1996	00/09/1996	1	4.56	White Clover Markets
23	10270	WARTH	1	00/01/1996	00/29/1996	00/02/1996	1	136.54	Wartian Herku
24	10271	SPLIR	6	00/01/1996	00/29/1996	00/30/1996	2	4.54	Split Rail Beer & Ale
25	10272	RATTC	6	00/02/1996	00/30/1996	00/06/1996	2	98.03	Rattlesnake Canyon Grocery
26	10273	QUICK	3	00/05/1996	00/02/1996	00/12/1996	3	76.07	QUICK-Stop
27	10274	VINET	6	00/06/1996	00/03/1996	00/16/1996	1	6.01	Vins et alcools Chevalier
28	10275	MAGAA	1	00/07/1996	00/04/1996	00/09/1996	1	26.93	Magazzini Alimentari Riuniti
29	10276	TORTU	8	00/08/1996	00/22/1996	00/14/1996	3	13.84	Tortuga Restaurante
30	10277	MORGK	2	00/09/1996	00/06/1996	00/13/1996	3	125.77	Morgenstern Gesundkost
31	10278	BERGS	8	00/12/1996	00/09/1996	00/16/1996	2	92.69	Berglunds snabbköp
32	10279	LEHMS	8	00/13/1996	00/10/1996	00/16/1996	2	25.83	Lehmans Marktstand
33	10280	BERGS	2	00/14/1996	00/11/1996	00/12/1996	1	8.98	Berglunds snabbköp
34	10281	ROMEY	4	00/14/1996	00/28/1996	00/21/1996	1	2.94	Romero y tomillo
35	10282	ROMEY	4	00/15/1996	00/12/1996	00/21/1996	1	12.69	Romero y tomillo
36	10283	LILAS	3	00/16/1996	00/13/1996	00/23/1996	3	84.81	LILA-Supermercado
37	10284	LEHMS	4	00/19/1996	00/16/1996	00/27/1996	1	76.56	Lehmans Marktstand
38	10285	QUICK	1	00/20/1996	00/17/1996	00/26/1996	2	76.83	QUICK-Stop
39	10286	QUICK	8	00/21/1996	00/18/1996	00/30/1996	3	229.24	QUICK-Stop

Figure 448: Grid Control

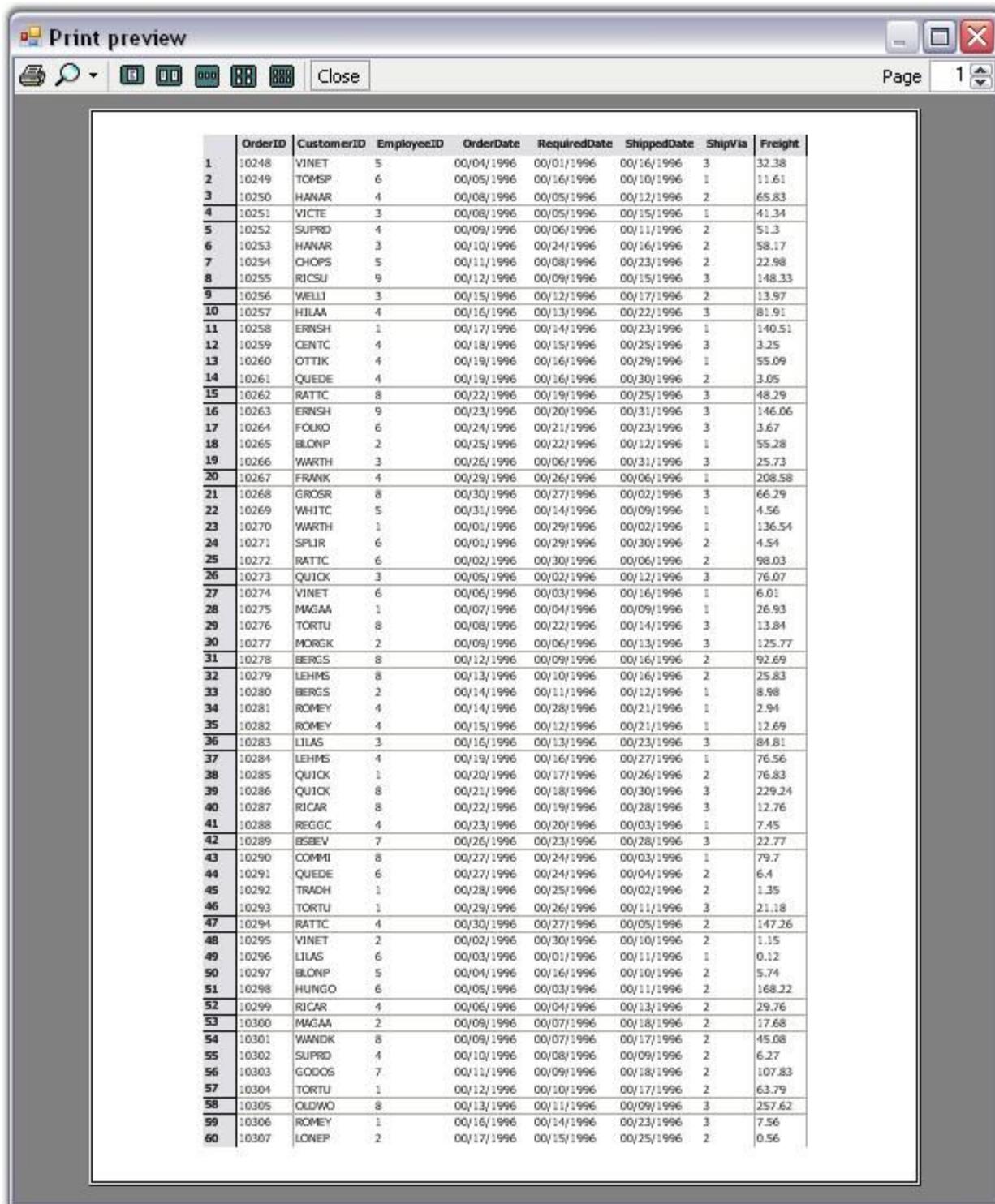


Figure 449: Print Preview of the Grid Control

### Print Page Layout

The Print Page Layout feature enables to view the page layout on the grid by displaying a segment line and a page number with each segment. This helps users to analyze page breaks within the grid, and manage them accordingly.

Properties are available to define colors for the line and text of the page layout. The following code examples illustrate how to set the line and text color of the page layout.

1. Using C#

[C#]

```
LayoutSupportHelper layoutHelper;
layoutHelper = new LayoutSupportHelper(gridControl1);
layoutHelper.LineColor= Color.Blue;
layoutHelper.TextColor = Color.Green;
```

2. Using VB.NET

[VB .NET]

```
Dim layoutHelper As LayoutSupportHelper
layoutHelper = New LayoutSupportHelper(gridControl1)
layoutHelper.TextColor = Color.Orange
layoutHelper.LineColor = Color.SteelBlue
```

Following screen shot illustrates the print page layout feature of the Grid control.

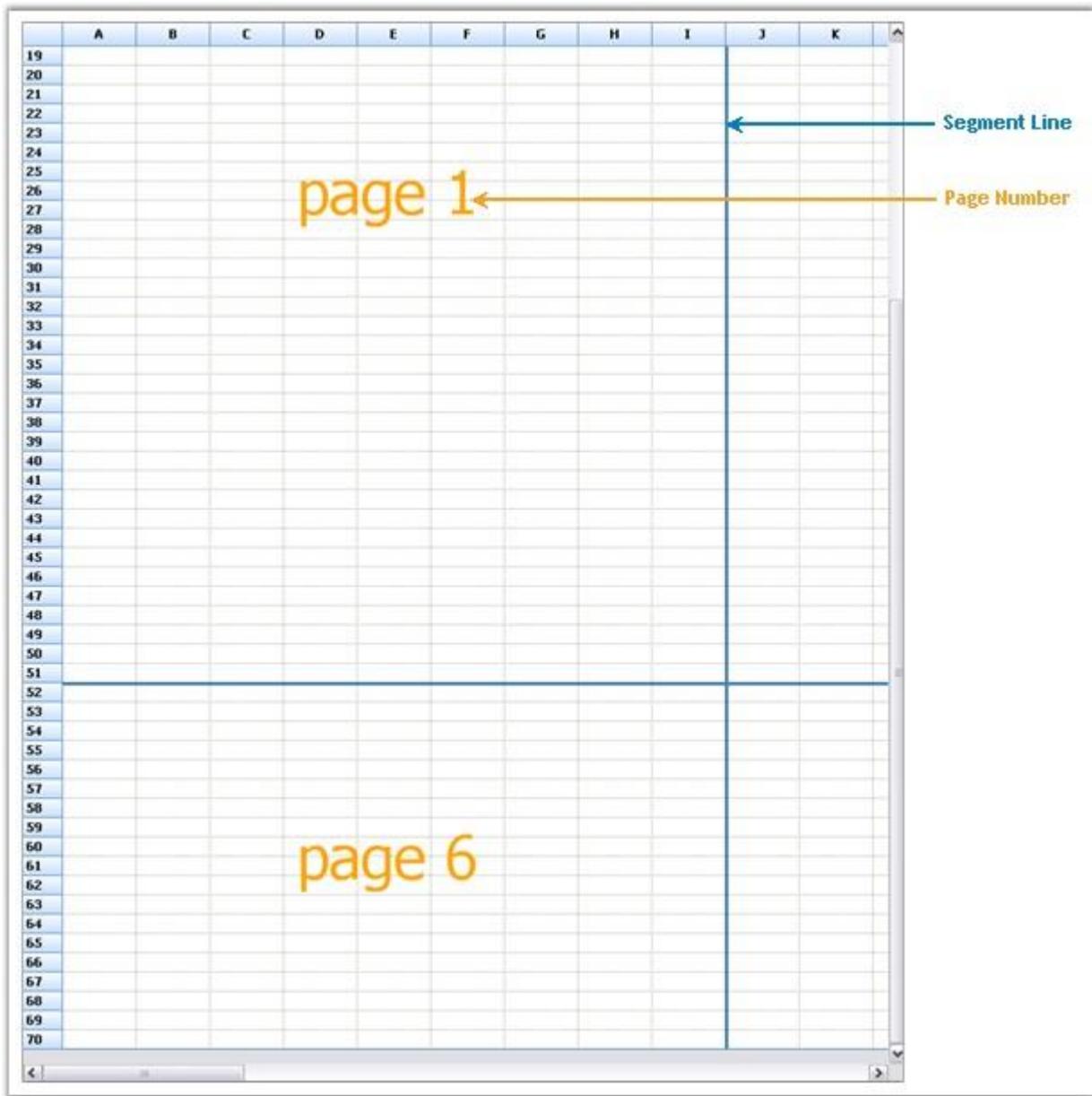


Figure 450: Page Layout of Grid Control

#### 4.7.10 Word Converter

This topic illustrates how to convert Grid and Grid Grouping content to Word format.

##### Grid to Word Conversion

The **GridWordConverter** class provides support to convert grid content into a Word document. It also provides support to add headers and footers to the document.

Essential DocIO libraries are used to support the conversion of grid content into a Word document. The following dependent assemblies must be included in your Windows application to work with the GridWordConverter helper class: **Syncfusion.DocIO.Base** and **Syncfusion.GridHelperClasses.Windows**.

The following code examples illustrate the conversion of Grid content to Word document.

### 1. Using C#

[C#]

```
GridWordConverter converter = new GridWordConverter(true, true);
converter.DrawHeader+=new
GridWordConverterBase.DrawDocHeaderFooterEventHandler(converter_DrawHea
der);
converter.DrawFooter+=new
GridWordConverterBase.DrawDocHeaderFooterEventHandler(converter_DrawFoo
ter);
converter.GridToWord("Sample.doc", gridControl1);
System.Diagnostics.Process.Start("Sample.doc");

void converter_DrawFooter(object sender, DocHeaderFooterEventArgs e)
{
    e.Footer.AddParagraph().AppendText("Copyright 2001-2008");
}

void converter_DrawHeader(object sender, DocHeaderFooterEventArgs e)
{
    e.Header.AddParagraph().AppendText("Syncfusion Inc.");
}
```

### 2. Using VB.NET

[VB .NET]

```
Private converter As New GridWordConverter(True, True)
Private converter.DrawHeader+= New
GridWordConverterBase.DrawDocHeaderFooterEventHandler(AddressOf
converter_DrawHeader)
Private converter.DrawFooter+= New
GridWordConverterBase.DrawDocHeaderFooterEventHandler(AddressOf
```

```
converter_DrawFooter)
converter.GridToWord("Sample.doc", gridControl1)
System.Diagnostics.Process.Start("Sample.doc")

void converter_DrawFooter(Object sender, DocHeaderFooterEventArgs e)
e.Footer.AddParagraph().AppendText("Copyright 2001-2008")

void converter_DrawHeader(Object sender, DocHeaderFooterEventArgs e)
e.Header.AddParagraph().AppendText("Syncfusion Inc.")
```

The following screen shots illustrate Grid to Word conversion.

The screenshot shows a hierarchical grid control with three levels of data:

- Level 1:** parentID, ParentName, ParentDec
- Level 2:** childID, Name
  - Row 0: ChildName0
  - Row 5: ChildName5
    - Level 3:** GrandChildID, Name
      - Row 5: GrandChildName5
      - Row 25: GrandChildName25
      - Row 45: GrandChildName45
    - Row 10: ChildName10
    - Row 15: ChildName15
  - Row 1: parentName1, parentDec1
  - Row 2: parentName2, parentDec2
    - Level 3:** childID, Name
      - Row 2: ChildName2
      - Row 7: ChildName7
      - Row 12: ChildName12
      - Row 17: ChildName17

Figure 451: Grid Control

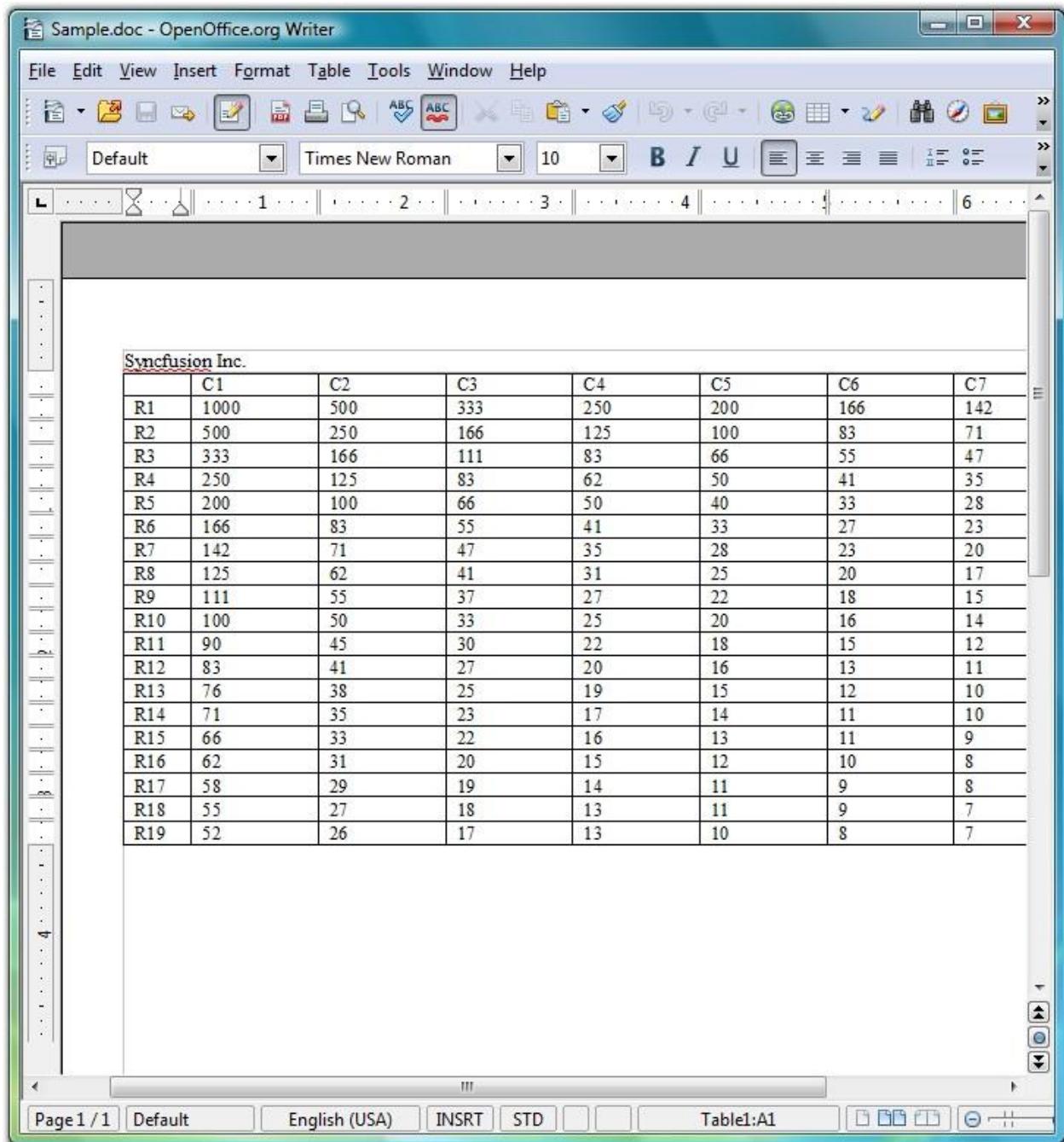


Figure 452: Grid control content converted to Word Document

### Grouping Grid to Word Conversion

The GroupingGridWordConverter class provides support to convert grouping grid content into a Word document. It also provides support to add headers and footers to the document.

Essential DocIO libraries are used to support the conversion of grouping grid content into a Word document. The following dependent assemblies must be included in your Windows application to work with the GroupingGridWordConverter helper class: Syncfusion.DocIO.Base and Syncfusion.GridHelperClasses.Windows.

The following code examples illustrate the conversion of Grouping Grid content to Word document.

## 1. Using C#

[C#]

```
GroupingGridWordConverter converter = new
GroupingGridWordConverter(true, true);
converter.DrawHeader += new
GridWordConverterBase.DrawDocHeaderFooterEventHandler(converter_DrawHeader);
converter.DrawFooter += new
GridWordConverterBase.DrawDocHeaderFooterEventHandler(converter_DrawFooter);
converter.GroupingGridToWord("Sample.doc", gridGroupingControl1);
System.Diagnostics.Process.Start("Sample.doc");

void converter_DrawFooter(object sender, DocHeaderFooterEventArgs e)
{
    IWTextRange txt =
e.Footer.AddParagraph().AppendText("\t\t\tCopyright Syncfusion Inc.
2001 - 2008");
    txt.CharacterFormat.Font = new Font("verdana", 12f,
FontStyle.Bold);
}

void converter_DrawHeader(object sender, DocHeaderFooterEventArgs e)
{
    IWTextRange txt =
e.Header.AddParagraph().AppendText("\t\t\tSyncfusion Inc.\n");
    txt.CharacterFormat.Font = new Font("verdana", 12f,
FontStyle.Bold);
}
```

## 2. Using VB.NET

[VB.NET]

```
Private converter As New GroupingGridWordConverter(True, True)
Private converter.DrawHeader += New
GridWordConverterBase.DrawDocHeaderFooterEventHandler(AddressOf
converter_DrawHeader)
```

```

Private converter.DrawFooter += New
GridWordConverterBase.DrawDocHeaderFooterEventHandler(AddressOf
converter_DrawFooter)
converter.GroupingGridToWord("Sample.doc", gridGroupingControl1)
System.Diagnostics.Process.Start("Sample.doc")

void converter_DrawFooter(Object sender, DocHeaderFooterEventArgs e)
Dim txt As IWTextRange =
e.Footer.AddParagraph().AppendText(Constants.vbTab + Constants.vbTab +
Constants.vbTab & "Copyright Syncfusion Inc. 2001 - 2008")
txt.CharacterFormat.Font = New Font("verdana", 12.0F, FontStyle.Bold)

void converter_DrawHeader(Object sender, DocHeaderFooterEventArgs e)
Dim txt As IWTextRange =
e.Header.AddParagraph().AppendText(Constants.vbTab + Constants.vbTab +
Constants.vbTab + Constants.vbTab & "Syncfusion Inc." & Constants.vbLf)
txt.CharacterFormat.Font = New Font("verdana", 12.0F, FontStyle.Bold)

```

The following screen shots illustrate Grouping Grid to Word conversion.

The screenshot shows a Grouping Grid Control displaying a hierarchical dataset. The data is organized into three main groups, each represented by a row in the first table:

	parentID	ParentName	ParentDec																																														
0	parentName0	parentDec0	<table border="1"> <thead> <tr> <th></th> <th>childID</th> <th>Name</th> </tr> </thead> <tbody> <tr> <td>+ 0</td> <td>ChildName0</td> <td></td> </tr> <tr> <td>- 5</td> <td>ChildName5</td> <td> <table border="1"> <thead> <tr> <th></th> <th>GrandChildID</th> <th>Name</th> </tr> </thead> <tbody> <tr> <td>5</td> <td>GrandChildName5</td> <td></td> </tr> <tr> <td>25</td> <td>GrandChildName25</td> <td></td> </tr> <tr> <td>45</td> <td>GrandChildName45</td> <td></td> </tr> <tr> <td>+ 10</td> <td>ChildName10</td> <td></td> </tr> <tr> <td>+ 15</td> <td>ChildName15</td> <td></td> </tr> </tbody> </table> </td> </tr> <tr> <td>1</td> <td>parentName1</td> <td>parentDec1</td> <td> <table border="1"> <thead> <tr> <th></th> <th>childID</th> <th>Name</th> </tr> </thead> <tbody> <tr> <td>+ 2</td> <td>ChildName2</td> <td></td> </tr> <tr> <td>+ 7</td> <td>ChildName7</td> <td></td> </tr> <tr> <td>+ 12</td> <td>ChildName12</td> <td></td> </tr> <tr> <td>+ 17</td> <td>ChildName17</td> <td></td> </tr> </tbody> </table> </td> </tr> </tbody> </table>		childID	Name	+ 0	ChildName0		- 5	ChildName5	<table border="1"> <thead> <tr> <th></th> <th>GrandChildID</th> <th>Name</th> </tr> </thead> <tbody> <tr> <td>5</td> <td>GrandChildName5</td> <td></td> </tr> <tr> <td>25</td> <td>GrandChildName25</td> <td></td> </tr> <tr> <td>45</td> <td>GrandChildName45</td> <td></td> </tr> <tr> <td>+ 10</td> <td>ChildName10</td> <td></td> </tr> <tr> <td>+ 15</td> <td>ChildName15</td> <td></td> </tr> </tbody> </table>		GrandChildID	Name	5	GrandChildName5		25	GrandChildName25		45	GrandChildName45		+ 10	ChildName10		+ 15	ChildName15		1	parentName1	parentDec1	<table border="1"> <thead> <tr> <th></th> <th>childID</th> <th>Name</th> </tr> </thead> <tbody> <tr> <td>+ 2</td> <td>ChildName2</td> <td></td> </tr> <tr> <td>+ 7</td> <td>ChildName7</td> <td></td> </tr> <tr> <td>+ 12</td> <td>ChildName12</td> <td></td> </tr> <tr> <td>+ 17</td> <td>ChildName17</td> <td></td> </tr> </tbody> </table>		childID	Name	+ 2	ChildName2		+ 7	ChildName7		+ 12	ChildName12		+ 17	ChildName17	
	childID	Name																																															
+ 0	ChildName0																																																
- 5	ChildName5	<table border="1"> <thead> <tr> <th></th> <th>GrandChildID</th> <th>Name</th> </tr> </thead> <tbody> <tr> <td>5</td> <td>GrandChildName5</td> <td></td> </tr> <tr> <td>25</td> <td>GrandChildName25</td> <td></td> </tr> <tr> <td>45</td> <td>GrandChildName45</td> <td></td> </tr> <tr> <td>+ 10</td> <td>ChildName10</td> <td></td> </tr> <tr> <td>+ 15</td> <td>ChildName15</td> <td></td> </tr> </tbody> </table>		GrandChildID	Name	5	GrandChildName5		25	GrandChildName25		45	GrandChildName45		+ 10	ChildName10		+ 15	ChildName15																														
	GrandChildID	Name																																															
5	GrandChildName5																																																
25	GrandChildName25																																																
45	GrandChildName45																																																
+ 10	ChildName10																																																
+ 15	ChildName15																																																
1	parentName1	parentDec1	<table border="1"> <thead> <tr> <th></th> <th>childID</th> <th>Name</th> </tr> </thead> <tbody> <tr> <td>+ 2</td> <td>ChildName2</td> <td></td> </tr> <tr> <td>+ 7</td> <td>ChildName7</td> <td></td> </tr> <tr> <td>+ 12</td> <td>ChildName12</td> <td></td> </tr> <tr> <td>+ 17</td> <td>ChildName17</td> <td></td> </tr> </tbody> </table>		childID	Name	+ 2	ChildName2		+ 7	ChildName7		+ 12	ChildName12		+ 17	ChildName17																																
	childID	Name																																															
+ 2	ChildName2																																																
+ 7	ChildName7																																																
+ 12	ChildName12																																																
+ 17	ChildName17																																																

Figure 453: Grouping Grid Control

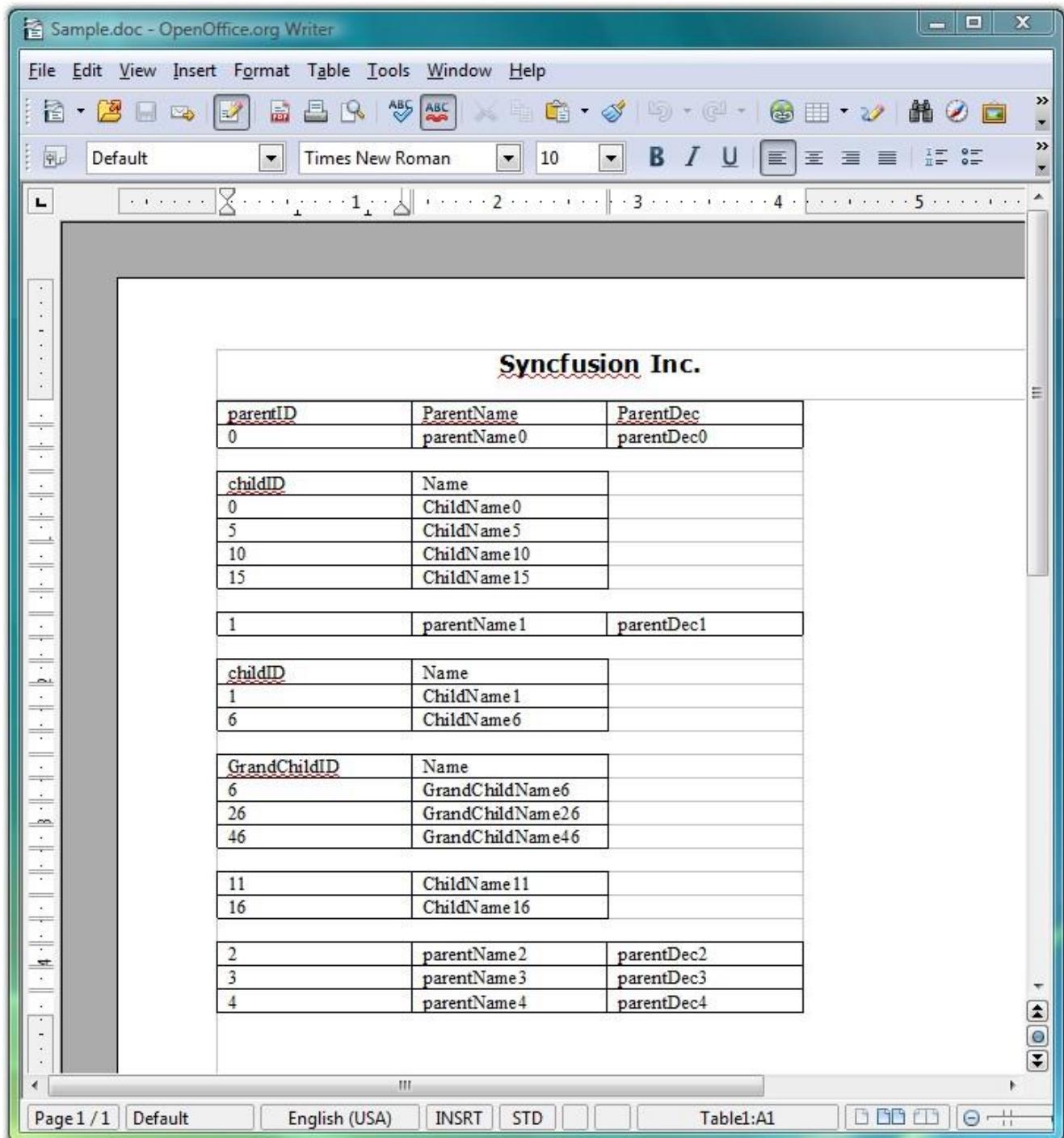


Figure 454: Grouping Grid control content converted to Word Document

## 4.7.11 Fill Series

### Support to implement Excel-like Fill Series in the Grid

A helper class implementing IMouseController interface has been added to GridHelperClasses library to implement Excel-like Fill Series in the Grid.

To make use of this functionality, Syncfusion.GridHelperClasses.Windows.dll must be referred and the mouse controller has to be added in MouseControllerDispatcher of grid.

The following support has been provided since 8.2

The behavior has extended support which pops up a menu after the drag that has two items:

- **Copy Series** - Copy paste the content from the cell.
- **Fill Series** - Fill the cell with appropriate sequence.

The Excel Like fill Series has support on:

- **Number** - From active range with single or multiple cells (e.g. 1, 2, 3...)
- **Text** - Will paste the same text for both 'copy series' and 'fill series'
- **Date** - Date format must be MM/DD/YYYY
- **Month** - The month in text (e.g. January, February, March... or Jan, Feb, Mar...)

The following code illustrates how to add Excel Like fill Series.

[C#]

```
gridControl1.ExcelLikeCurrentCell = true;

Syncfusion.GridHelperClasses.ExcelSelectionMarkerMouseController
marker = new

Syncfusion.GridHelperClasses.ExcelSelectionMarkerMouseController(this.
gridControl1);
this.gridControl1.MouseControllerDispatcher.Add(marker);
```

[VB]

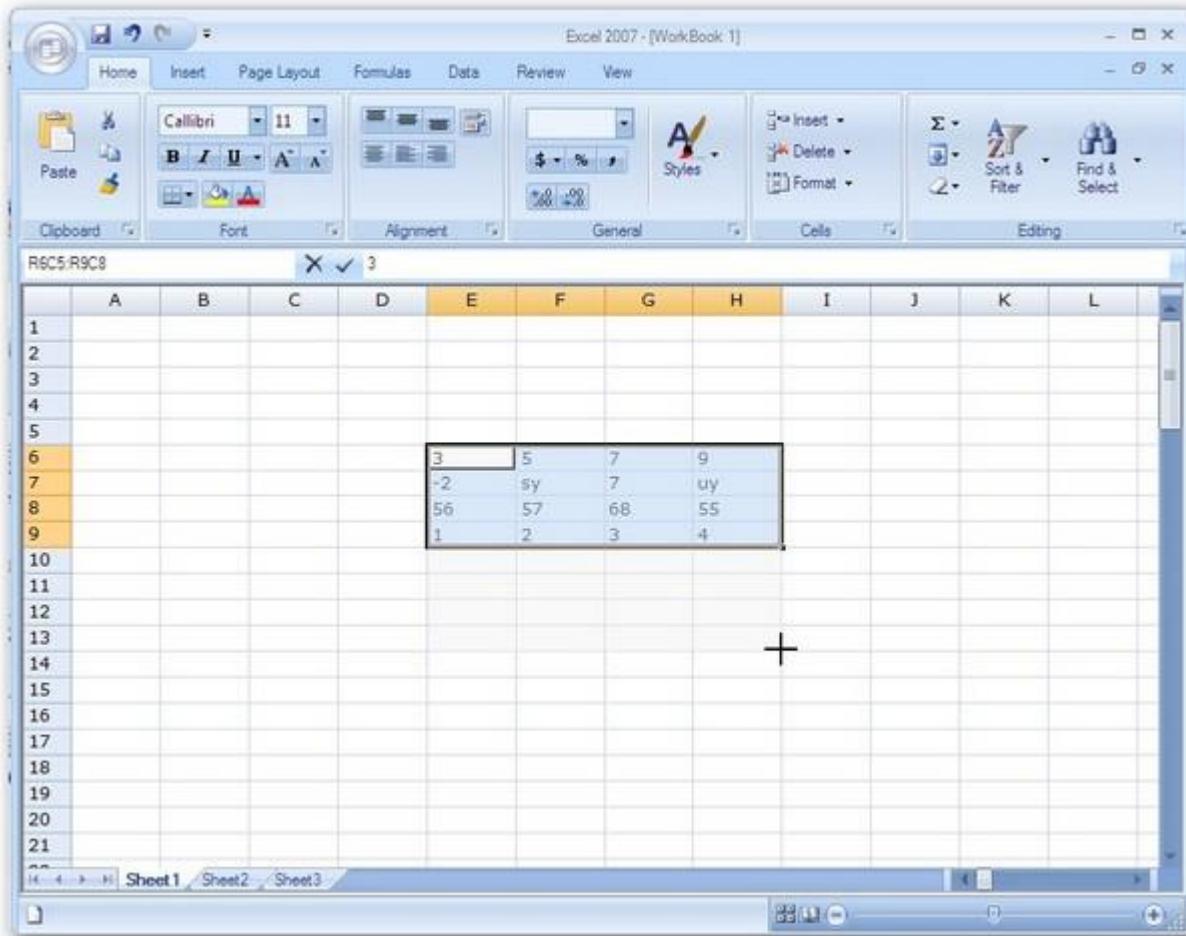
```
GridControl1.ExcelLikeCurrentCell = True
Dim excelMarker As New
ExcelMarkerMouseController(GridControl1)
GridControl1.MouseControllerDispatcher.Add(excelMarker)
```

### Methods of IMouseController Interface Implemented

- **MouseMove**-The code handled in this method allows dragging the series in either one of the four directions at a time, retaining a rectangular layout.
- **MouseUp**-The code handled in this method sets the cell values based on the dragged series accordingly (if it is a formula or text or numeric value).

Following are screen shots illustrating the feature.

1. Image displaying drag operation of the selected series towards bottom.



*Figure 455: Drag Operation of the Selected Series*

2. Image displaying the filled series.

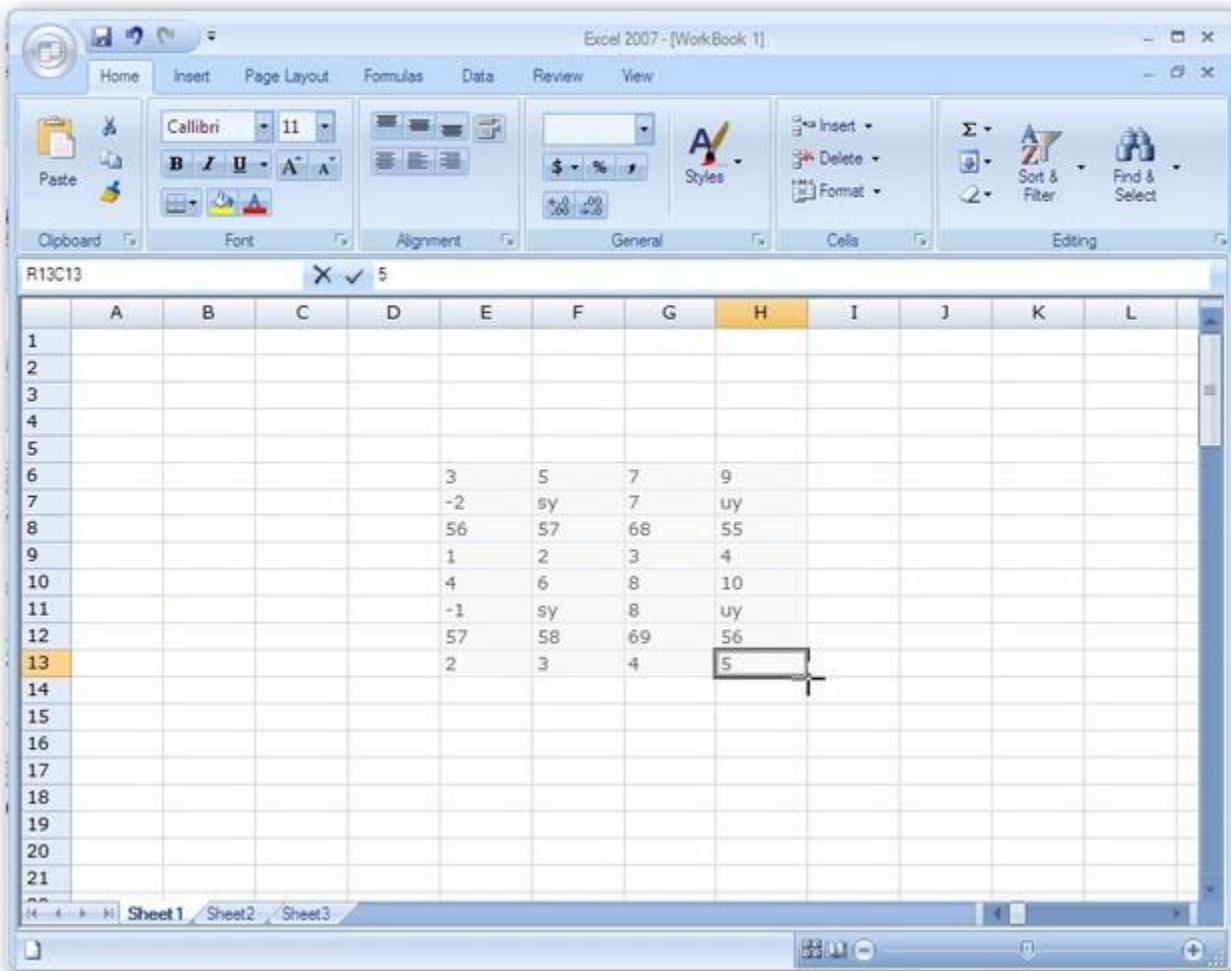


Figure 456: Filled Series

3. The image shows the popup menu displayed after dragging the cell that displays **January**.

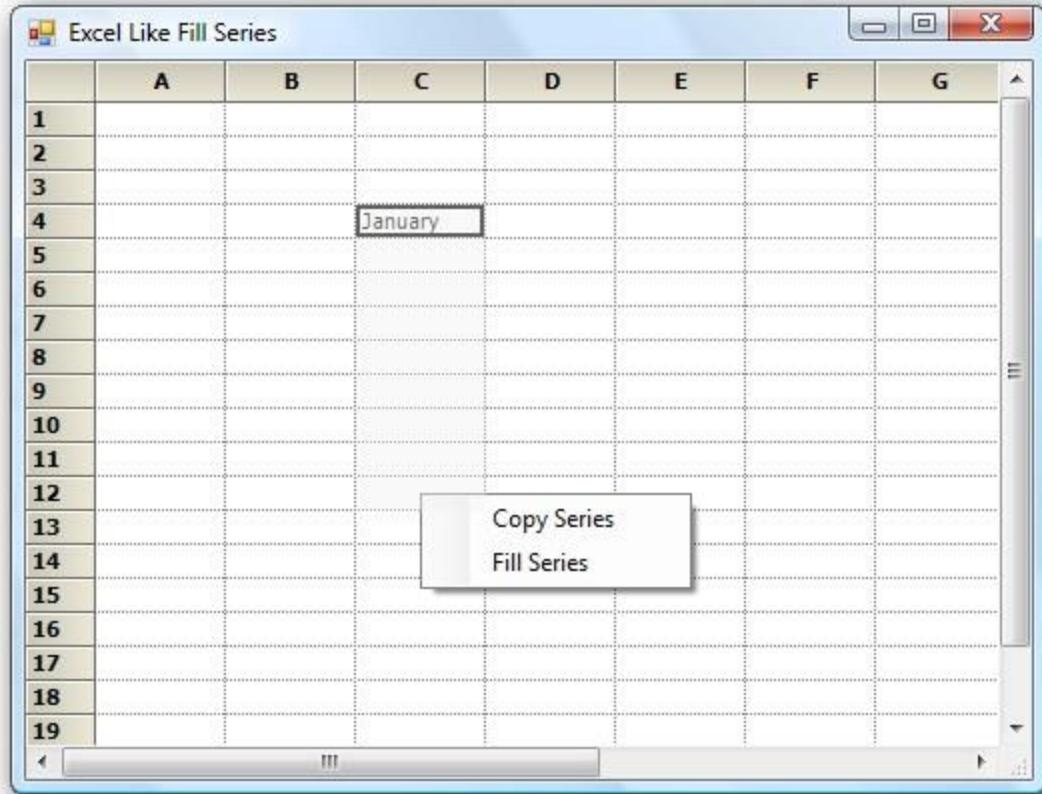


Figure 457: Popup Menu Displayed after Dragging the Cell



**Note:** The cell has been dragged exactly the same as it is done in Excel.

4. The image shows cells have been filled after the Fill series has been selected from the popup menu.

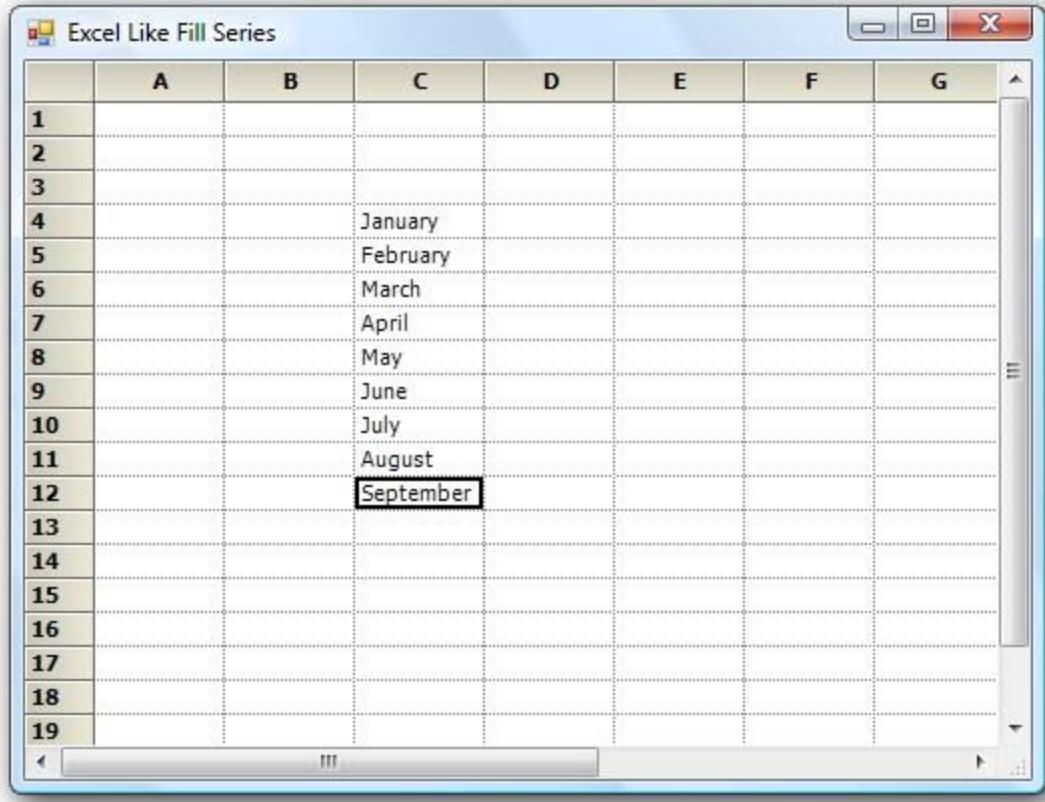


Figure 458: Fill Series

#### 4.7.12 Excel-Like Filters

The **Office2007Filter** and optimized **GridExcelFilter** are the two types of Excel-like filter. The next sections explain these types in detail.

##### 4.7.12.1 Office2007Filter

Essential Grid now provides an in-built filter similar to Microsoft Excel 2007 from the class **GridOffice2007Filter**, with which the grid has to be wired.

###### Enabling Excel like filter

Set **Allow filter** to **true** when Grid Control is wired with the **GridOffice2007Filter** to enable **Excel like filter** to the **Grid filter bar**.

The following code illustrates how to add **Excel Like Filter** to the **Grid filter bar**.

```
[C#]
```

```
        GridOffice2007Filter filter;
        private void showFilter_CheckedChanged(object sender,
EventArgs e)
    {

this.gridGroupingControl1.TableDescriptor.Columns[0].AllowFilter =
true;
        if (this.showFilter.Checked)
    {
        filter.WireGrid(this.gridGroupingControl1);
    }
        else
    {
        filter.UnWireGrid(this.gridGroupingControl1);
    }
    }
}
```

[VB]

```
Private filter As GridOffice2007Filter
Private Sub showFilter_CheckedChanged(ByVal sender As Object,
ByVal e As EventArgs)
    Me.gridGroupingControl1.TableDescriptor.Columns(0).AllowFilter
= True
    If Me.showFilter.Checked Then
        filter.WireGrid(Me.gridGroupingControl1)
    Else
        filter.UnWireGrid(Me.gridGroupingControl1)
    End If
End Sub
```



**Note:** The *GridOffice2007Filter* can unwire from the grid to disable the Excel like filter.

### Specifying Value To Filter

The feature has multiple selections of values to filter.

You can specify the value the column has to filter in the check box in the tree view inside the drop down container.

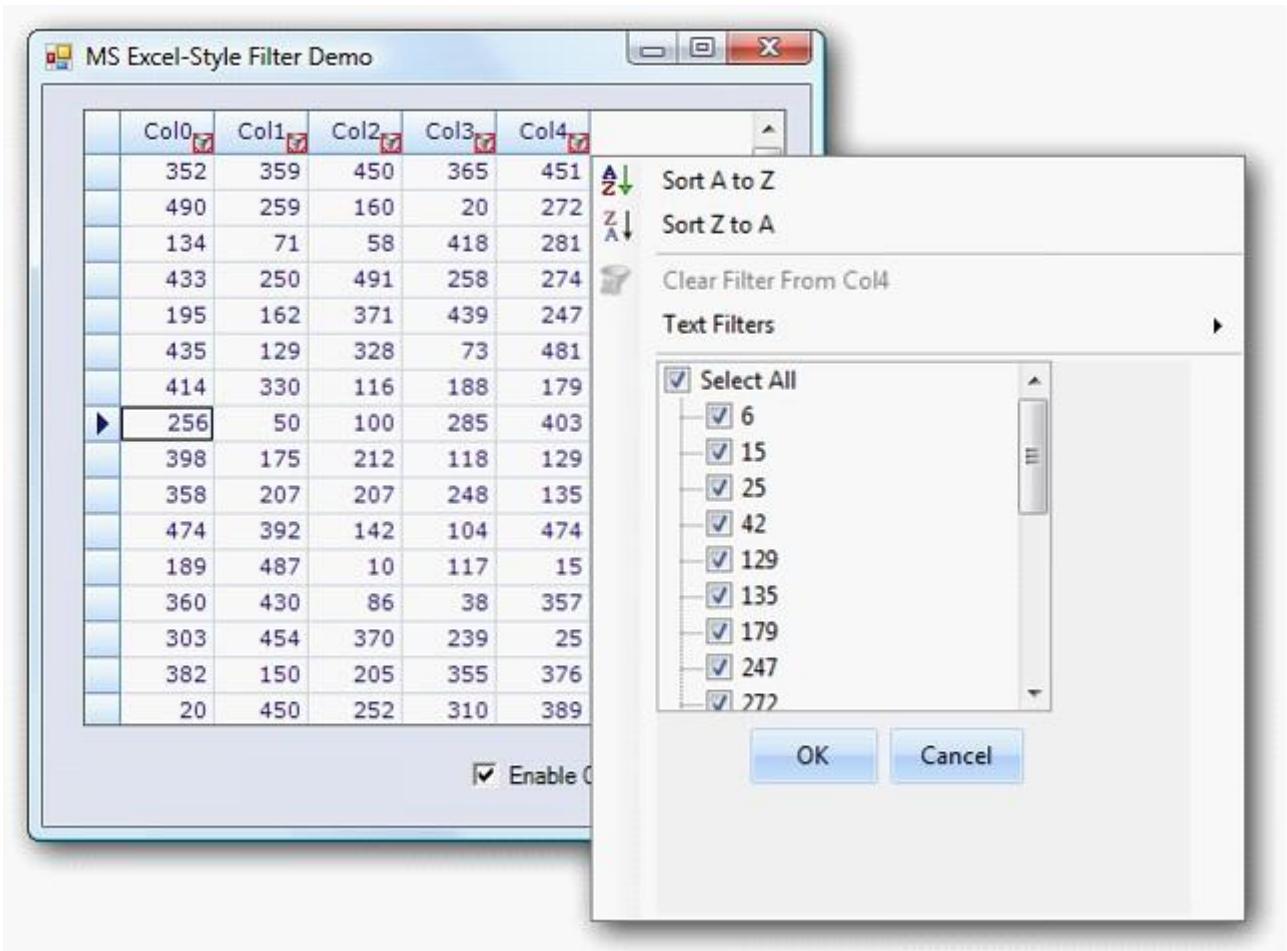


Figure 459: Filter Bar Drop Down

#### 4.7.12.2 Optimized GridExcelFilter

##### Use Case Scenarios

If the Office2007Filter is used in a WF GridGroupingControl where the columns have a large number of unique items (say 5000 or 10000 unique items), the grid is unusable (hanged). To improve the performance, the new optimized GridExcelFilter can be used.

##### Methods

Method	Description	Parameters	Type	Return Type
WireGrid	Wires grid with filter.	this.gridGroupingControl1 (control as argument)	Method	void

UnWireGrid	Unwire grid with filter.	this.gridGroupingControl1 (control as argument)	Method	void
------------	--------------------------	---	--------	------

**Sample Link**

To view a sample:

- Open **Syncfusion Dashboard**.
- Select **UI > Windows Forms**.
- Click **Run Samples**.
- Navigate to **GridGrouping Samples > Filters and Expressions > Optimized Excel Filter Demo**

**Implementing the optimized GridExcelFilter to GGC**

Set **AllowFilter** to **True** when the Grid control is wired with the GridOffice2007Filter to enable Excel-like filtering in the grid filter bar.

The following code illustrates how to add the Excel-like filter to the grid filter bar:

**[C#]**

```
GridExcelFilter filter;

private void showFilter_CheckedChanged(object sender, EventArgs e)
{
    this.gridGroupingControl1.TableDescriptor.Columns[0].AllowFilter = true;
    if (this.showFilter.Checked)
    {
        filter.WireGrid(this.gridGroupingControl1);
    }
    else
    {
        filter.UnWireGrid(this.gridGroupingControl1);
    }
}
```

**[VB]**

```
Private filter As GridExcelFilter
```

```

Private Sub showFilter_CheckedChanged(ByVal sender As Object, ByVal e As EventArgs)
    Me.gridGroupingControl1.TableDescriptor.Columns(0).AllowFilter = True
    If Me.showFilter.Checked Then
        filter.WireGrid(Me.gridGroupingControl1)
    Else
        filter.UnWireGrid(Me.gridGroupingControl1)
    End If
End Sub

```

#### 4.7.12.3 Filtering Null Values from the Grid

This feature is to provide support for filtering null or empty values from the grid using the excel-like filters. The filter choice will be in the name of "(Blanks)" when there are empty values in the grid.

##### Option to Filter Empty values in Application

The option to filter the empty values (Blanks) will automatically be included in the filter choices when there are one or more empty values in the grid. No property is needed to enable this feature.

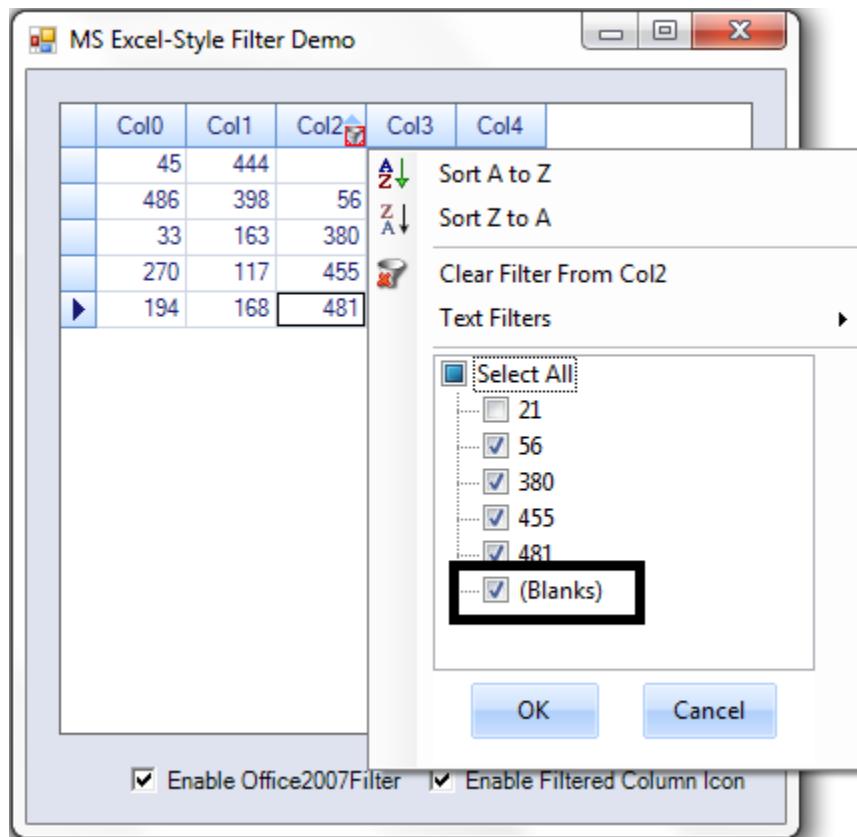


Figure 460: Blanks Option to Filter Empty Values

## 4.7.13 Card View Layout

The card view displays records as separate cards arranged in a grid-like layout. It is implemented similarly to binding the GridDataBoundGrid control to the GridCardView object.

### 4.7.13.1 Tables for Properties, Methods, and Events

#### Properties

Property	Description	Type	Data Type
CaptionField	Sets or gets the caption of each card. Users can set a column name to this.	string	string
CardStyle	Applies a card model such as the standard, merged, variable, or compressed styles.	CardStyle	enum
VisualStyle	Applies the visual style of the cards in the grid. The new GridVisualStyles are applied to this.	CardVisualStyle	enum
CardSpacingWidth	Gets or sets the width of the spacing between each card.	int	int
CardSpacingHeight	Gets or sets the height of the spacing between each card.	int	int
MaxCardCols	Gets or sets the maximum number of cards in the column area. MaxCardRows will be automatically calculated.	int	int
MaxCardRows	Gets or sets the maximum number of cards in the row area. MaxCardCol will be automatically calculated if it isn't set.	int	int
ShowCaption	Shows or hides the caption of each card.	bool	bool

ShowCardCellBorders	Shows or hides the cell borders.	bool	bool
AllowResizing	Allows or prevents the card resizing.	bool	bool
CaptionHeight	Gets or sets the height of the caption row.	bool	bool
ActivateCurrentCellBehavior	Specifies current cell activation behavior when moving the current cell or clicking inside a cell.	GridCellActivateAction	enum
HighlightActiveCard	Gets or sets the highlight of the active card.	bool	bool

### Methods

Method	Description	Parameters	Type	Return Type
WireGrid	Gets the GridDataBoundGrid and changes it to Card View Style.	GridDataBoundGrid	Method	void
UnWireGrid	Unhooks all the events which are hooked in the WireGrid() method.	N/A	Method	Void
IsActiveCard	Indicates the state of the card if active.	rowIndex, CollIndex	Method	bool
IsHeaderCell	Indicates if the cell is a header column cell.	rowIndex, CollIndex	Method	bool
IsRecordCell	Indicates whether the cell is a record cell.	rowIndex, collIndex.	Method	bool

IsValueCell	Indicates whether the cell is a value cell.	rowIndex, colIndex.	Method	bool
IsCardCaption	Indicates whether the cell is a caption cell.	rowIndex, colIndex.	Method	bool
GetCardCellType GetCardCellType	Specifies the type of the card cell.	rowIndex, colIndex.	Method	CardCellType

### Events

Event	Description	Arguments	Type
QueryCardCellInfo	Occurs when the card model queries for style information about a specific cell.	public QueryCardCellInfoEventArgs(GridQueryCellInfoEventArgs e, GridCardView cardView)	Event
CellClick	Occurs when the user clicks inside a cell.	public CardCellClickEventArgs(GridCellClickEventArgs e, GridCardView cardView)	Event
SaveCardCellInfo	Occurs when the card model is about to save style information about a specific cell.	public SaveCardCellInfoEventArgs(GridSaveCellInfoEventArgs e, GridCardView cardView)	Event
PushButtonClick	Occurs when the user clicks a push button.	public CardCellPushButtonEventArgs(GridCellPushButtonEventArgs e, GridCardView cardView)	Event

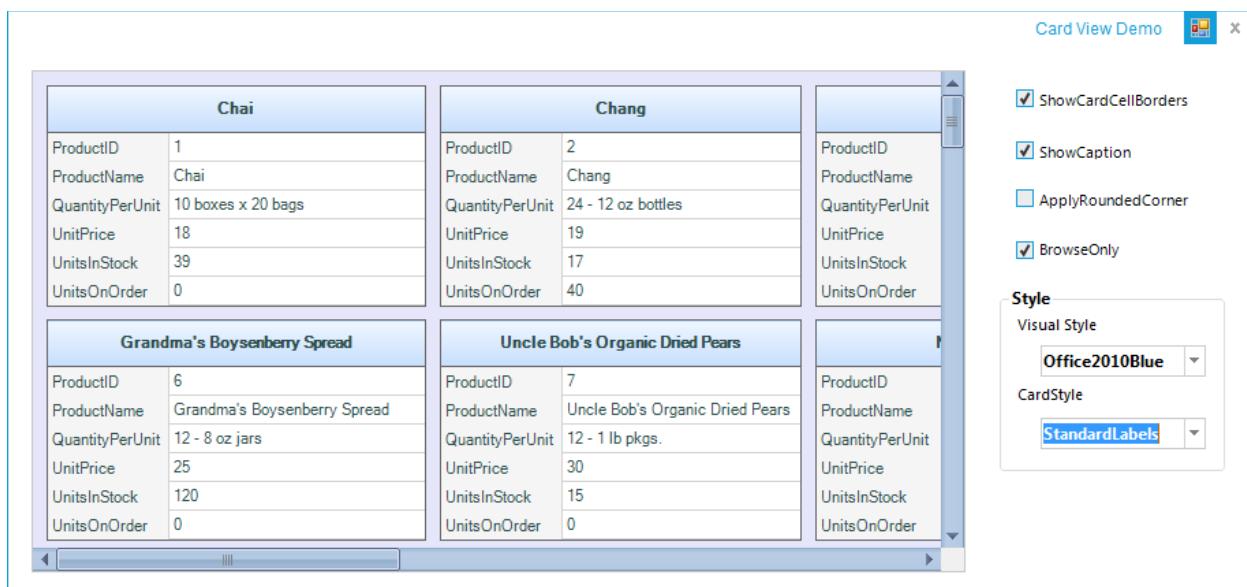


Figure 461: Image for Card View Layout in GridDataBoundGrid

### Sample Link

```
{installed
drive}\AppData\Local\Syncfusion\EssentialStudio\{version}\Windows\GridDataBound.Windows\Sa
mples\2.0\Product Showcase\Card View Demo
```

#### 4.7.13.2 Enable the Card View Layout

The following code is used to enable the card view layout in the GridDataBoundGrid control.

[C#]

```
GridCardView card = new GridCardView();
card.CaptionField = "ProductName";
card.WireGrid(this.gridDataBoundGrid1);
```

[VB]

```
Private card As New GridCardView()
card.CaptionField = "ProductName"
card.WireGrid(Me.gridDataBoundGrid1)
```

## 4.8 Office2010 Theme in Windows Grids

This feature provides support to have MS Office 2010 themes namely Blue, Black, and Silver) for the Windows Grids: GridControl, GridGroupingControl, GridDataBoundGrid, GridListControl, GridRecordNavigationControl and associated scrollbars.

To enable this support in grid, the following need to be handled:

- Apply Office 2010 Visual Style to Grid
- Enable Office 2010 Scrollbars

### **Applying Office2010 Visual Style to Grid**

To apply Office 2010 Visual Style to Grid:

1. Create a grid enabled sample application.
2. Set the **Office2010** theme to grid control using **GridVisualStyles**.

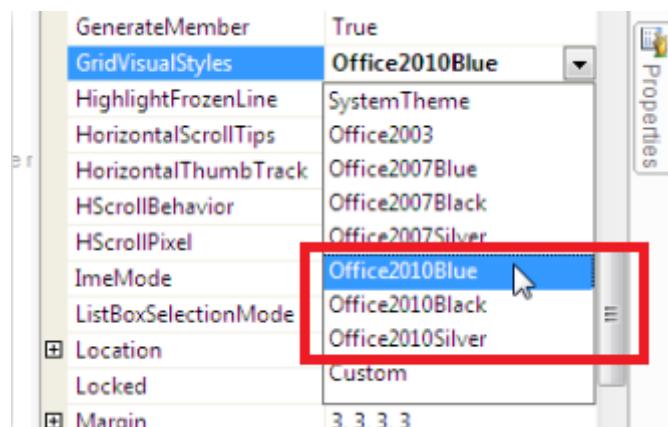


Figure 462: Set *GridVisualStyles* property to enable this theme.

### **Enabling Office2010 Scrollbars**

To enable Office 2010 Scrollbars:

1. Set the **GridOfficeScrollBars** property to Office 2010.
2. Set the **Office2010ScrollBarColorScheme** property to Blue, Black or Silver.

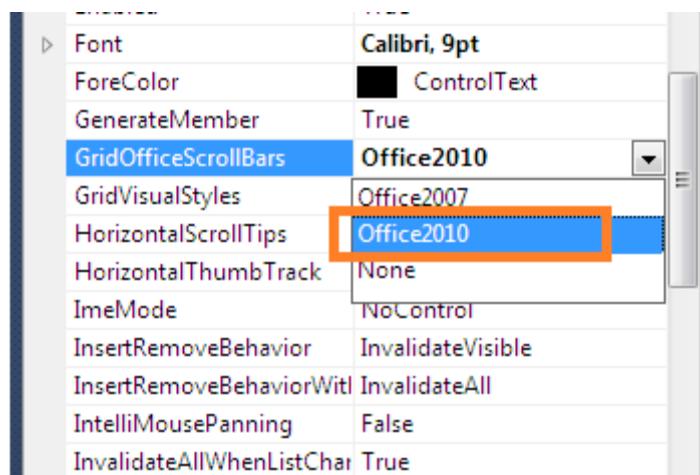


Figure 463: Set *GridOfficeScrollBars* property to enable this theme.

### Use Case Scenarios

Office2010Theme support for Windows Grids is useful for commercial applications in order to attract its users with inspiring UI look and feel.

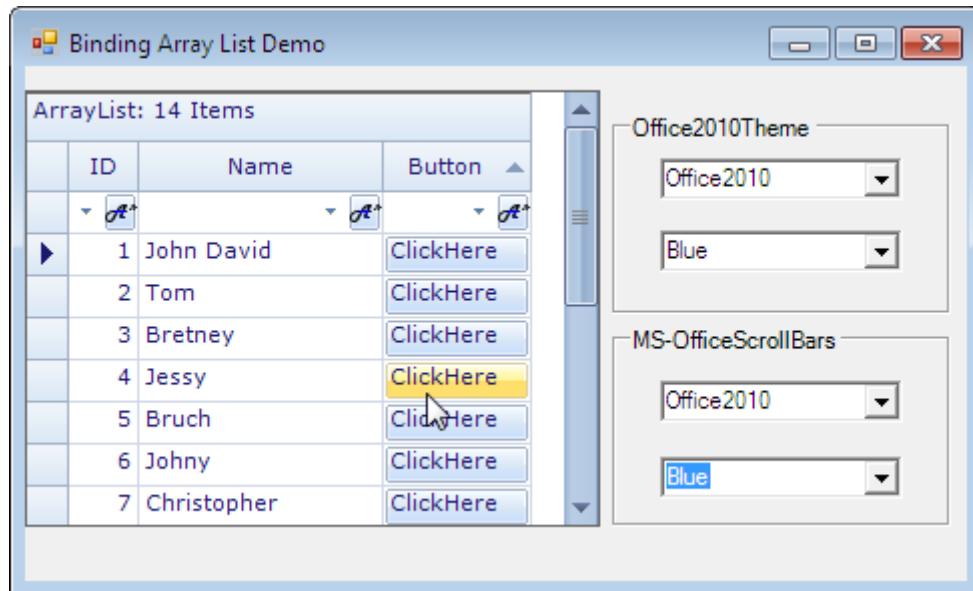


Figure 464: Office2010 Blue theme

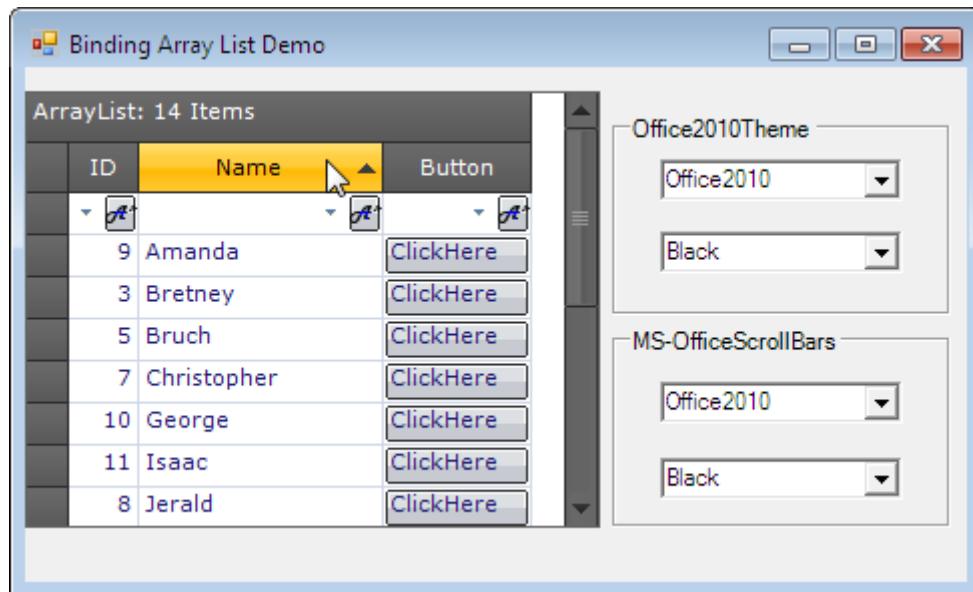


Figure 465: Office2010 Black theme

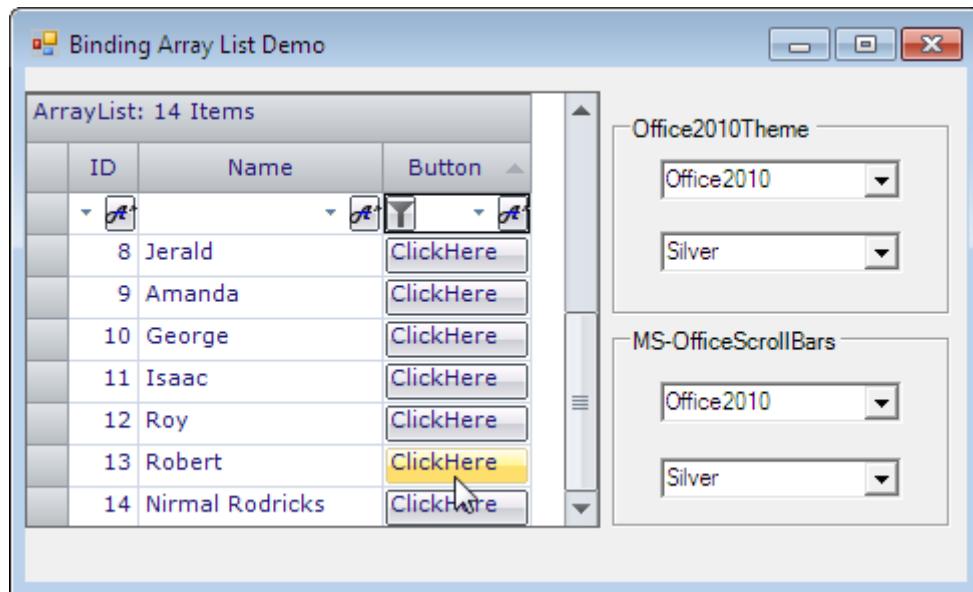


Figure 466: Office2010 Silver theme

## Tables for Properties and Events

### Properties

Table 16: Properties Table

Property	Description	Data Type
GridVisualStyles	<ul style="list-style-type: none"> <li>This is an Enumeration type property.</li> <li>This property is used to get or set the VisualStyles (skins) like Office2010, Office2007, Office2003.</li> </ul>	Syncfusion.Windows.Forms.GridVisualStyles
GridOfficeScrollbars	<ul style="list-style-type: none"> <li>This is an Enumeration type property.</li> <li>This property is used to get or set the Office like scrollbars.</li> </ul>	Syncfusion.Windows.Forms.OfficeScrollBars
Office2010ScrollBar sColorScheme	<ul style="list-style-type: none"> <li>This is an Enumeration type property.</li> <li>This property is used to get or set the style of Office2010 scroll bars</li> </ul>	Syncfusion.Windows.Forms.Office2010ColorScheme

### Events

- The following event is used when applying the Office2010 theme to Essential Windows Grids.

Event	Parameters	Description
ThemeChanged	Object sender, EventArgs e	Occurs when the ThemesEnabled property is changed.

- The following events occur when the GridOfficeScrollBars are applied to Essential Windows Grids.

Event	Parameters	Description
Office2010ScrollBarsColorSchemeChanged	Object sender, EventArgs e	Occurs when the Office2010ScrollBarsColorScheme property has changed.
OfficeScrollBarsChanged	object sender, GridGroupingControl.OfficeScrollBarsEventArgs e	Occurs when the GridOfficeScrollBars property has changed.

### Adding Grid with Office2010 Theme to an Application

To add Grid with Office 2010 theme to an application:

- Create a **GridControl** enabled application.
- Set the **GridVisualStyles** property to apply **Office2010** theme in GridControl.

The following sample code sets an Office2010 Black skin theme to the Essential Grid Control.

[C#]

```
this.gridGroupingControl1.GridVisualStyles =  
GridVisualStyles.Office2010Black;
```

[VB]

```
Me.gridGroupingControl1.GridVisualStyles =  
GridVisualStyles.Office2010Black
```

- Set the **GridOfficeScrollBars** property to **Office2010**, to apply Office2010 like scroll bars in Essential Windows Grids.
- Set the **Office2010ScrollBarsColorScheme**, to apply the color scheme of the scroll bars.

[C#]

```
this.gridGroupingControl1.GridOfficeScrollBars =  
OfficeScrollBars.Office2010;  
this.gridGroupingControl1.Office2010ScrollBarsColorScheme =  
Office2010ColorScheme.Black;
```

[VB]

```
Me.gridGroupingControl1.GridOfficeScrollBars =
```

```
OfficeScrollBars.Office2010  
Me.gridGroupingControl1.Office2010ScrollBarsColorScheme =  
Office2010ColorScheme.Black
```

### **Sample Link**

To get the Schedule samples from the dashboard:

1. Open Essential Studio Dashboard by selecting Start -> All Programs -> Syncfusion -> Essential Studio <> Version Number -> Dashboard.
2. Select “Run Locally Installed Samples” from the Windows Forms drop-down list on the User Interface pane.
3. Expand “Grid samples” in the left panel of sample browser.
4. Expand “Appearance” subsection and select “Grid Style Demo”.
5. Click the “Run Sample” button in the right panel.

To open sample project:

1. Navigate to the following sample location in your system:  
**“<<Sample Installation Location>>\Syncfusion\Essential Studio<>Version Number>>\Windows\Grid.Windows\Samples\2.0\Appearance\Grid Style Demo”**
2. This location contains two sub folders CS and VB. You can open the sample projects from the respective folders based on your application language.

## **4.9 Enhanced Visual Styles for the Syncfusion Windows Grids**

This feature enables you to apply an enhanced visual styles for the Windows Grids: Grid control, GridGrouping control, GridDataBoundGrid, GridList control, GridRecordNavigation control and associated scrollbars, combo box drop-down container, filter (dynamic filter and filter bar), selected row and group drop area.

By default, the **EnableLegacyStyle** property value is true.

### **Use Case Scenarios**

You can use this control to apply skin for the entire control including the scrollbars, combo box, drop-down container, filter and so on.

### **Properties**

*Table 17 Properties Table*

<b>Property</b>	<b>Description</b>	<b>Data Type</b>
EnableLegacyStyle	Specifies whether the enhanced GridVisual style has to be enabled.	Boolean

EnableGridListControlInComboBox	Specifies whether the grid combo box should contain the GridList control.	Boolean
---------------------------------	---	---------

**Sample Link**

Samples for the feature are available in the following locations:

{Installed

Path}\Syncfusion\EssentialStudio\{Version}\Windows\Grid.Grouping.Windows\Samples\2.0\Styling and Formatting\Skin Customization Demo

### 4.9.1 Applying Enhanced GridVisualStyle to the Application

To apply enhanced visual style to the entire control, you have to enable the enhanced GridVisualStyle. You can enable this using the *EnableLegacyStyle* property. To enable the enhanced GridVisualStyle, set *EnableLegacyStyle* to *false*. To enable GridVisualStyle, set *EnableLegacyStyle* to *true*. By default, this is set to *true*.

The following code illustrates how to enable the enhanced GridVisualStyle.

[C#]

```
this.gridGroupingControl1.TableModel.EnableLegacyStyle = false;
```

[VB]

```
Me.gridGroupingControl1.TableModel.EnableLegacyStyle = False
```

#### EnhancedGridVisualStyle

You can apply one of the following skins to the control using the EnhancedGridVisualStyle property:

- Office2003
- Office2007Blue
- Office2007Black
- Office2007Silver
- Office2010Blue
- Office2010Black
- Office2010Silver
- Metro
- SystemTheme

SystemTheme is the default skin.

The following code illustrated how to customize the skin for the control:

[C#]

```
this.gridGroupingControl1.GridVisualStyles =
```

```
GridVisualStyles.Office2007Black;
```

[VB]

```
this.gridGroupingControl1.GridVisualStyles =
GridVisualStyles.Office2007Black
```

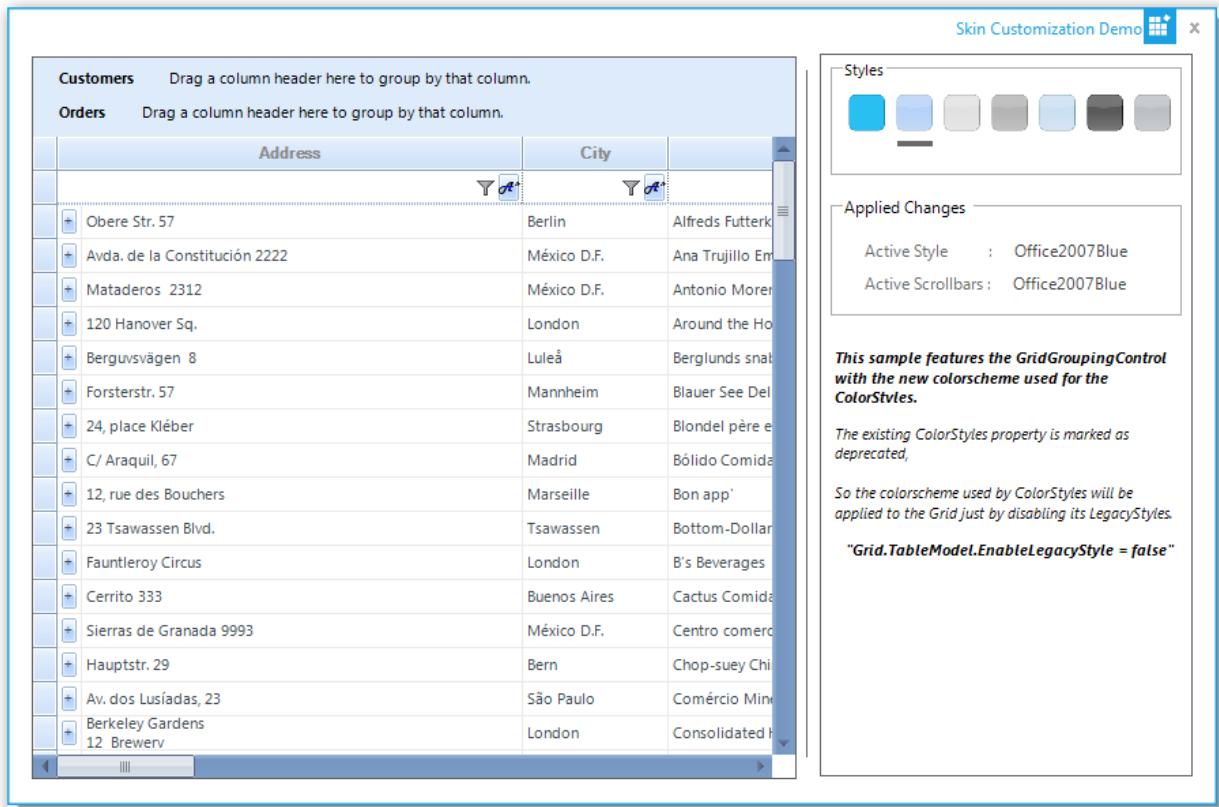


Figure 467 Skin Customization

## 4.9.2 Combo Box

Skin cannot be applied to the ListBox control inside the combo box. To overcome this limitation, Essential Grid uses the GridList control in the combo box. The enhanced GridVisualStyle settings uses the combo box with the *GridDropDownGridListControlCellModel* which is inherited from the *GridComboBoxCellModel* class, instead of using the *GridComboBoxCellModel* directly.

To use the GridList control in the combo box (which enables you to apply styles), set the *EnableGridListControlInComboBox* property to *true*. This is the default value.

The following code illustrated this:

[C#]

```
this.gridControl1.Model.EnableGridListControlInComboBox = true;
```

[VB]

```
Me.gridControl1.Model.EnableGridListControlInComboBox = true
```

If you want to create an application for which the combo box is created from the GridComboBoxCellModel and not to use the GridDropDownGridListControlCellModel, you need to set the *EnableGridListControlInCobmoBox* property to *false*. By default, this is set to *true*.

The following code illustrates how to disable the *EnableGridListControlInCobmoBox* property:

[C#]

```
this.gridControl1.Model.EnableGridListControlInComboBox = false;
```

[VB]

```
this.gridControl1.Model.EnableGridListControlInComboBox = false;
```

## 4.10 Metro Theme for Essential Grid Controls

This feature enables you to apply new Metro styles to the controls of Essential Grid for Windows Forms. It provides support for the following controls and its associated scroll bars:

- Grid control
- GridGrouping control
- GridDataBound control
- GridList control
- GridRecordNavigation control

### Use case scenarios

The Metro theme support is useful for commercial applications in order to attract end users with inspiring UI look and feel.

#### 4.10.1 Tables for Properties and Events

##### Properties

Property	Description
GridVisualStyles	This is an enumeration type property. It is used to get or set the visual styles (skins) such as Office2010, Office2007, Office2003, Metro, etc.

## Events

Event	Parameters	Description
ThemeChanged	Object sender, EventArgs e	Occurs when the <b>ThemesEnabled</b> property is changed.

## 4.10.2 Applying Metro Theme to a Control

You can apply Metro theme to an application by setting the **GridVisualStyles** property as **Metro**. The following code example illustrates how to apply the Metro theme for the **GridGrouping** control.

[C#]

```
this.gridGroupingControl1.GridVisualStyles = GridVisualStyles.Metro;
```

[VB]

```
Me.gridGroupingControl1.GridVisualStyles = GridVisualStyles.Metro
```

The following screenshot is a sample output for the previous code.

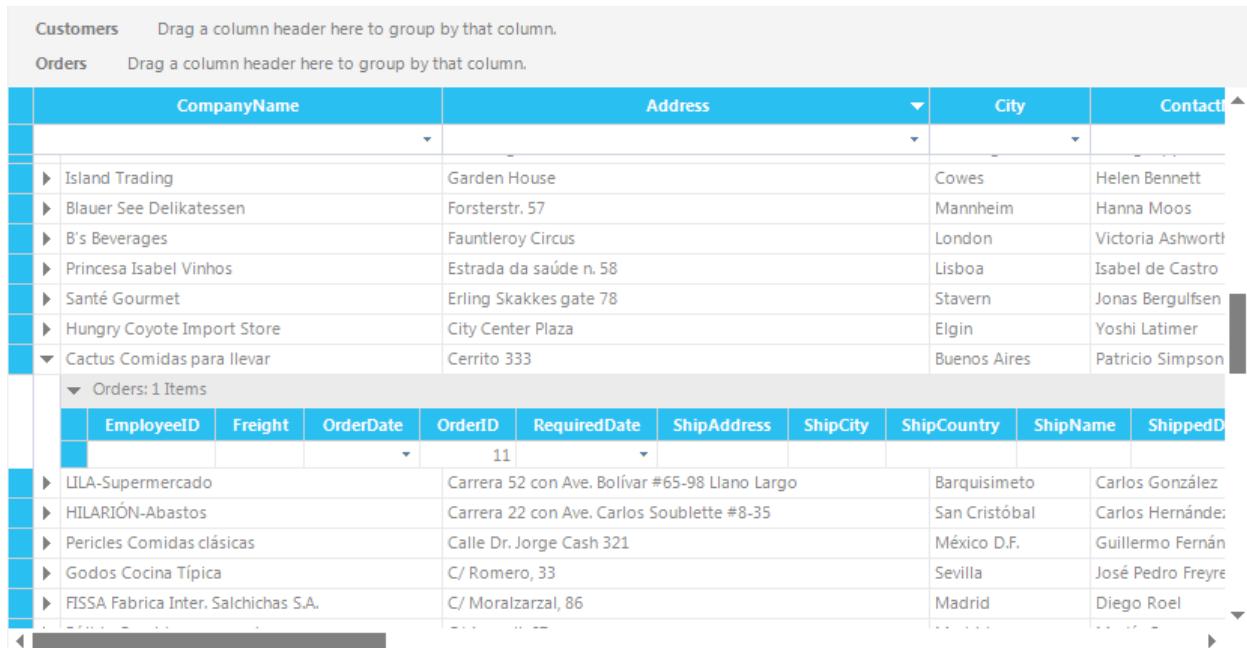


Figure 468: GridGrouping Control Applied with Metro Theme

### 4.10.3 Applying User-Defined Colors as Metro Themes

Users can now utilize the SetMetroStyle method to set user-defined colors as the Metro theme in a grid.

#### Method Table

Method	Prototype	Description
SetMetroStyle	<code>public void SetMetroStyle(     Color metroColor,     Color metroHoverColor,     Color metroColorPressed);</code>	A method that takes the user-defined back color, mouse hover color and mouse pressed color to apply to grid.

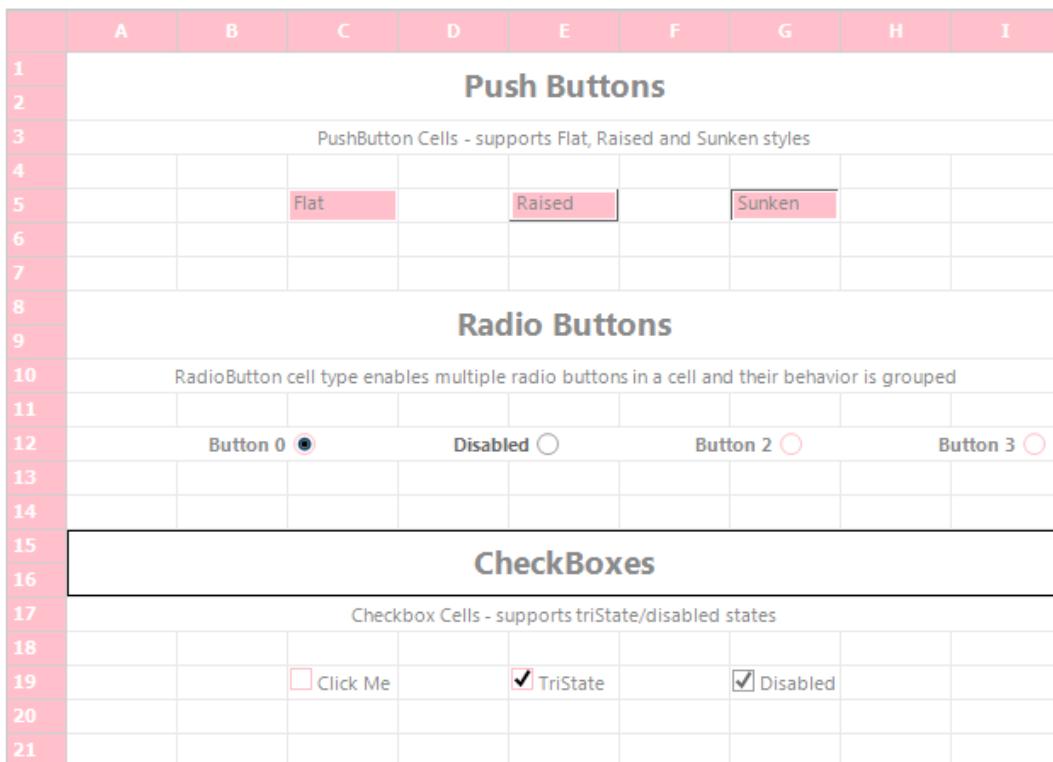


Figure 469: Grid Control with Customized Metro Color (Pink)

	A	B	C	D	E	F	G	H	I
1									
2									
<b>DropDown Cells</b>									
3	<b>ComboBox Cells :</b> Following Categories illustrates the features supported by a ComboBox cell								
4	<input type="checkbox"/> ComboBox with AutoComplete in Editmode								
9									
10	<input type="checkbox"/> ComboBox using ChoiceList								
20									
21	<input type="checkbox"/> ComboBox using Datasource, DisplayMember = CompanyName								
31									
32	<input type="checkbox"/> ComboBox using Datasource, ValueMember = empty								
42									
43	<b>GridListControl Cells :</b> Following Categories illustrates the features supported by a GridListControl cell								
44	<b>GridListControl using ChoiceList :</b> Not Supported								
45									
46	<input type="checkbox"/> GridListControl using Datasource, DisplayMember = CompanyName								
56									
57	<input type="checkbox"/> GridListControl using Datasource, ValueMember = empty								
67									
68	<input type="checkbox"/> GridListControl showing ArrayList with Icons								
78									
79	<b>DropDown Calendar Cells</b>				<b>DropDown ColorEdit Cells</b>				
80	3/14/20	▼	3/14/2013 3:23:31 AM	▼		@		@	
81									
82	<input type="checkbox"/> Rich Text Cells : Check this to view a drop-down menu which displays formatting helpers to modify the text								
84									

Figure 470: Grid Control with Customized Metro Color (Green)

Customers Drag a column header here to group by that column.

	Address			City	
▶	Obere Str. 57			Berlin	
▼	Avda. de la Constitución 2222			México D.F.	
Orders: 4 Items					
	EmployeeID	Freight	OrderDate	▲	OrderID
*	7	1.61	10/19/1994 12:00:00 AM	▼	10308 11/16/1994
	3	43.9	9/8/1995 12:00:00 AM	▼	10625 10/6/1995
	3	11.99	12/29/1995 12:00:00 AM	▼	10759 1/26/1996
	4	39.92	4/3/1996 12:00:00 AM	▼	10926 5/1/1996

**Nested Table Group Options**

- Show Caption +/-
- Show Caption
- Modify Caption Text

**Add New Record Field**

- Before Details
- After Details

Changed NestedTableGroupOptions  
Property :

Changed Property :

Figure 471: GridGrouping Control with Customized Metro Color (Pink)

[C#]

```
this.gridControl1.SetMetroStyle(Color.Pink, Color.Plum,
Color.PowderBlue);
```

[VB]

```
Me.gridControl1.SetMetroStyle(Color.Pink, Color.Plum, Color.PowderBlue)
```

## 4.11 Coded UI Support in Windows Grids

Essential Grid for Windows Forms now supports automated UI testing with VS 2010 Coded UI technology. The Grid Test plugin blends in with the automated UI testing framework in VS 2010 by implementing the following classes:

- UITechnologyManager
- UITestPropertyProvider
- UIActionFilter

The architectural diagram is as follows:



Figure 472: Architectural Diagram

- The Grid Test Plugin implements the necessary details to communicate with the VS 2010 Test Framework.
- The Grid application host runs with a .NET Remoting channel hosted internally to communicate with the test plugin through an interface. The data is then channeled across the VS 2010 Test Framework, to identify the Cells and Grid controls.

### Use Case Scenarios

You can create a Coded UI Test with Essential Grid for Windows Forms. The following example shows the implementation of the feature.

Perform the following initial steps before creating the Coded UI Test project:

1. Deploying Extension assembly
2. Prepare the Grid sample application
3. Write UI tests using VS 2010
4. Testing the application with generated Coded UI Tests

#### 4.11.1 Deploying Extension Assembly

To deploy the extension assembly:

1. Navigate to the **copydrop.bat** file in the **UITest** folder.
2. Run the **Bat** file to place the **Extension and Provider** assemblies with appropriate directories to pick the assemblies by **VS2010**.

## 4.11.2 Preparing the Grid Application

To prepare the Grid Application:

1. Syncfusion.VisualStudio.TestTools.UITest.GridCommunication.dll contains implementation to easily change an existing application to the test application that the plugin would require.
2. Let the parent container inherit **GridControlTestApplication** class as shown below:

[C#]

```
public class Form1 : GridControlTestApplication
{}
```

[VB]

```
Public Class Form1
    Inherits GridControlTestApplication
End Class
```

3. Build and run the application to make it ready for testing.

## 4.11.3 Creating Unit Tests with VS2010

To create Unit Tests with VS2010:

1. Create a new test project in **VS2010**.

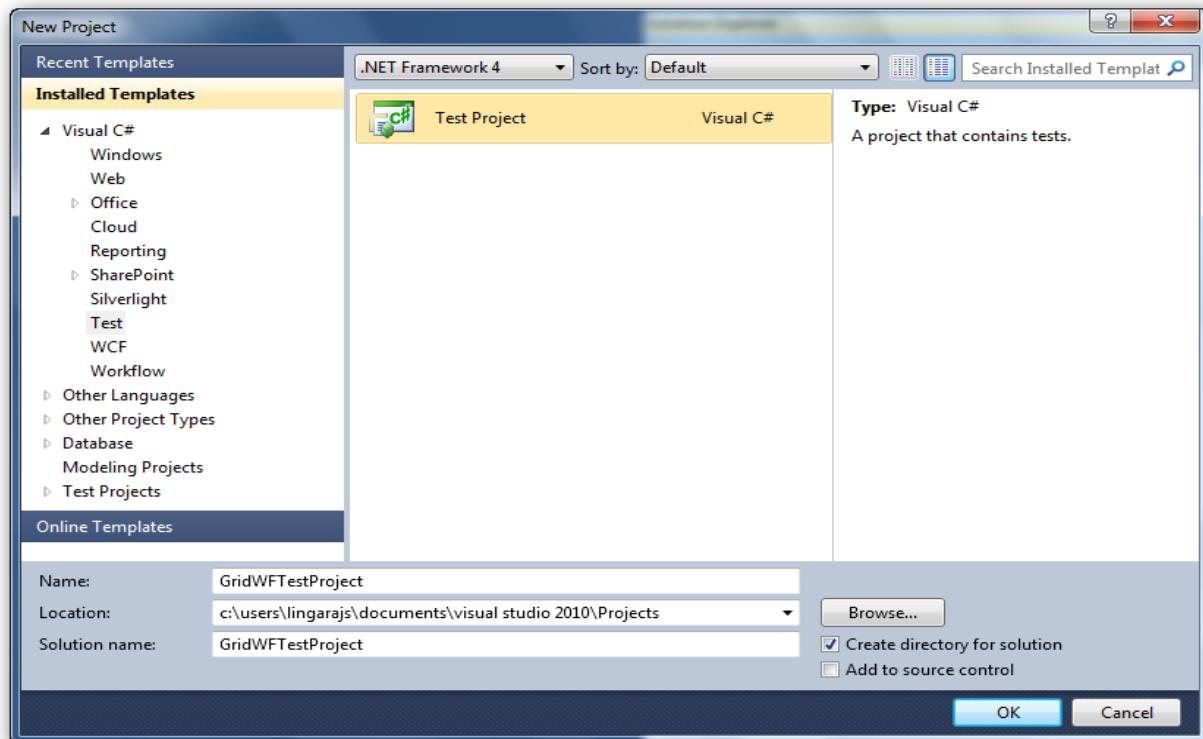


Figure 473: Creating Test Application

2. Add a new **CodedUITest** item for the project.

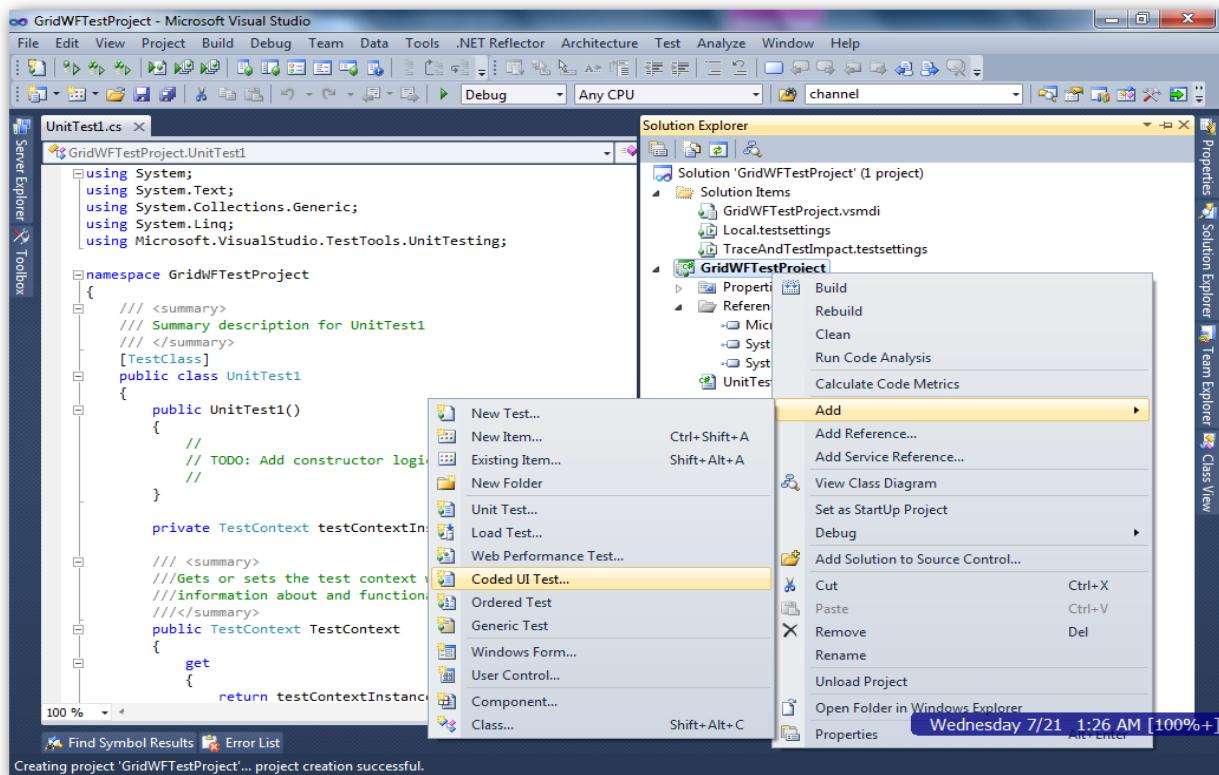


Figure 474: Adding Coded UI Test file

#### 4.11.4 Testing the Application with Generated Coded UI Tests

To test the application with generated coded UI Tests:

1. Add a TestMethod called CodedUITestMethod1.

```
[C#]
[TestMethod]
public void CodedUITestMethod1()
{
    // To generate codes for this test, select "Generate Code for Coded
    // UI Test" from the shortcut menu and select one of the menu items.
    // For more information on generated code, see:
    http://go.microsoft.com/fwlink/?LinkId=179463
}
```

```
[VB]
<TestMethod()>
Public Sub CodedUITestMethod1()
'
```

```
'To generate codes for this test, select "Generate Code for Coded UI Test" from the shortcut menu and select one of the menu items.
' For more information on generated code, see:
http://go.microsoft.com/fwlink/?LinkId=179463
'
End Sub
```

2. Build and run the **Grid** application that was configured already.
3. Right-click the **TestMethod** body and then select **Generate Code for Coded UI Test -> Use Coded UI Test Builder** as shown in the following screenshot:

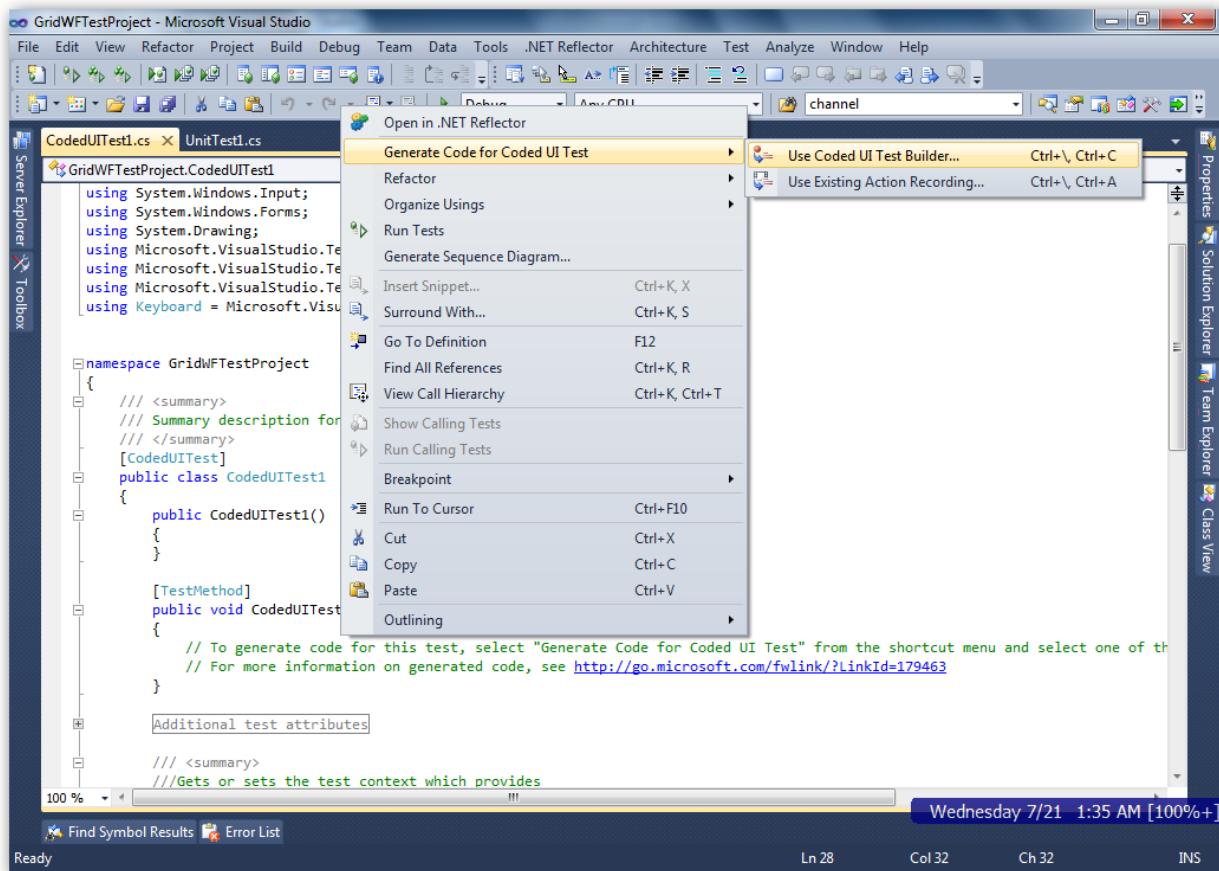


Figure 475: Opening Coded UI Test Builder

4. Click the **Record** button to perform actions. In this scenario, add a text **Hello World** in a cell [x, y].



Figure 476: Coded UI Map

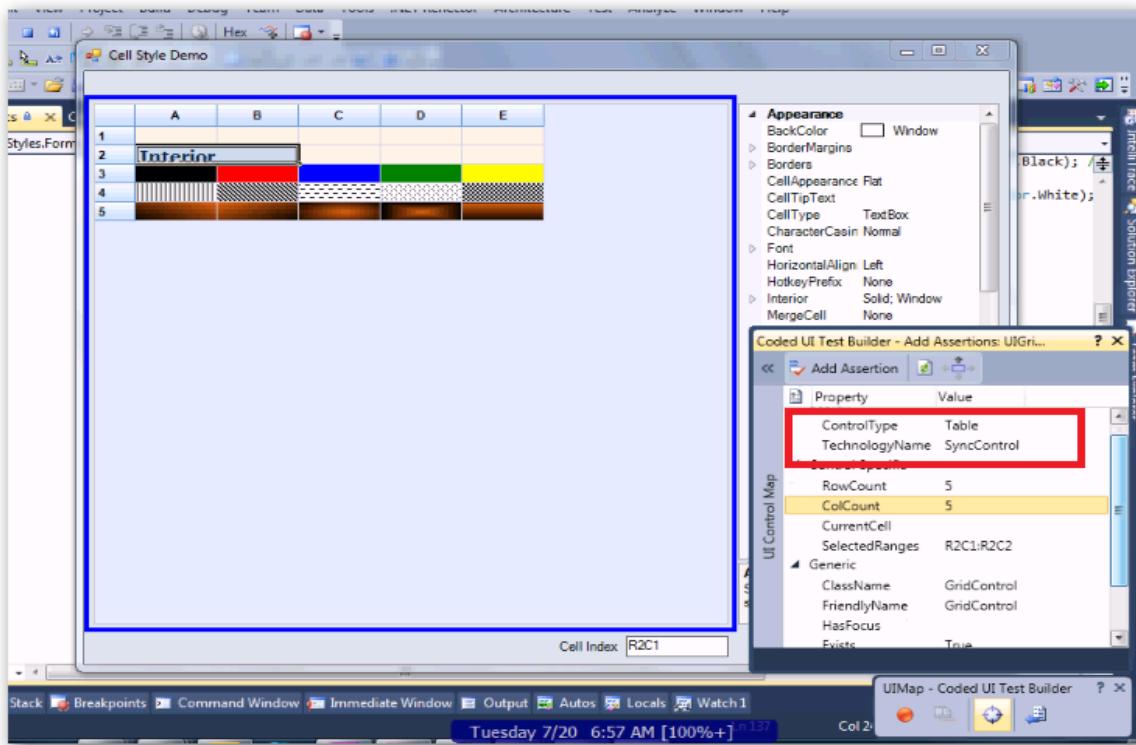


Figure 477: Identifying the Table of the Syncfusion Grid

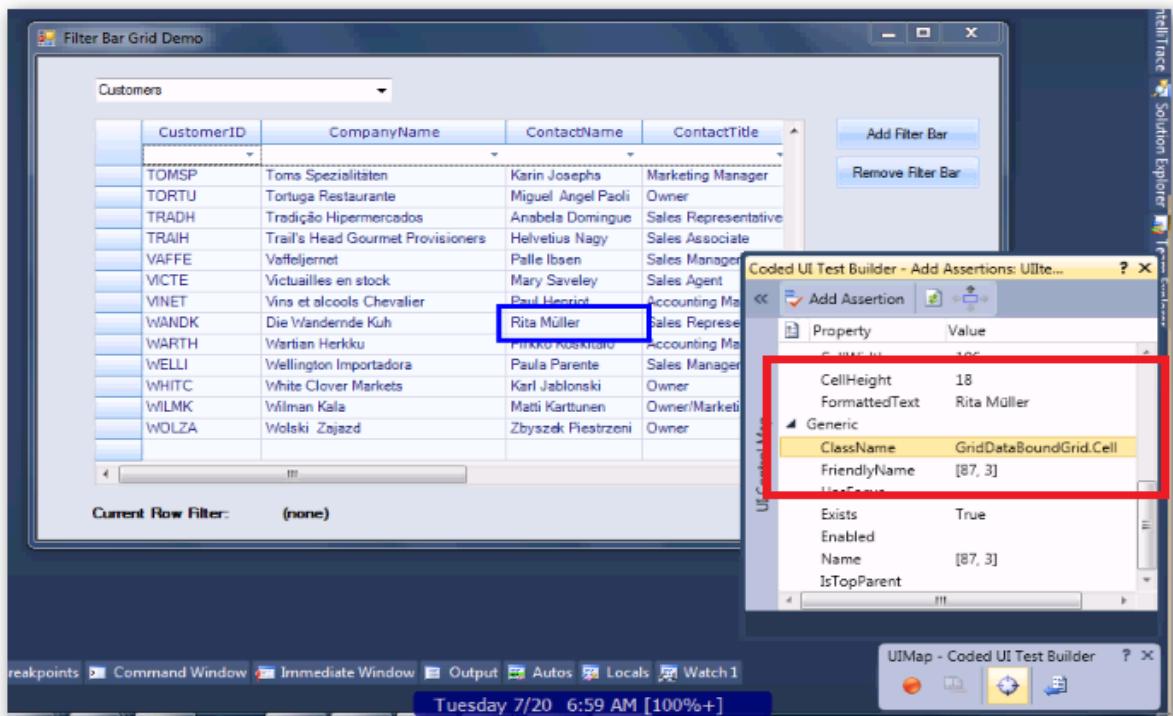


Figure 478: Identifying a Cell in the Syncfusion Grid

5. Assert the cell value using the cross-hair present in the **Coded UI Test builder**.
6. Click the cross-hair and hover to the cell. It will display the **Assert** window as shown in the following screenshot:

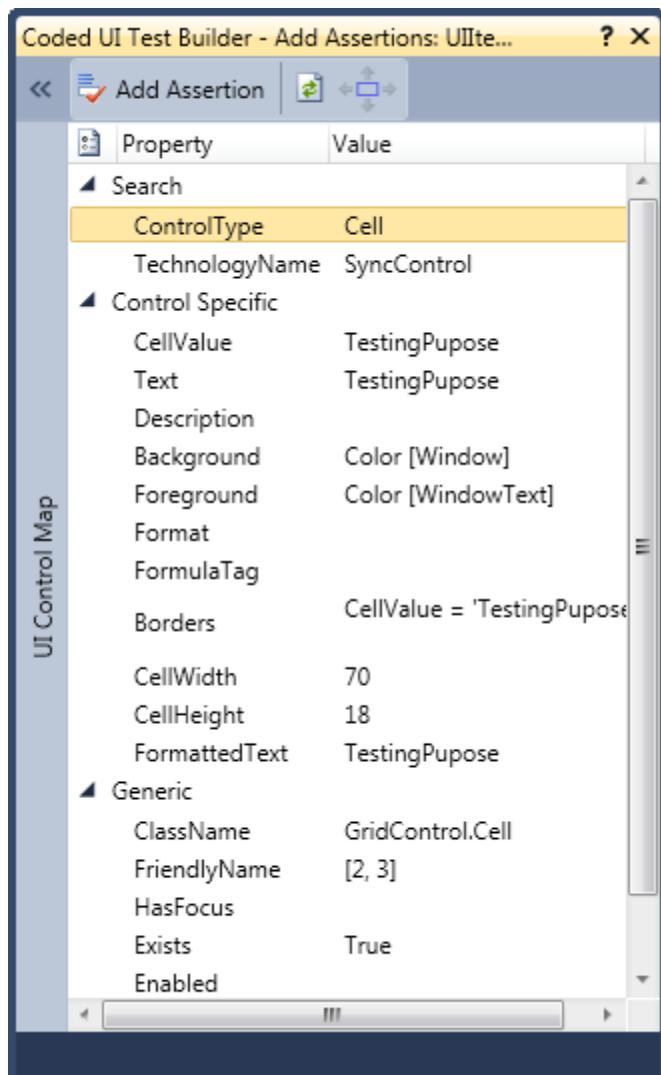


Figure 479: Assert Window

#### 4.11.5 Properties

Below is the Cell Properties Table-

Property	Description	Type	Data Type
AllowEnter	Indicates that by pressing the <Enter>-Key, a new line is inserted into the edited	-	bool

Property	Description	Type	Data Type
	text		
AllowPrompt	Specifies if the prompt character can be allowed to be entered as an input character	-	bool
AutoSize	Specifies that the cell height should automatically increase when the edited text does not fit into the cell and when WrapText is True. If WrapText is False, AutoSize will affect the column width	-	bool
BackGradientEndColor	The end color of the dual gradient of the background (Progress bar)	-	Color
BackGradientStartColor	The start color of the dual gradient of the background (Progress bar)	-	Color
Background	Background color of the control	-	Color
BackgroundFallbackStyle	Determines the background style when System mode is selected and the machine does not support themes	-	ProgressBarBackgroundStyles
BackgroundImageID	Gets / sets the image that the PictureBox displays	-	Image
BackgroundImageMode	Indicates how the image is displayed	-	GridBackgroundImageMode

Property	Description	Type	Data Type
BackgroundStyle	Determines the background style	-	ProgressBarBackgroundStyles
BackMultipleColors	The array of colors used to draw the multiple gradient of the background	-	Color[]
BackSegments	Determines if the background is segmented	-	bool
BackTubeEndColor	The outer color of the tube of the background	-	Color
BackTubeStartColor	The middle color of the tube of the background	-	Color
Borders	Top, left, bottom, and right border settings	-	GridBordersInfo
CellAppearance	Specifies if cell edges shall be drawn raised, sunken, or flat (default)	-	GridCellAppearance
CellTipText	ToolTip text to be displayed when the user hovers the mouse over the cell	-	string
CellType	Contains cell type information of a cell	-	string
CellValue	Contains cell value information of a cell	-	object
CheckState	Indicates the check state of the node	-	CheckState
Clickable	Specifies if the user can click on any cell button elements in this renderer	-	bool

Property	Description	Type	Data Type
Control	A custom control you can associate with a cell	-	Control
CultureInfo	The culture information holds rules for parsing and formatting the cell's value	-	CultureInfo
CurrencyDecimalDigits	Gets or sets the maximum number of digits for the decimal portion of the currency	-	int
CurrencyDecimalSeparator	Gets or sets the decimal separator character that will be used for the display	-	string
CurrencyGroupSeparator	Gets or sets the separator to be used for grouping digits	-	string
CurrencyGroupSizes	Gets or sets the grouping of CurrencyDigits in the CurrencyTextBox	-	int[]
CurrencyNumberDigits	Gets/Sets the number of digits for the number part. This value is initially set based on the maximum value of the Currency data type	-	int
CurrencySymbol	Gets or sets the currency symbol to be used in the CurrencyTextBox	-	string
DataSource	Indicates the list that this cell will use to get its items	-	object
DateSeparator	Gets or sets the character to use when a date separator position	-	char

Property	Description	Type	Data Type
	is specified		
DecimalSeparator	Gets or sets the character to use when a decimal separator position is specified	-	char
Description	Gets / sets the text that is shown in check box or pushbuttons	-	string
DisplayMember	Indicates the property to display for the items in this control	-	string
Enabled	Specifies if the cell can be activated as the current cell or if the cell should be skipped when moving the current cell	-	bool
FloatCell	Gets / sets if text can float into the boundaries of a neighboring cell	-	bool
FloodCell	Gets / sets if this cell can be flooded by a previous cell	-	bool
Font	The font for drawing text	-	GridFontInfo
Foreground	The foreground color used to display text and graphics in the control	-	Color
ForeSegments	Determines if the foreground is segmented (Progress bar)	-	bool
Format	Contains format information of a cell	-	string

Property	Description	Type	Data Type
FormulaTag	A custom tag you can associate with a cell	-	GridFormulaTag
GradientEndColor	The end color of the dual gradient of the foreground (Progress bar)	-	Color
GradientStartColor	The start color of the dual gradient of the foreground (Progress bar)	-	Color
HorizontalAlignment	Specifies the horizontal alignment of text in the cell	-	GridHorizontalAlignment
ImageIndex	Holds an image index that specifies an image inside an image list	-	int
ImageList	A list of images	-	ImageList
ImageSizeMode	Indicates how the image is displayed	-	GridImageSizeMode
Interior	Let's you specify a solid backcolor, gradient, or pattern style with both back and forecolor for a cell's background	-	BrushInfo
MaxLength	Limits the number of characters the user can type into the cell. Note: When selecting a text from a choice list or when pasting text, the text can be longer but additional validation is necessary from your side	-	int

Property	Description	Type	Data Type
MaxValue	Specifies the maximum value that can be set through the MaskedTextBox	-	decimal
MergeCell	Specifies if cell edges shall be drawn raised, sunken, or flat (default)	-	GridMergeCellDirection
MinValue	Specifies the minimum value that can be set through the MaskedTextBox	-	decimal
NegativeColor	Gets/Sets the forecolor when the current value is negative	-	Color
NegativeSign	Specifies the sign that is to be used to indicate a negative value	-	string
NullString	Specifies the string to be displayed when the DecimalValue is 0	-	string
NullValue	Specifies the value to be saved when the modified display text matches the null string	-	object
PaddingCharacter	Specifies the character that will be used instead of mask characters when the mask position has not been filled when the Text property is used	-	char
PassivePromptCharacter	The prompt character that will serve as a placeholder for mask characters when the control does not have the focus	-	char

Property	Description	Type	Data Type
PasswordChar	The character used to mask characters of a password in a password-entry cell	-	char
PositiveColor	Gets/Sets the forecolor when the current value is positive	-	Color
ProgressFontColor	The color of the font used to draw the text of the ProgressBar	-	Color
ProgressForeColor	The color used to draw the foreground in segment mode and constant mode	-	Color
ProgressMaximum	The higher bound of the range of the ProgressBar	-	int
ProgressMinimum	The lower bound of the range of the ProgressBar	-	int
ProgressMultipleColors	The array of colors used in the multiple gradient of the foreground	-	Color
ProgressOrientation	Determines the orientation of the text	-	Orientation
ProgressStep	The amount to increment the value of the ProgressBar when Increment() is called	-	int
ProgressStretchImage	Determines if the foreground image will be stretched	-	bool
ProgressStyle	Determines the foreground drawing style	-	ProgressBarStyles

Property	Description	Type	Data Type
ProgressTextVisible	Determines if the text of the Progressbar is visible	-	bool
ProgressValue	The current value between the minimum and maximum values	-	int
PromptCharacter	The prompt character that will serve as a placeholder for mask characters when the control does not have the focus (MaskEdit)	-	char
ReadOnly	Specifies if cell contents can be modified by the user. You can programmatically change Read-only cells by setting IgnoreReadOnly to True	-	bool
RightToLeft	Specifies if the cell content reads from right to left	-	RightToLeft
ShowButtons	Specifies when to show or display the cell buttons. Possible choices are: show the button only for the current cell, always show buttons, or never show buttons	-	GridShowButtons
StretchMultGrad	Determines if the multiple gradient will be stretched	-	bool
StrictValueType	Gets or sets a value indicating whether an exception should be thrown in the ApplyFormattedText method if the formatted	-	bool

Property	Description	Type	Data Type
	text cannot be parsed and converted to the type specified with CellValueType		
Tag	A custom tag you can associate with a cell	-	object
Text	Gets / sets the value as a string	-	string
TextAlign	Align text left of button elements (which is typical for combo boxes). Or align text right of button elements	-	GridTextAlign
TextShadow	Determines if the text shadow is visible	-	bool
TextStyle	Determines the style of the text	-	ProgressBarTextStyles
ThousandSeparator	The character to use when a thousand separator position is specified	-	char
TimeSeperator	The character to use when a time separator position is specified	-	char
Trimming	Indicates how text is trimmed when it exceeds the edges of the cell text rectangle	-	StringTrimming
TubeEndColor	The outer color of the tube of the foreground (Progress bar)	-	Color
TubeStartColor	The middle color of the tube of the foreground	-	Color

<b>Property</b>	<b>Description</b>	<b>Type</b>	<b>Data Type</b>
UsageMode	Specifies if the MaskedTextBox is to behave as a numeric control	-	MaskedUsageMode
UseCultureInfo	Specifies whether the NumberFormat should be based on Grid's style or individual properties of this object	-	bool
UseLocaleDefault	Specifies if the individual globalization property changes are to be ignored	-	bool
UseUserOverride	The UseUserOverride parameter for CultureInfo	-	bool
ValueMember	Indicates the property to use as the actual value for the items in the control	-	string
VerticalAlignment	Specifies the vertical alignment of the text in the cell	-	GridVerticalAlignment
VerticalScrollBar	Specifies if the text box should show a vertical scrollbar when text is being edited and does not fit in cell. WrapText must be initialized to True	-	bool

Here is the table for Control Properties Table

<b>Property</b>	<b>Description</b>	<b>Type</b>	<b>Data Type</b>
RowCount	Gets or sets the number of rows in the grid	-	int

ColCount	Gets or sets the number of columns in the grid	-	int
SelectedRanges	Gets the Selected Ranges	-	GridRangeInfoList
CurrentCell	Gets GridCurrentCell object that provides storage for current cell information and manages all current cell operations such as activating, deactivating, saving, editing, and moving the current cell	-	GridCurrentCell
GridOfficeScrollBars	Gets or sets MS Office-like scrollbars	-	OfficeScrollBars
GridVisualStyles	Specifies the look and feel skins for the Grid	-	GridVisualStyles
ThemesEnabled	Specifies whether XP Themes should be used for this control when available	-	bool

#### 4.11.6 In-built Coded UI Support for 3.5 and 4.0 Frameworks

Essential Grid for Windows Forms supports automated UI testing with VS 2010 coded UI technology. Without using the existing plugins, the coded UI test is now enabled in frameworks 3.5 and 4.0.

#### Properties Table

Table 18: Property Table

Property	Description	Data Type
AccessibilityEnabled	Gets or sets a value indicating whether the control should enable its accessibility support.	Boolean, true/false

#### Enabling AccessibilityEnabled Property

The following code example illustrates how to enable the **AccessibilityEnabled** property for the control.

[C#]

```
this.grid.AccessibilityEnabled = true;
```

[VB]

```
Me.grid.AccessibilityEnabled = True
```



**You can follow the same steps to test the application with the generated coded UI tests as described in the chapter [Testing the Application with Generated Coded UI Tests](#).**

## 4.12 Built-in Error Provider Support

Essential Grid for Windows Forms now provides a built-in error provider for error alerts. This feature enables you to display an error icon in a specific cell and row header when incorrect data is entered in a cell. This also enables you to specify the error conditions.

### Use Case Scenarios

This feature is useful when you want to set that only numeric values can be entered in a cell.

### Properties

Property	Description	Type	Data Type
ShowerrorIcon	Specifies whether to show error icon.	NA	Boolean
ShowRowHeaderErrorIcon	Specifies whether to show error icon in the row header.	NA	Boolean
ShowErrorMessageBox	Specifies whether to show error message box.	NA	Boolean
ValidationEditText	Specifies the text to be displayed in the error message box.	NA	Boolean

### Methods

Method	Description	Parameters	Type	Return Type

SetError()		Method (string)	Method	String
------------	--	-----------------	--------	--------

**Sample Link**

A sample of this feature is available in the following location:

**GridDataBoundGrid**

**{Installed**  
**Path}\Syncfusion\EssentialStudio\{Version}\Windows\GridDataBound.Windows\Samples\2.0\Appearance>Error Provider Demo**

**GridGroupingGrid**

**{Installed**  
**Path}\Syncfusion\EssentialStudio\{Version}\Windows\Grid.Grouping.Windows\Samples\2.0\Grouping Grid Layout>Error Provider Demo**

**GridControl**

**{Installed**  
**Path}\Syncfusion\EssentialStudio\{Version}\Windows\Grid.Windows\Samples\2.0\GridLayout>Error Provider Demo**

### 4.12.1 Enabling Error Alert

You can show an error icon or error message box as an alert for incorrect data.

**Displaying Error Icon**

You can show the error icon in the cell and row header using the **ShowerrorIcon** and the **ShowRowHeaderErrorIcon** properties respectively.

To show the error icon in the cell, set the **ShowerrorIcon** property to **True**. By default this will be set to **True**. To show the error icon in the row header, set the **ShowRowHeaderErrorIcon** property to **True**. By default this will be set to **False**.

You can display the error icon in the cell as well as a row header if needed.

The following code illustrates how to display the error icon on both the cell as well as the row header:

[C#]

```
this.gridDataBoundGrid1.CurrentCell.ShowerrorIcon = true;
this.gridDataBoundGrid1.ShowRowHeaderErrorIcon = true;
```

[VB]

```
Me.gridDataBoundGrid1.CurrentCell.ShowerrorIcon = True
Me.gridDataBoundGrid1.ShowRowHeaderErrorIcon = True
```

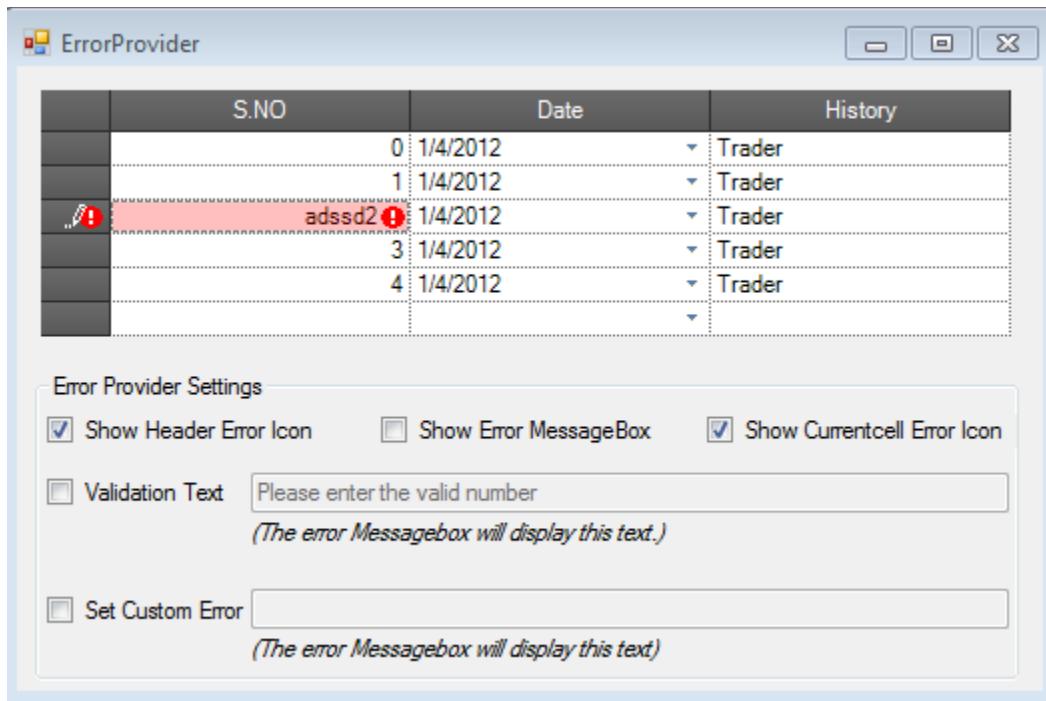


Figure 480: Error Icon

### Displaying Error Message Box

You can show an error dialog using the **ShowErrorMessageBox** property and specify the content to be displayed using the **ValidationText** property.

The following code illustrates this:

[C#]

```
this.gridDataBoundGrid1.CurrentCell.ShowErrorMessageBox= false;
```

```
this.gridDataBoundGrid1.CurrentCell.ValidationErrorText = "this is the text";
```

[VB]

```
Me.gridDataBoundGrid1.CurrentCell.ShowErrorMessageBox= False  
Me.gridDataBoundGrid1.CurrentCell.ValidationErrorText = "this is the text"
```

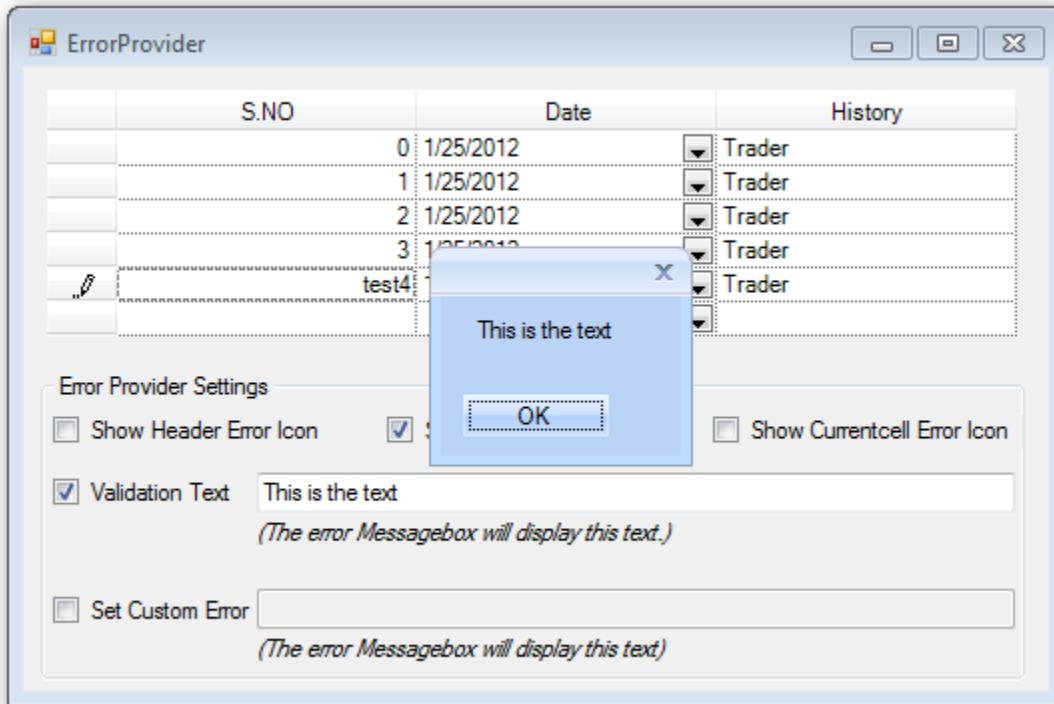


Figure 481: Error Message Box

### Specifying Error Content

You can specify error conditions for individual cells using the **SetError()** method of the **GridCurrentCell**.

The following code illustrates this:

[C#]

```
this.gridDataBoundGrid1.CurrentCell.SetError("Please enter valid number");
```

[VB]

```
Me.gridDataBoundGrid1.CurrentCell.SetError("Please enter valid number")
```

## 4.13 Support for Skin Manager in Windows Forms Grid

This feature enables you to easily apply themes to all grids in an application through single functional call. The Skin Manager provides support for the following themes:

- Office 2007 Blue
- Office 2007 Black
- Office 2007 Silver
- Office 2010 Blue
- Office 2010 Black
- Office 2010 Silver

### Use Case Scenarios

When you create an application with multiple Grid controls, you can apply uniform themes for the entire form and its child controls using this feature.

### Properties Table

*Table 19: Properties Table*

Property	Description	Data Type
Control	Specifies the parent control for which theme has to be applied.	Control
VisualTheme	Specifies the style.	Enum

### Methods Table

Table 20: Method Table

Method	Description	Parameters	Return Type
VisualTheme(Control,VisualStyle)	Specifies the parent control and the theme to be applied.	Overloads: <ul style="list-style-type: none"><li>• (Control, String)</li><li>• (Control,VisualStyle)</li></ul>	Void

**Sample Link**

<Sample installation Location>\Syncfusion\EssentialStudio\<>Version Number>>\Windows\Grid.Windows\Samples\2.0\Appearance\SkinManager Demo

#### 4.13.1 Adding Skin Manager to an Application

You can add the SkinManager control to one of the controls in your form or to the entire control as needed by specifying the root control. You can specify the root control using the **Control** property.

To add the SkinManager control to one of the controls, specify the selected control as the root control. The following code illustrates this, where the control is styled with Office2010Blue theme.

**[C#]**

```
SkinManager.SetVisualStyle(this.gridControl1, VisualTheme.Office2010Blue);
```

**[VB]**

```
SkinManager.SetVisualStyle(Me.gridControl1, VisualTheme.Office2010Blue)
```

To add Skin Manager to the entire form, specify the form as the root control. The following code illustrates this.

**[C#]**

```
SkinManager.SetVisualStyle(this, VisualTheme.Office2010Blue);
```

**[VB]**

```
SkinManager.SetVisualStyle(Me, VisualTheme.Office2010Blue)
```

Implementing the above code will create the following output.

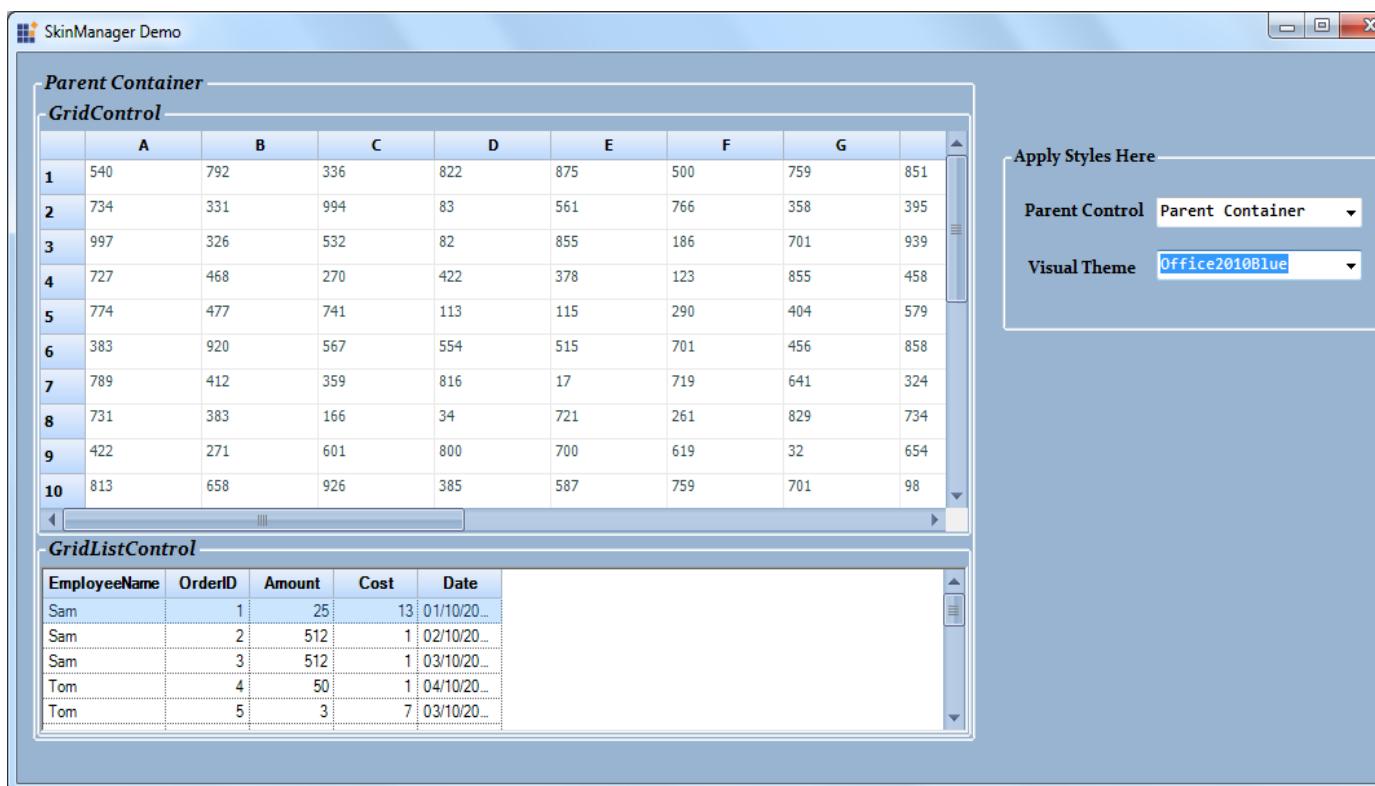


Figure 482: Grid control styled with Office2010Blue

## 4.14 Localization Support

Localization support allows you to set the content according to the language or culture of a specific country or region. Essential Grid provides the localization support for all controls.

### Use Case Scenarios

With this feature, you can localize the options in the grid to display the content according to the language or culture of a specific country or region.

### Sample Link

To open sample project, navigate to the following sample location in your system:

**<<Sample Installation Location>>\Syncfusion\EssentialStudio\<<Version Number>>\Windows\Grid.Grouping.Windows\Samples\2.0\Localization Samples\GroupingGridLocalization Demo**

### **Adding Localization Support to Grid Controls**

To localize the content, create a class file and add the **ILocalizationProvider** interface to the class. Assign the required content to be displayed to **DynamicFilterResourceIdentifiers**, **GroupingResourceIdentifiers**, and **GridResourceIdentifiers** of the **GetLocalizedString** method as illustrated in the following code.

[C#]

```
public string GetLocalizedString(System.Globalization.CultureInfo culture, string name, object obj)
{
    switch (name)
    {
        case DynamicFilterResourceIdentifiers.StartsWith:
            if (comparerList)
                return "empieza con";
            else
                return "StartsWith";

        case GroupingResourceIdentifiers.RecordNavigatorOF:
            if (recordNavBar)
                return "von";
            else
                return "Of";

        //Drag group column
        case GroupingResourceIdentifiers.DragColumnHeaderHereText:
            return "Ziehen Sie die Spaltenüberschrift";

        case DynamicFilterResourceIdentifiers.SortAtoZ:
            if (office2007Filter)
                return "&SortierenAbisZ";
            else
```

```
        return "&SortAtoZ";  
    default:  
        return string.Empty;  
    }  
}
```

**[VB]**

```
Public Function GetLocalizedString(ByVal culture As  
System.Globalization.CultureInfo, ByVal name As String, ByVal obj As  
Object) As String Implements ILocalizationProvider.GetLocalizedString  
    Select Case name  
        Case  
DynamicFilterResourceIdentifiers.StartsWith  
            If comparerList Then  
                Return "empieza con"  
            Else  
                Return "StartsWith"  
            End If  
  
        Case  
GroupingResourceIdentifiers.RecordNavigatorOF  
            If recordNavBar Then  
                Return "von"  
            Else  
                Return "Of"  
            End If  
  
        Case DynamicFilterResourceIdentifiers.SortAtoZ  
            If office2007Filter Then
```

```

        Return "&SortierenAbisZ"
    Else
        Return "&SortAtoZ"
    End If

Case Else
    Return String.Empty
End Select
End Function

```

Add the following table items to the above code example by substituting the identifiers and Enum values as in they correspond in the table.

Enum		
DynamicFilterResourceIdentifiers	StartsWith	CustomAutoFilterGreaterthan
	EndsWith	SelectAll
	Equals	SortAtoZ
	GreaterThan	SortZtoA
	GreaterThanOrEqualTo	ClearFilterFrom
	LessThan	All
	LessThanOrEqualTo	Custom
	Like	Empty
	Match	ExpressionMATCH
ResourceIdentifiers		OK
GridResourceIdentifiers		Cannotchangepartofamergedcell

This will generate the following output.

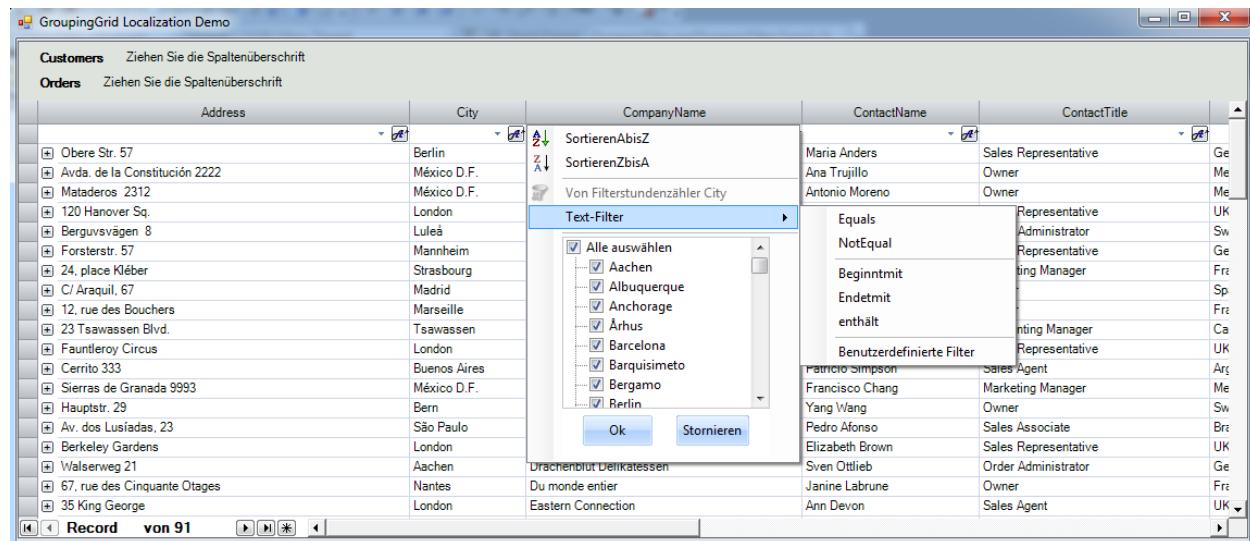


Figure 483: Localization support for Gridgrouping control

#### 4.14.1 Localization Support for ComboBox Cell

Localization support allows you to set the content in a ComboBox cell according to the language or culture of a specific country or region. You can localize the content in the ComboBox cell using the **ConvertToString** method of **Status** class.

##### Use Case Scenarios

With this feature, you can localize the options in the ComboBox cell to display the content according to the language or culture of a specific country or region.

##### Sample Link

To get the DataGrid samples from the dashboard:

1. Open **Syncfusion Dashboard**.
2. Select **UI > Windows Forms**.
3. Click **Run Samples**.
4. Select **Data Grid**.
5. Expand **Localization samples** in the left panel of sample browser and click **ComboBoxCellLocalization Demo**.

##### Adding Localization Support to ComboBox Cell

The following code example illustrates how to add the localization support to the ComboBox cell using **ConvertToString** method of **Status** class.

[C#]

```
public object ConvertToString(ITypeDescriptorContext context,
System.Globalization.CultureInfo culture, object value)
{
    switch ((Status)value)
    {
        case Status.Divorced:
            return "geschieden";
        case Status.Married:
            return "verheiratet";
        case Status.Single:
            return "Einzel";
        case Status.Widow:
            return "Witwe";
        default:
            return string.Empty;
    }
}
```

[VB]

```
Private Overloads Function ConvertFromString(ByVal context As
ITypeDescriptorContext, ByVal culture As
System.Globalization.CultureInfo, ByVal value As Object) As Object
    Select Case CStr(value)
        Case "geschieden"
            Return Status.Divorced
        Case "verheiratet"
            Return Status.Married
        Case "Einzel"
```

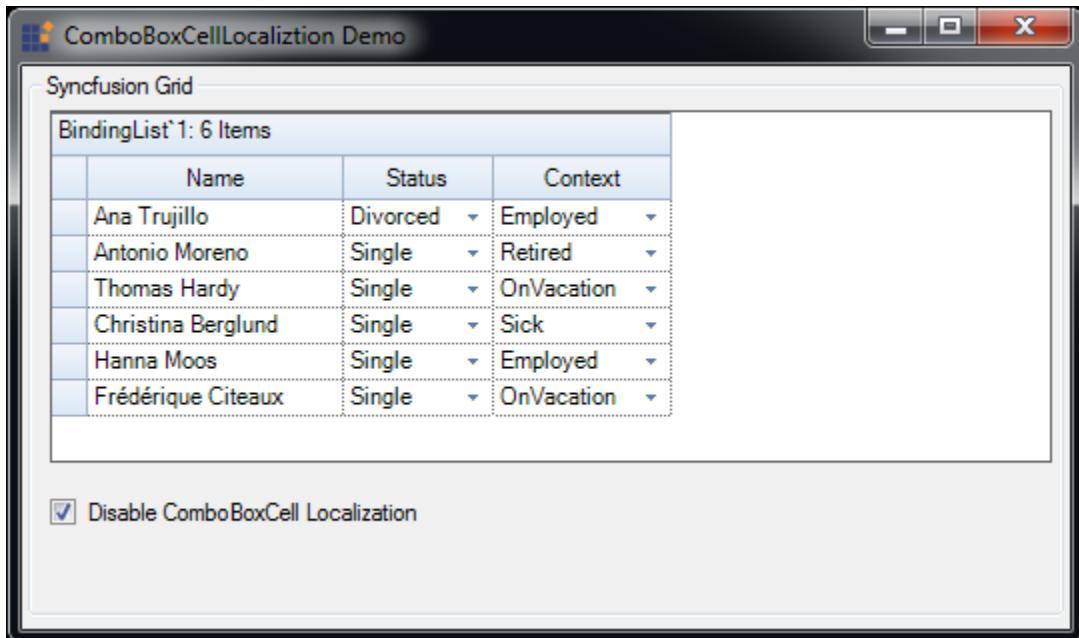
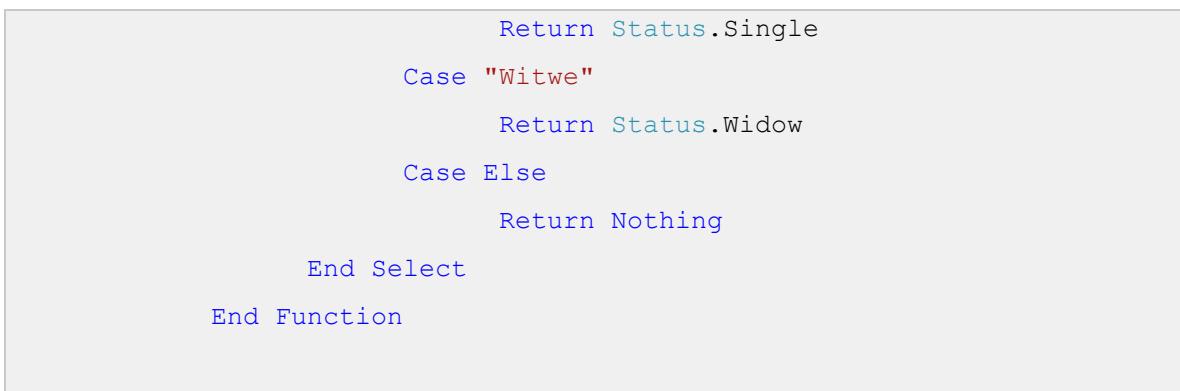


Figure 484: Localization Support for ComboBox Cell

## 4.15 Zoom Support

Essential Grid provides zoom support for the Grid, GridDataBoundGrid, and GridGrouping controls. With this support, users can zoom in or out of the entire grid as well as individual cells of the grid.

### Use Case Scenarios

You can zoom in or out the Grid at the specified percentage level in your application by using this feature.

### Properties Table

Table 21: Property/ies Table

Property	Description	Data Type
ZoomGridControlCell	Used to enable zooming at the cell level for the GridControl.	Boolean
ZoomGridDataBoundCell	Used to enable zooming at the cell level for the GridDataBoundGrid.	Boolean

**Methods Table**

Table 22: Methods Table

Method	Description	Parameters	Return Type
zoomGrid	Gets the percentage value as the parameter and uses this value to set font and cell size for the grid.	Overloads: 1) (string Arg1)	Void
ZoomGrid	This constructor gets Grid control to be zoomed as a parameter.	Overloads: 1)(Class Arg1)	Constructor
ZoomGroupingGrid	This constructor gets the GridGrouping control as a parameter and defines zooming function for the GridGroupingControl.	Overloads: 1)(Class Arg1)	Constructor

**Sample Link**

To open sample project, navigate to the following sample locations in your system:

For GridControl:

**<<Sample Installation Location>>\Syncfusion\EssentialStudio\<<Version Number>>\Windows\Grid.Windows\Samples\2.0\Zoom and Scrolling\GridControl Zoom**

For GridDataBoundGrid:

<<Sample Installation Location>>\Syncfusion\EssentialStudio\<<Version Number>>\Windows\GridDataBound.Windows\Samples\2.0\Zoom\Zoom Grid Demo

For GridGroupingControl:

<<Sample Installation Location>>\Syncfusion\EssentialStudio\<<Version Number>>\Windows\Grid.Grouping.Windows\Samples\2.0\Zoom\Zoom Grid Demo

### 4.15.1 Zooming Grid Controls

To enable the zooming support for a control, you need to pass the control as a parameter to implement zooming functionality. Then, you need to define percentage value of zooming by using the **zoomGrid** method. The following code examples illustrate how to zoom the entire grid of the GridControl, GridDataBoundGrid, and GridGrouping controls.

#### For GridControl

[C#]

```
ZoomGrid zoom = new ZoomGrid(this.gridControl1);
zoom.zoomGrid(this.trackBar1.Value.ToString());
```

[VB]

```
Dim zoom As New ZoomGrid(Me.gridControl1)
zoom.zoomGrid(Me.trackBar1.Value.ToString())
```

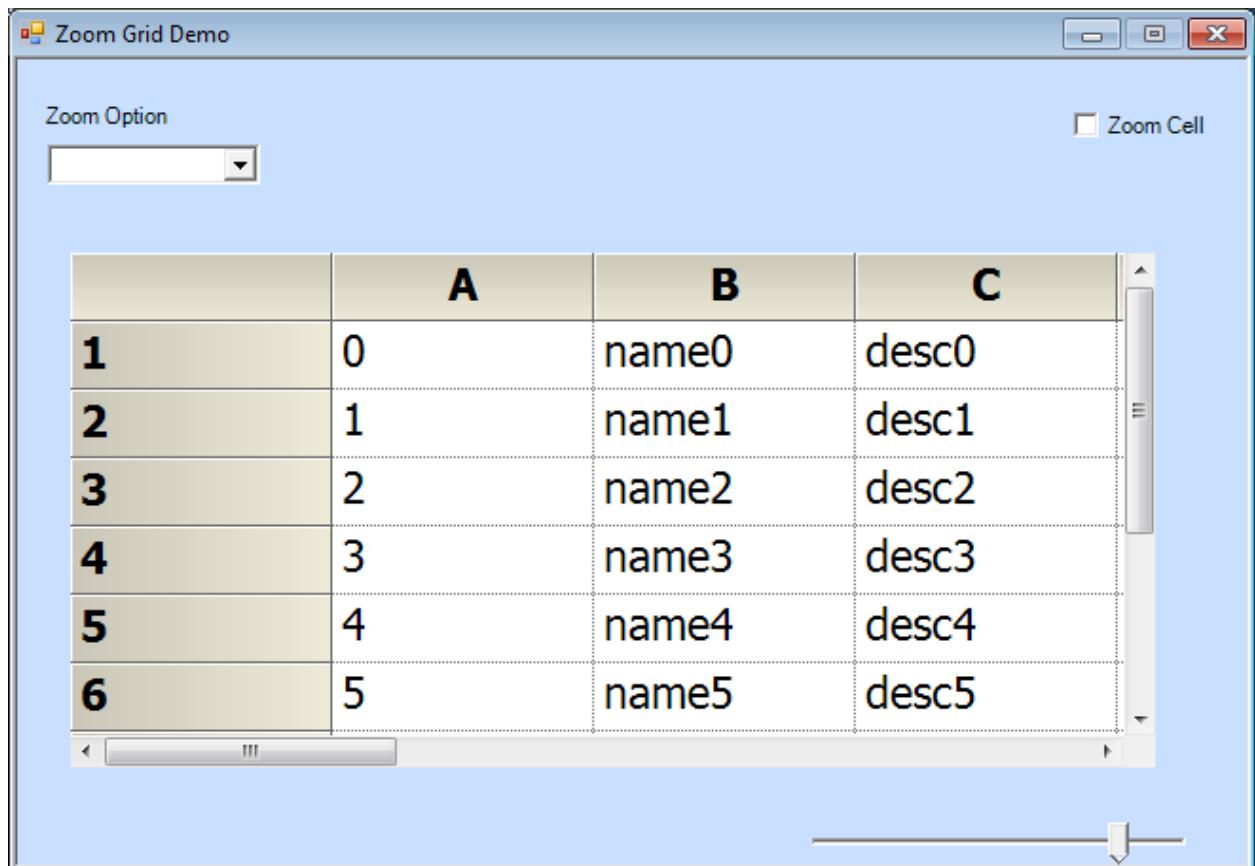


Figure 485: Zooming GridControl

#### For GridDataBoundGrid Control

[C#]

```
ZoomGrid zoom = new ZoomGrid(this.gridDataBoundGrid1);
zoom.zoomGrid(this.trackBar1.Value.ToString());
```

[VB]

```
Dim zoom As New ZoomGrid(Me.gridDataBoundGrid1)
zoom.zoomGrid(Me.trackBar1.Value.ToString())
```

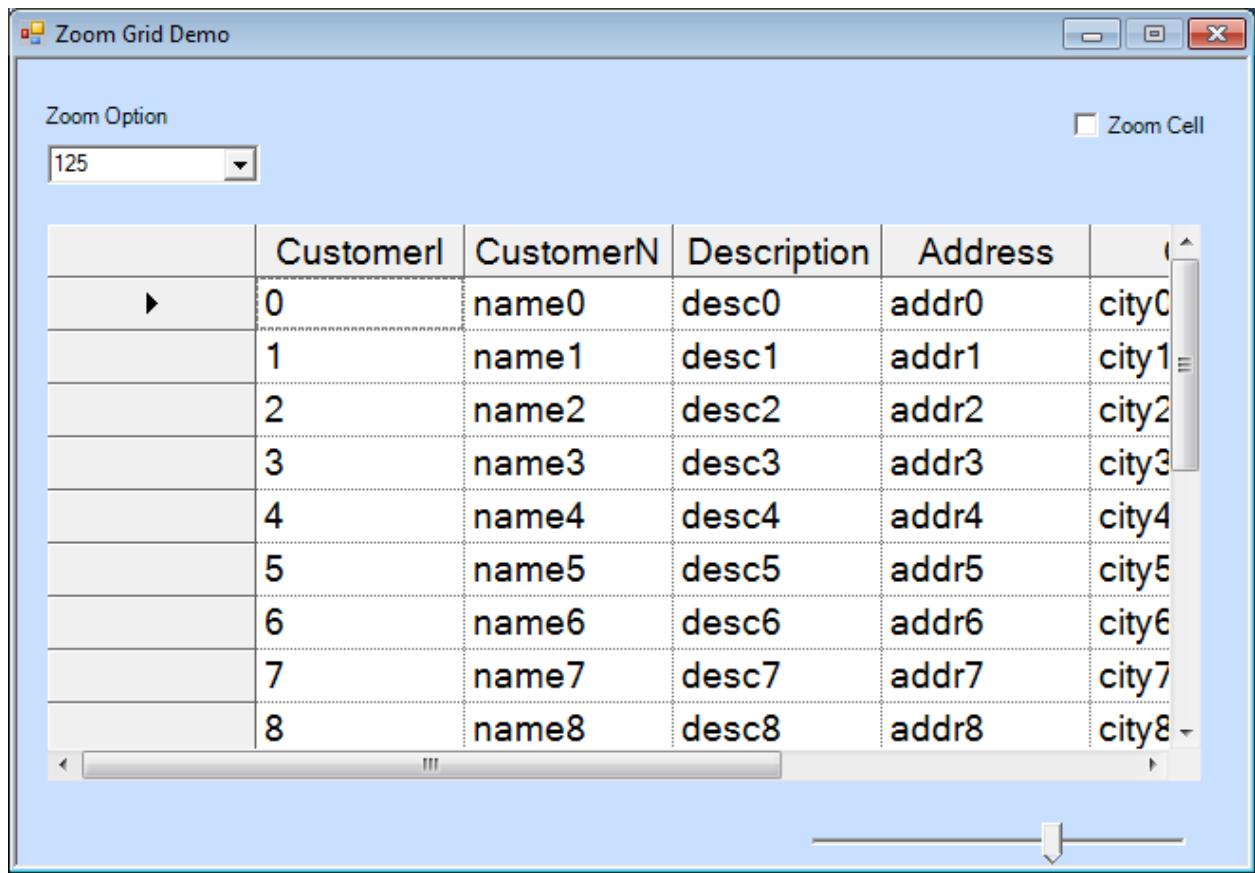


Figure 486: Zooming GridDataBoundGrid Control

### For GridGrouping Control

[C#]

```
ZoomGroupingGrid zoom = new
ZoomGroupingGrid(this.gridGroupingControl1);
zoom.zoomGrid(this.trackBar1.Value.ToString());
```

[VB]

```
Dim zoom As New ZoomGroupingGrid (Me.gridGroupingControl1)
zoom.zoomGrid(Me.trackBar1.Value.ToString())
```

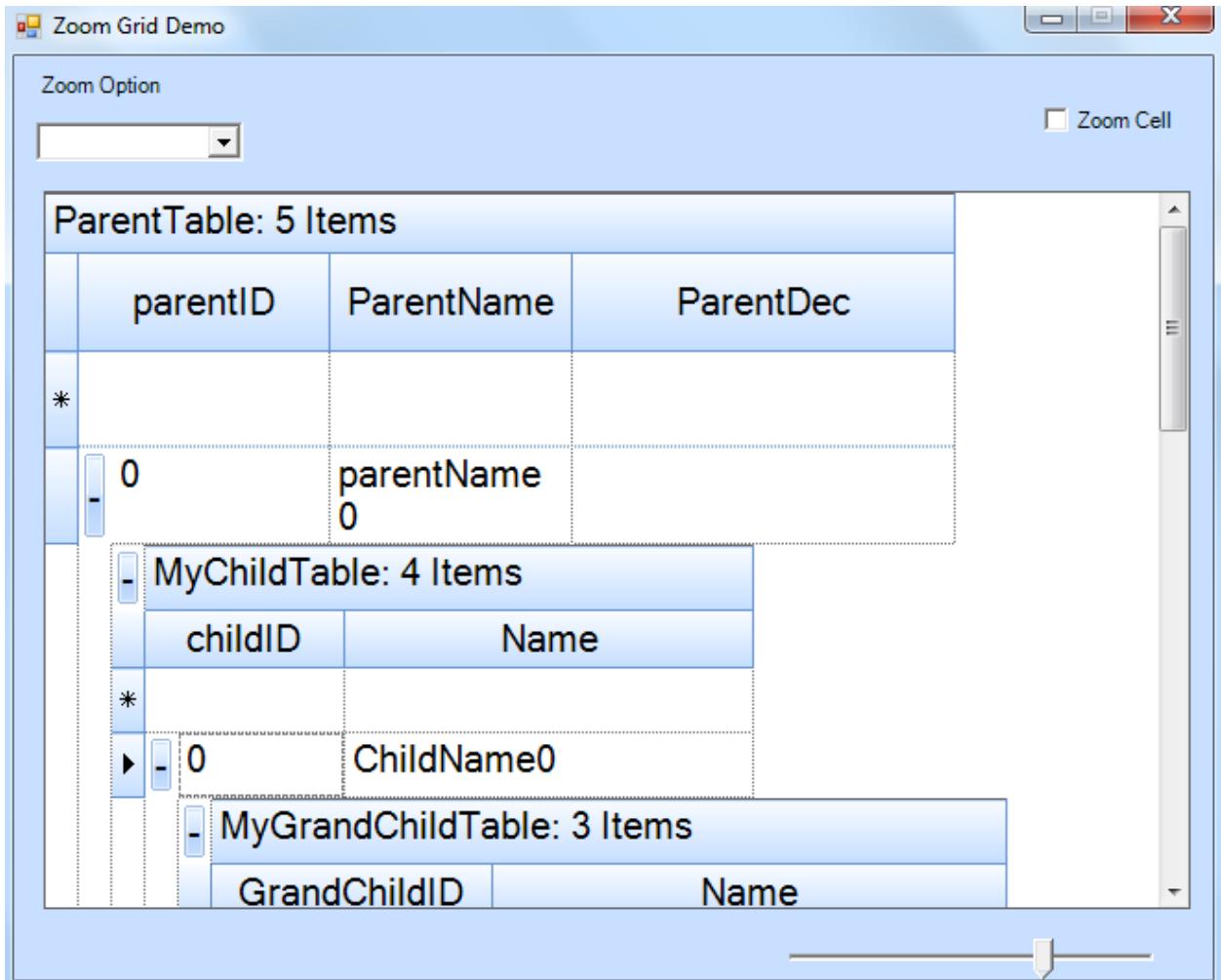


Figure 487: Zooming GridGrouping Control

#### 4.15.2 Zooming Individual Cells of the Grid

You can also zoom individual cells of a grid by using the **ZoomGridControlCell** property. This facilitates to display the selected cell content in a maximized form. When you click any cell, this sample displays a picture box control over the cell, displaying the content in large fonts.

[C#]

```
ZoomGrid.ZoomGridControlCell = true;
```

[VB]

```
ZoomGrid.ZoomGridControlCell = True
```

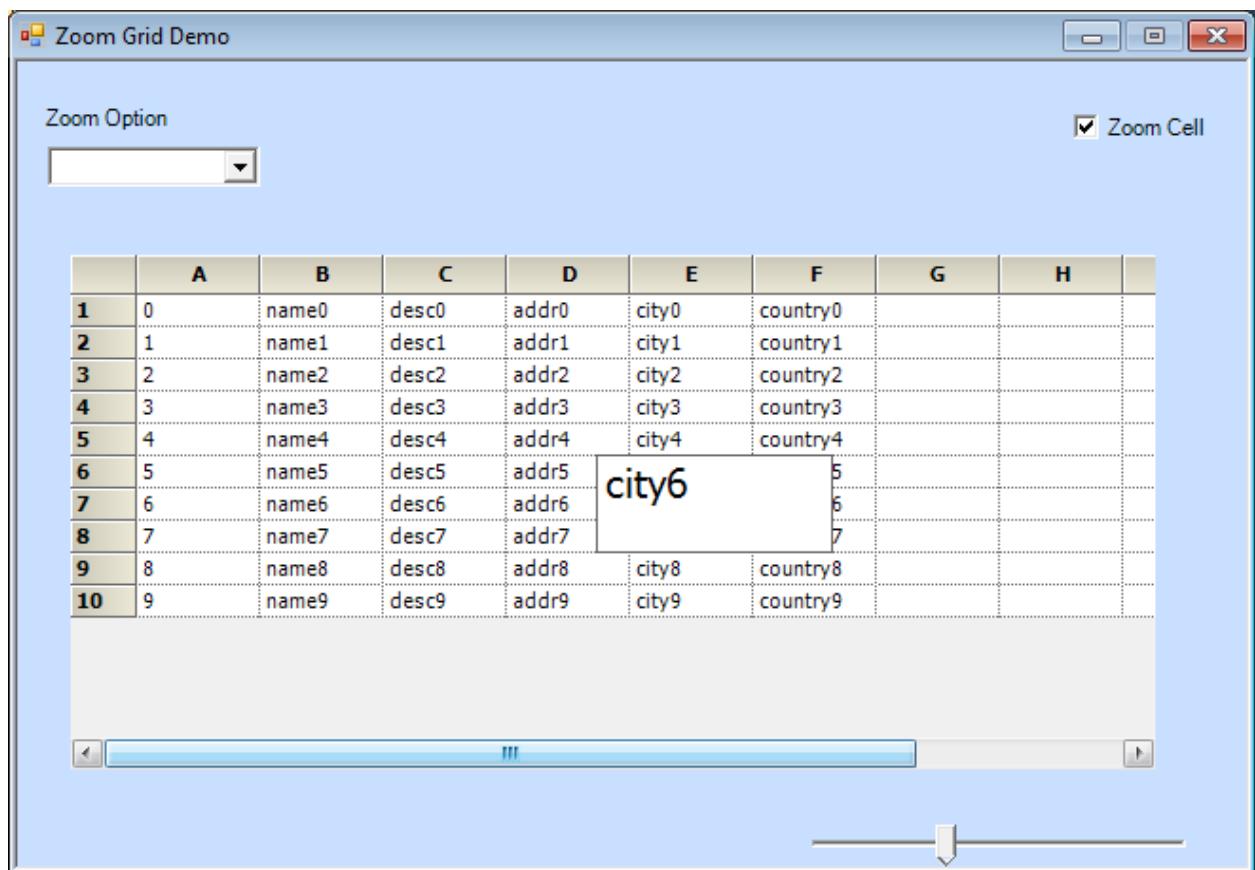


Figure 488: Zooming a Cell in a Grid

## 5 Frequently Asked Questions

---

This section discusses specific solutions for very specific tasks. It is organized into various topics based primarily on the control. There are several sections that contain information which is not control specific. In general, if a discussion applies to a specific control like a GridControl or a GridDataBoundGrid, the variable names used in the discussion will reflect that particular control. If the discussion applies to either a GridControl or a GridDataBoundGrid, the generic variable name, grid, will be used.

It comprises the below sections:

### 5.1 GridControl

The GridControl is a cell-oriented grid that can be used in an unbound mode where the GridControl holds the data or in a virtual mode where the data is dynamically provided to the GridControl on demand through certain events. The tasks and solutions discussed in this section are specific to the GridControl.

#### 5.1.1 How to Align Radio Buttons

The Windows Forms Grid control includes support for displaying the radio button of the RadioButton cell type in both vertical and horizontal order.

By default, the RadioButton cell aligns the buttons in horizontal order. The display order can be changed using the RadioButtonAlignment property.

The following code examples illustrate how to invoke the RadioButtonAlignment property.

##### Using C#:

```
[C#]

this.gridControl1[1, 2].RadioButtonAlignment =
ButtonAlignment.Vertical;

this.gridControl1[2, 2].RadioButtonAlignment =
ButtonAlignment.Horizontal;
```

##### Using VB.NET:

[VB]

```
me.gridControl1[1, 2].RadioButtonAlignment = ButtonAlignment.Vertical  
  
me.gridControl1[2, 2].RadioButtonAlignment = ButtonAlignment.Horizontal
```

The following screenshot illustrates the RadioButtonAlignment property:

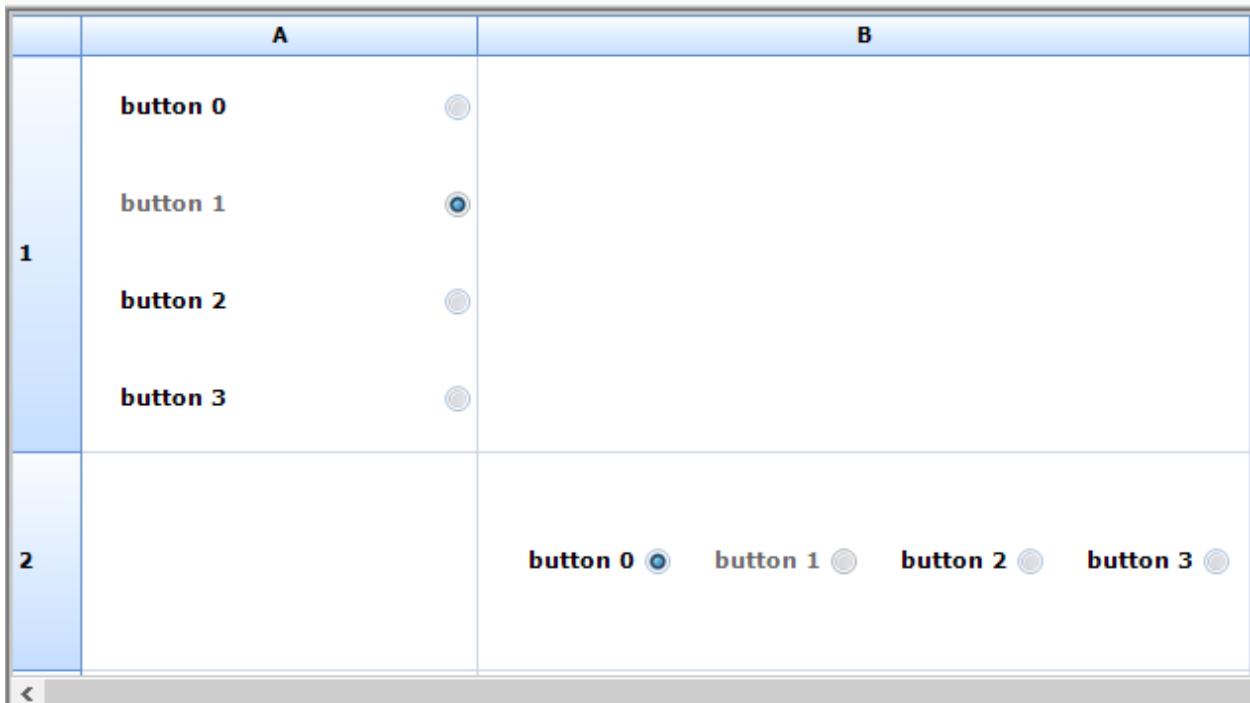


Figure 489: Radio Buttons with Horizontal and Vertical alignment

## 5.1.2 How to Add a Sort Icon (Up / Down Arrow) in a GridControl's Column Header

### Introduction

Add the **GridSortColumnHeaderCellModel** to the GridControl's **CellModels** collection to include the **SortColumn** HeaderCell. Then assign this as the CellType and set the **tag** property to either **ListSortDirection.Ascending** or **ListSortDirection.Descending** to show the up / down arrow mark.

### Example

[C#]

```
// Registering the GridSortColumnHeaderCellModel to the GridModel.  
this.gridControl1.CellModels.Add("SortHeader", new  
GridSortColumnHeaderCellModel(this.gridControl1.Model));  
  
// Setting the new celltype to a column header cell.  
this.gridControl1[0,1].CellType = "SortHeader";  
  
// Specifying the sort direction in the Tag property of the column  
header.  
this.gridControl1[0,1].Tag = ListSortDirection.Ascending;
```

**[VB.NET]**

```
' Registering the GridSortColumnHeaderCellModel to the GridModel.  
Me.GridControl1.CellModels.Add("SortHeader", New  
GridSortColumnHeaderCellModel(Me.GridControl1.Model))  
  
' Setting the new celltype to a column header cell.  
Me.GridControl1(0, 1).CellType = "SortHeader"  
  
' Specifying the sort direction in the Tag property of the column  
header.  
Me.GridControl1(0, 1).Tag = ListSortDirection.Ascending
```



**Note:** This will only display the sort indicator; it does not actually do any sorting.

### 5.1.3 How to Avoid the A, B, C and / or 1, 2, 3 in the Headers

#### Introduction

In a GridControl, whether the headers contain the default A, B, C, ... or 1, 2, 3, ... values is controlled by the properties in the Model.Options property.

#### Example

**[C#]**

```
// Hiding the A, B, C in the column headers.  
this.gridControl1.Model.Options.NumberedColHeaders = false;
```

```
// Hiding the 1, 2, 3 in the row headers.  
this.gridControl1.Model.Options.NumberedRowHeaders = false;
```

**[VB.NET]**

```
' Hiding the A, B, C in the column headers.  
Me.GridControl1.Model.Options.NumberedColHeaders = False  
  
' Hiding the 1, 2, 3 in the row headers.  
Me.gridControl1.Model.Options.NumberedRowHeaders = False
```

#### 5.1.4 How to Avoid RangeStyles being Written out to Code in a Derived GridControl

When a derived grid is dropped onto a form at design-time, the styles that are changed in the derived grid's constructor will be written out as code in the form. This serialization to code can be avoided by overriding the respective properties and by setting it to the `DesignerSerializationVisibility.Hidden`.

As an example the RangeStyles being serialized to code can be avoided by overriding the `RangeStyles` and setting it to the `DesignerSerializationVisibility.Hidden`.

**[C#]**

```
[Browsable(false),  
DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)  
]  
public new GridRangeStyleCollection RangeStyles  
{  
    get  
    {  
        return base.RangeStyles;  
    }  
    set  
    {  
        base.RangeStyles = value;  
    }  
}
```

**[VB.NET]**

```
<Browsable(False),  
DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)  
>  
Public Shadows Property RangeStyles() As GridRangeStyleCollection  
    Get  
        Return MyBase.RangeStyles  
    End Get  
    Set  
        MyBase.RangeStyles = Value  
    End Set  
End Property
```

## 5.1.5 How to Change the Appearance of a Single Header Cell

### Introduction

To make changes to individual cells (header cells or otherwise), use an indexer on the GridControl. In a GridControl with the default headers, the column headers are row zero and the row headers are column zero. Given below is the code that will change a column header.

### Example

#### [C#]

```
// Changing the font properties of the header cell.  
gridControl1[0, 3].Font.Italic = true;  
gridControl1[0, 3].Font.Bold = true;  
gridControl1[0, 3].Font.Orientation = 270;  
  
// Changing the Text Color and Text of the header cell.  
gridControl1[0, 3].TextColor = Color.Red;  
gridControl1[0, 3].Text = "Sales";
```

#### [VB .NET]

```
// Changing the font properties of the header cell.  
GridControl1(0, 3).Font.Italic = True  
GridControl1(0, 3).Font.Bold = True  
GridControl1(0, 3).Font.Orientation = 270  
  
// Changing the Text Color and Text of the header cell.
```

```
GridControl1(0, 3).TextColor = Color.Red  
GridControl1(0, 3).Text = "Sales"
```

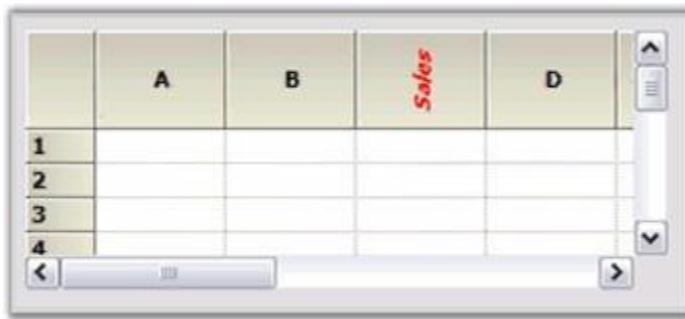


Figure 490: GridControl Showing Modified Header Cell

## 5.1.6 How to Change the Backcolor of a Column

### Introduction

The GridControl.ColStyles collection contains [GridStyleInfo](#) objects that provide column style settings for the GridControl. Changing the properties on a particular column style will affect all the cells in that row (unless a particular cell has a more specific style setting, like a [cell style](#), applied).

### Example

#### [C#]

```
// Setting the BackColor for the third column.  
gridControl1.ColStyles[3].BackColor = Color.Red;
```

#### [VB .NET]

```
' Setting the BackColor for the third column.  
GridControl1.ColStyles(3).BackColor = Color.Red
```

## 5.1.7 How to Change the Backcolor of a Single Cell

### Introduction

The style object holds all the information that affects the cells appearance. One property contained in the style object is its [BackColor](#). Use a two-parameter indexer (rowIndex, collIndex) on your GridControl object to get a reference to that particular cells style, a [GridStyleInfo](#) object.

### Example

#### [C#]

```
// Changing the BackColor of a cell in the 1st row and 3rd column.  
gridControl1[1, 3].BackColor = Color.Red;
```

#### [VB .NET]

```
' Changing the BackColor of a cell in the 1st row and 3rd column.  
GridControl1(1, 3).BackColor = Color.Red
```

## 5.1.8 How to Change the Backcolor of a Single Row

### Introduction

The **GridControl.RowStyle** collection contains [GridStyleInfo](#) objects that provide row style settings for the GridControl. Changing the properties on a particular RowStyle will affect all the cells in that row (unless a particular cell has a more specific style setting, like a [cellstyle](#), applied).

### Example

#### [C#]

```
// Setting the BackColor of the 3rd row.  
gridControl1.RowStyle[3].BackColor = Color.Red;
```

#### [VB .NET]

```
' Setting the BackColor of the 3rd row.  
GridControl1.RowStyle(3).BackColor = Color.Red
```

## 5.1.9 How to Change a Cell's Font to a Particular Font Family

### Introduction

Use the **FaceName** property of the **font** property on the style object for the cell.

```
gridControl1[rowIndex, colIndex].Font.Facename = "Arial";
```

You can also change several font properties in one statement by using the font class's SetXXX method which, will return a reference to the font object.

### Example

#### [C#]

```
// Snippet 1
gridControl1[rowIndex, colIndex].Font.Facename = "Arial";
gridControl1[rowIndex, colIndex].Font.Bold = true;
gridControl1[rowIndex, colIndex].Font.Size = 10;

// Snippet 2
gridControl1[rowIndex,
colIndex].Font.SetFacename("Arial").SetBold(true).SetSize(10);
```

#### [VB.NET]

```
' Snippet 1
gridControl1(rowIndex, colIndex).Font.Facename = "Arial"
gridControl1(rowIndex, colIndex).Font.Bold =
gridControl1(rowIndex, colIndex).Font.Size = 10

' Snippet 2
gridControl1(rowIndex,
colIndex).Font.SetFacename("Arial").SetBold(True).SetSize(10)
```

## 5.1.10 How to Change the Look of a Cell's Border

### Introduction

Use the **Borders** property of [GridStyleInfo](#) to change the style and the appearance of the grid cells border. Each border side of the cell can be configured individually with a GridBorder value. There is a BorderMargins property to control the margins on all four sides.

### Example

#### [C#]

```
// Borders on all four sides of the cell.  
GridStyleInfo style = this.gridControl1.RowStyle[1];  
this.gridControl1.RowStyle[1].Borders.All = new  
GridBorder(GridBorderStyle.Solid, Color.Red, GridBorderWeight.Thin);  
this.gridControl1.RowStyle[2].Borders.Right = new  
GridBorder(GridBorderStyle.Dotted, Color.LightBlue,  
GridBorderWeight.ExtraThick);  
this.gridControl1.RowStyle[3].Borders.Bottom = new  
GridBorder(GridBorderStyle.Solid, Color.Pink, GridBorderWeight.Medium);  
this.gridControl1.RowStyle[4].Borders.Left = new  
GridBorder(GridBorderStyle.DashDot, Color.LightGreen,  
GridBorderWeight.ExtraThick);  
this.gridControl1.RowStyle[5].Borders.Top = new  
GridBorder(GridBorderStyle.DashDotDot, Color.Purple,  
GridBorderWeight.ExtraExtraThick);  
  
// Bordermargins  
this.gridControl1.RowStyle[1].BorderMargins.Right = 20;  
this.gridControl1.RowStyle[2].BorderMargins.Left = 22;  
this.gridControl1.RowStyle[3].BorderMargins.Top = 24;  
this.gridControl1.RowStyle[4].BorderMargins.Bottom = 26;
```

#### [VB .NET]

```
' Borders on all four sides of the cell.  
Me.GridControl1.RowStyle(1).Borders.All = New  
GridBorder(GridBorderStyle.Solid, Color.Red, GridBorderWeight.Thin)  
Me.GridControl1.RowStyle(2).Borders.Right = New  
GridBorder(GridBorderStyle.Dotted, Color.LightBlue,  
GridBorderWeight.ExtraThick)  
Me.GridControl1.RowStyle(3).Borders.Bottom = New  
GridBorder(GridBorderStyle.Solid, Color.Pink, GridBorderWeight.Medium)  
Me.GridControl1.RowStyle(4).Borders.Left = New  
GridBorder(GridBorderStyle.DashDot, Color.LightGreen,  
GridBorderWeight.ExtraThick)  
Me.GridControl1.RowStyle(5).Borders.Top = New  
GridBorder(GridBorderStyle.DashDotDot, Color.Purple,  
GridBorderWeight.ExtraExtraThick)  
  
' Bordermargins
```

```
Me.GridControl1.RowStyle(1).BorderMargins.Right = 20  
Me.GridControl1.RowStyle(2).BorderMargins.Left = 22  
Me.GridControl1.RowStyle(3).BorderMargins.Top = 24  
Me.GridControl1.RowStyle(4).BorderMargins.Bottom = 26
```

## 5.1.11 How to Create a Multi-Select DropDownList Grid in a Cell

### Introduction

To have a cell that has a multi-selection dropdown grid, you must use a derived custom cell that is derived from the **GridDropDownGridCellModel** and the **GridDropDownGridCellRenderer**.

1. In the derived renderer, the code embeds the **GridControl** whose **ListBoxSelectionMode** is set to **MultiSimple**. The renderer uses the **DropDownContainerCloseDropDown** override to move the text in the selected rows of the embedded grid, into a string that is set into the style. The **CellValue** of the cell in the parent grid which, hosts this custom **celltype** lists the selected options as a hyphen delimited string for every column, and the **NewLine** delimited string for every row.

### Example

#### [C#]

```
// Creates an instance of the DropDownList Model.  
DropDownGridCellModel aModel = new  
DropDownGridCellModel(this.gridControl1.Model);  
aModel.EmbeddedGrid = GridA;  
  
// To register the DropDownGridCellModel to GridModel.  
gridControl1.CellModels.Add("MultiSelectCombo", aModel);
```

#### [VB .NET]

```
' Creates an instance of the DropDownList Model.  
Dim aModel As DropDownGridCellModel = New  
DropDownGridCellModel(Me.gridControl1.Model)  
aModel.EmbeddedGrid = GridA  
  
' To register the DropDownGridCellModel to GridModel.  
gridControl1.CellModels.Add("MultiSelectCombo", aModel)
```

2. Set the **CellType** property to "MultiSelectCombo".

## Example

[C#]

```
// Set the CellType to 'MultiSelectCombo'.
this.gridControl1[4,2].CellType = "MultiSelectCombo";
```

[VB .NET]

```
' Set the CellType to 'InPlaceRTB'.
Me.gridControl1(4,2).CellType = "MultiSelectCombo"
```

## 5.1.12 How to Create a Cell that contains Hypertext Link along with Different Formatted Texts

### Introduction

To create a cell that contains a hypertext link along with different formatted texts ,you will need to handle the derived class called InplaceRTB, follow the steps that are given below:

#### 1. InplaceRTB Class

The **RichText** cell control in the library will allow you to edit the richtext via a dropdown panel. However, it will not allow you to edit the text in-place. You must derive the **GridRichTextBoxCellModel** and the **GridRichTextBoxCellRenderer** classes in order to use an embedded **RichTextBox** in the cell and to edit it. This **textbox** will allow you to Click on the text that will open an Internet Explorer session. To use this control,you must register the new **CellControl** using the **grid.CellModels.Add** function.

## Example

[C#]

```
// Adds the cell Model to the specified type Name "InPlaceRTB".
this.gridControl1.CellModels.Add("InPlaceRTB", new
InPlaceRichTextCellModel(this.gridControl1.Model));
```

[VB .NET]

```
// Adds the cell Model to the specified type Name "InPlaceRTB".
```

```
Me.gridControl1.CellModels.Add("InPlaceRTB", New  
InPlaceRichTextCellModel(Me.gridControl1.Model))
```

2. Set the **CellType** property to "InplaceRTB".

**Example**

**[C#]**

```
// Set the CellType to 'InPlaceRTB'.  
this.gridControl1(1,1).CellType = "InPlaceRTB"
```

**[VB .NET]**

```
// Set the CellType to 'InPlaceRTB'.  
Me.gridControl1(1,1).CellType = "InPlaceRTB"
```

3. Set the **Text** property to RTF Text which contains the different text formats with information.

**Example**

**[C#]**

```
// Assigns the Text property to RTF Text.  
this.gridControl1[1,1].Text =  
@"\{\\rtf1\\ansi\\ansicpg1252\\def0\\deftab709{\\fonttbl{\\f0\\froman\\fprq2\\fch  
arset0 Times New Roman;}}"+  
@"\{\\colortbl ;\\red0\\green0\\blue255;\\red0\\green0\\blue128;}" +  
@"\{*\\generator Msftedit  
5.41.15.1507;}\\viewkind4\\uc1\\pard\\lang1033\\b\\f0\\fs24 Bold\\b0\\par"+  
@"\i Italic\\i0\\par"+  
@"\{\\field{\\*\\fldinst{HYPERLINK 'http://www.google.com/'  
}}\\fldrslt{\\cf2\\ul www.google.com}}\\cf0\\ulnone\\f0\\fs24\\par}";
```

**[VB .NET]**

```
// Assigns the text property to RTF Text.  
Me.gridControl1(1,1).Text =  
"\{\\rtf1\\ansi\\ansicpg1252\\def0\\deftab709{\\fonttbl{\\f0\\froman\\fprq2\\fch  
arset0 Times New Roman;}}"+ "\{\\colortbl  
;\\red0\\green0\\blue255;\\red0\\green0\\blue128;}" + "\{*\\generator Msftedit  
5.41.15.1507;}\\viewkind4\\uc1\\pard\\lang1033\\b\\f0\\fs24 Bold\\b0\\par"+ "\\i  
Italic\\i0\\par"+ "\{\\field{\\*\\fldinst{HYPERLINK 'http://www.google.com/'  
}}\\fldrslt{\\cf2\\ul www.google.com}}\\cf0\\ulnone\\f0\\fs24\\par}"
```

### 5.1.13 How to Exclude Hidden Rows and Column for Scrolling?

When scrolling the grid, hidden rows and columns are considered as existing rows and columns. This takes a long time to scroll the grid. To avoid this and save time, you can exclude the hidden rows and columns for scrolling. You can achieve this by using the HScrollPixel and the VScrollPixel properties. To exclude hidden columns for scrolling, set the HScrollPixel to true. Similarly, to exclude hidden rows for scrolling, set the VScrollPixel to true.

The following code illustrates how to exclude hidden columns for scrolling:

[C#]

```
private void Form1_Load(object sender, System.EventArgs e)
{
    this.gridControl1.HScrollPixel = true;
}
```

[VB]

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    Me.gridControl1.HScrollPixel = True;

End Sub
```

The following code illustrates how to exclude hidden rows for scrolling:

[C#]

```
private void Form1_Load(object sender, System.EventArgs e)
{
    this.gridControl1.VScrollPixel = true;
}
```

[VB]

```
Private Sub Form1 Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    Me.gridControl1.VScrollPixel = True;

End Sub
```

## 5.1.14 How to Get all the Data in a GridControl as an Array

### Introduction

Using an indexer to retrieve the grid[row, col].**CellValue** triggers events(like QueryCellInfo). In a **GridControl** where the data is stored in the grid, you can avoid the triggering of these events (which slow things down) by accessing the **GridData** directly.

### Example

[C#]

```
// Accessing data through the GridData object....
GridData gridData = this.gridControl1.Data;
int numRows = gridData.RowCount;
int numCols = gridData.ColCount;
Console.WriteLine("{0} rows by {1} cols", numRows, numCols);
for(int i = 1; i <= numRows; ++i)
{
    for(int j = 1; j <= numCols; ++j)
    {
        int arrayCount=0;
        if(gridData[i, j] != null)
        {
            GridStyleInfo style = new GridStyleInfo(gridData[i, j]);
            strArray[arrayCount] = style.Text;
            Console.Write(strArray[arrayCount] + " ");
            arrayCount++;
        }
        else
            Console.Write("empty");
    }
    Console.WriteLine("");
}
```

**[VB.NET]**

```
' Accessing data through the GridData object....  
Dim gridData As GridData = Me.gridControl1.Data  
Dim numRows As Integer = gridData.RowCount  
Dim numCols As Integer = gridData.ColCount  
Console.WriteLine("{0} rows by {1} cols", numRows, numCols)  
Dim i As Integer = 1  
Do While i <= numRows  
    Dim j As Integer = 1  
    Do While j <= numCols  
        Dim arrayCount As Integer = 0  
        If Not gridData(i, j) Is Nothing Then  
  
            ' Storing all the Cell values in an Array using the GridData object.  
            Dim style As GridStyleInfo = New GridStyleInfo(gridData(i, j))  
            strArray(arrayCount) = style.Text  
            Console.Write(strArray(arrayCount) & " ")  
            arrayCount += 1  
        Else  
            Console.Write("empty")  
        EndIf  
        j+=1  
    Loop  
    Console.WriteLine("")  
    i+=1  
Loop
```

### 5.1.15 How to hide the numbered row and column headers while printing or print previewing in GridControl

You have to set the **PrintRowHeader** and **PrintColHeader** properties to *False*, to hide the row and column headers while printing or print previewing in GridControl.

**[C#]**

```
private void button2_Click(object sender, EventArgs e)  
{  
    this.gridControl1.Model.Properties.PrintRowHeader = false;  
    this.gridControl1.Model.Properties.PrintColHeader = false;  
    if (this.gridControl1 != null)  
    {  
        try  
        {
```

```
GridPrintDocument pd = new GridPrintDocument(this.gridControl1, true);
PrintDialog dlg = new PrintDialog();
dlg.Document = pd;
if (dlg.ShowDialog() == DialogResult.OK)
pd.Print();
}
catch (Exception ex)
{
MessageBox.Show("An error occurred attempting to print the grid - " +
ex.Message);
}
}
```

**[VB.NET]**

```
Private Sub button2_Click(ByVal sender As Object, ByVal e As EventArgs)
Handles button2.Click
Me.gridControl1.Model.Properties.PrintRowHeader = False
Me.gridControl1.Model.Properties.PrintColHeader = False
If Me.gridControl1 IsNot Nothing Then
Try
Dim pd As GridPrintDocument = New GridPrintDocument(Me.gridControl1,
True)
Dim dlg As PrintDialog = New PrintDialog()
dlg.Document = pd
If dlg.ShowDialog() = System.Windows.Forms.DialogResult.OK Then
pd.Print()
End If
Catch ex As Exception
MessageBox.Show("An error occurred attempting to print the grid - " &
ex.Message)
End Try
End If
End Sub
```

## 5.1.16 How to Implement a Fill Paste in the Grid

### Introduction

Copy a cell value and replicate it by selecting a range. Then paste this value into that range. This is called a "Fill Paste".

You can try handling this in the **ClipboardPaste** event.

### Example

#### [C#]

```
private void gridControl1_ClipboardPaste(object sender,
Syncfusion.Windows.Forms.Grid.GridCutPasteEventArgs e)
{
    DataObject data = (DataObject)Clipboard.GetDataObject();
    string[] rows = null;
    int numRows = 0;
    int numCols = 0;

    // Get the size of the paste.
    if(data.GetDataPresent(DataFormats.Text))
    {
        string s = (string)data.GetData(DataFormats.Text);
        rows = s.Split(new char[]{'\n'});
        numRows = rows.GetLength(0);
        if(numRows > 0 && rows[numRows - 1].Length == 0)

            // Remove extra empty row if present.
        numRows--;
        if(numRows > 0)
        {
            string[] cols = rows[0].Split(new char[]{'\t'});
            numCols = cols.GetLength(0);
        }
    }

    // Paste one to many.
    if(numRows == 1 && numCols == 1 &&
!this.gridControl1.Selections.Ranges.ActiveRange.IsEmpty)
    {

this.gridControl1.ChangeCells(this.gridControl1.Selections.Ranges.ActiveRange,
                                rows[0]);
        e.Handled = true;
        e.Result = true;
    }
}
```

#### [VB.NET]

```
Private Sub gridControl1_ClipboardPaste(ByVal sender As Object, ByVal e As Syncfusion.Grid.GridCutPasteEventArgs) Handles gridControl1.ClipboardPaste
    Dim data As DataObject = CType(Clipboard.GetDataObject(),
```

```
    DataObject)
        Dim rows As String() = Nothing
        Dim numRows As Integer = 0
        Dim numCols As Integer = 0

        ' Get the size of the paste.
        If data.GetDataPresent(DataFormats.Text) Then
            Dim s As String = CStr(data.GetData(DataFormats.Text))
            rows = s.Split(New Char(){ControlChars.Lf})
            numRows = rows.GetLength(0)
            If numRows > 0 AndAlso rows(numRows - 1).Length = 0 Then

                ' Remove extra empty row if present.
                numRows -= 1
            End If
            If numRows > 0 Then
                Dim cols As String() = rows(0).Split(New
Char(){ControlChars.Tab})
                numCols = cols.GetLength(0)
            End If
        End If

        ' Paste one to many.
        If numRows = 1 AndAlso numCols = 1 AndAlso (Not
Me.gridControl1.Selections.Ranges.ActiveRange.IsEmpty) Then
            Me.gridControl1.ChangeCells(Me.gridControl1.Selections.Ranges.ActiveRange,
rows(0))
            e.Handled = True
            e.Result = True
        End If
    End Sub
```

## 5.1.17 How to Include an Icon in the Column Header

### Introduction

The GridControl will allow you to place images in cells by specifying a style.ImageIndex and style.ImageList value for the cell provided the style.CellType is either "Static" or "Text Box". So, to make your header cell hold an icon, make it "Static" and set the following properties.

### Example

[C#]

```
// GridControl
```

```
this.gridControl1[0,3].CellType = "Static";
this.gridControl1[0,3].CellAppearance = GridCellAppearance.Raised;
this.gridControl1[0,3].ImageList = this.imageList1;

// Some imagelist defined already.
// Some index in the imagelist.
this.gridControl1[0,3].ImageIndex = 1;
```

**[VB.NET]**

```
' GridControl
Me.gridControl1(0,3).CellType = "Static"
Me.gridControl1(0,3).CellAppearance = GridCellAppearance.Raised
Me.gridControl1(0,3).ImageList = imageList
Me.gridControl1(0,3).ImageIndex = 1
```

## 5.1.18 How to Optimize Pixel Scrolling in a GridControl

### Introduction

The grid's base class pixel scrolling is not optimized for large row scenarios. In the derived **GridControl**, you will have to override a couple of virtual methods to make it perform well and use binary tree structures to quickly get the row index for an absolute pixel position and vice versa.

Follow the steps that are given below to do this.

1. To Create a Custom derived GridControl and to override the following methods for Optimized Vertical scrolling.

RowIndexToVScrollPixelPos(int rowIndex)

VScrollPixelPosToRowIndex(int pixelPos, out int rowIndex, out int pixelDelta)

GetVScrollPixelHeight()

### Example

**[C#]**

```
public class DerivedGridControl : GridControl
{
    /// Get the Scroll Position for the pixel scrolling of a row.
```

```
public override int RowIndexToVScrollPixelPos(int rowIndex)
{
    // Taking separate height for column headers into account.
    rowIndex = Math.Min(rowIndex, Model.RowCount);
    if (rowIndex > 0) return (rowIndex - 1) *
Model.Rows.DefaultSize + Model.RowHeights[0];
    else
        return Model.RowHeights[0];
}

/// Get the value for the vertical pixel Position.
public override int GetVScrollPixelHeight()
{
    // To check the number of rows in the Grid.
    if (Model.RowCount == 0) return 0;

    // To return the vertical pixel Position.
    return (Model.RowCount - 1) * Model.Rows.DefaultSize +
Model.RowHeights[0];
}

/// Get the row and pixel Delta to the scroll position of the row for the specified scroll position.
public override void VScrollPixelPosToRowIndex(int pixelPos,
out int rowIndex, out int pixelDelta)
{
    if (pixelPos < pixelPos - Model.RowHeights[0])
    {
        rowIndex = 0; pixelDelta = pixelPos;
    }
    rowIndex = (pixelPos - Model.RowHeights[0]) /
Model.Rows.DefaultSize + 1; pixelDelta = (pixelPos -
Model.RowHeights[0]) % Model.Rows.DefaultSize;
}
```

**[VB.NET]**

```
Public Class DerivedGridControl : Inherits GridControl

    ' Gets the Scroll Position for the pixel scrolling of a row.
    Public Overrides Function RowIndexToVScrollPixelPos(ByVal
rowIndex As Integer) As Integer
```

```
' Taking separate height for column headers into account.  
rowIndex = Math.Min(rowIndex, Model.RowCount)  
If rowIndex > 0 Then  
    Return (rowIndex - 1) * Model.Rows.DefaultSize +  
        Model.RowHeights(0)  
Else  
    Return Model.RowHeights(0)  
End If  
End Function  
  
''' Gets the value for vertical pixel Position.  
Public Overrides Function GetVScrollPixelHeight() As Integer  
  
    ' To check the number of rows in the Grid.  
    If Model.RowCount = 0 Then  
        Return 0  
    End If  
  
    ' To return the vertical pixel Position.  
Return (Model.RowCount - 1) * Model.Rows.DefaultSize +  
    Model.RowHeights(0)  
End Function  
  
''' Gets the row and pixel Delta to the scroll position of the row for  
the specified scroll position.  
Public Overrides Sub VScrollPixelPosToRowIndex(ByVal pixelPos As  
Integer, ByRef rowIndex As  
Integer, ByRef pixelDelta As Integer)  
    If pixelPos < pixelPos - Model.RowHeights(0) Then  
        rowIndex = 0  
        pixelDelta = pixelPos  
    End If  
    rowIndex = (pixelPos - Model.RowHeights(0)) /  
        Model.Rows.DefaultSize + 1  
    pixelDelta = (pixelPos - Model.RowHeights(0)) Mod  
        Model.Rows.DefaultSize  
End Sub  
End Class
```

2. Assign the New custom control to the GridControl.

**Example**

[C#]

```
// Setting the derived GridControl to the grid.
```

```
this.gridControl1 = new VscrollOptimization.DerivedGridControl();
```

**[VB.NET]**

```
' Setting the derived GridControl to the grid.  
Me.GridControl1 = New VscrollOptimization.DerivedGridControl
```

3. Set the **VScrollPixel** property to TRUE.

**Example**

**[C#]**

```
// Enable pixel scrolling.  
this.gridControl1.VScrollPixel = true;
```

**[VB.NET]**

```
' Enable pixel scrolling.  
Me.GridControl1.VScrollPixel = True
```

## 5.1.19 How to pre-select a checkbox inside a grid cell

You need to set a style member called **CheckBoxOptions** for a particular grid cell, for this purpose. CheckBoxOptions.CheckedValue is used to specify what value should be given in the CellValue to make the checkbox checked.

**Example**

If you want "true" to be checked and "false" to be unchecked state then use the below code.

**[C#]**

```
//Sets the cell type as Checkbox.  
this.gridControl1[1,1].CellType = "CheckBox";  
  
//Sets the Checked and Unchecked values for the checkbox.  
this.gridControl1[1,1].CheckBoxOptions.CheckedValue = "true";  
this.gridControl1[1,1].CheckBoxOptions.UncheckedValue = "false";
```

```
//Sets the type of the cell value as bool.  
this.gridControl1[1,1].CellValueType = typeof(bool);  
  
//Sets the Cell value.  
this.gridControl1[1,1].CellValue = true;
```

**[VB.NET]**

```
'Sets the cell type as Checkbox.  
Me.gridControl1(1, 1).CellType = "CheckBox"  
  
'Sets the Checked and Unchecked values for the checkbox.  
Me.gridControl1(1, 1).CheckBoxOptions.CheckedValue = "true"  
Me.gridControl1(1, 1).CheckBoxOptions.UncheckedValue = "false"  
  
'Sets the type of the cell value as bool.  
Me.gridControl1(1, 1).CellValueType = GetType(Boolean)  
  
'Sets the Cell value.  
Me.gridControl1(1, 1).CellValue = True
```

## 5.1.20 How to Retrieve the Text From a Cell

### Introduction

To retrieve text from a cell, simply use the [Text](#) property of the cells style object which, is obtained through an indexer in the **GridControl**.

### Example

**[C#]**

```
// Access the cell's Text property to retrieve the text from the cell.  
string cellText = gridControl1[2, 3].Text;
```

**[VB.NET]**

```
' Access the cell's Text property to retrieve the text from the cell.  
Dim cellText As String = gridControl1(2,3).Text
```



**Note:**

Depending upon exactly what object is stored in the [CellValue](#) property, you may have to do additional work to retrieve a usable value from the style. There is also a [FormattedText](#) property that may give you a different value. One example would be if the cell is a combobox cell type that is using the [DisplayMember](#) and [ValueMember](#) properties to show different values from the ones being stored in the [GridControl](#) (as in foreign key look tables). In this case, the [Text](#) property is the [ValueMember](#) value and the [FormattedText](#) property is the [DisplayMember](#) property.

### 5.1.21 How to save the contents of a Grid in memory rather than a file system to export to another grid

You can save the contents of a grid in memory rather than writing to a file system. This can be done using the [SaveSoap](#) or [SaveBinary](#) method of [GridControl](#). The below code snippet illustrates how this can be done.

#### [C#]

```
MemoryStream s = new MemoryStream();
gridControl1.Model.SaveSoap(s); // or gridControl1.Model.SaveBinary(s);
s.Position = 0;
gridControl1.Model = GridModel.LoadSoap(s);
gridControl2.Model = GridModel.LoadSoap(s);
```

#### [VB .NET]

```
Dim s As MemoryStream = New MemoryStream()
gridControl1.Model.SaveSoap(s) ' or gridControl1.Model.SaveBinary(s);
s.Position = 0
gridControl1.Model = GridModel.LoadSoap(s)
gridControl2.Model = GridModel.LoadSoap(s)
```

### 5.1.22 How to Set the Cell Properties for a Range of Cells

#### Introduction

Use the GridControl's **ChangeCells** method by passing it a [GridRangeInfo](#) object to change the appearance of a range of cells.

### Example

To set the **backcolor** and **textcolor** for a range of cells, use the below given code snippet.

[C#]

```
// Style settings.  
GridStyleInfo style = new GridStyleInfo();  
style.TextColor = Color.Red;  
style.BackColor = Color.LightBlue;  
  
// Modifying a range of cells.  
gridControl1.ChangeCells(GridRangeInfo.Cells(1, 1, 4, 5), style);
```

[VB .NET]

```
' Style settings.  
Dim style As New GridStyleInfo()  
style.TextColor = Color.Red  
style.BackColor = Color.LightBlue  
  
' Modifying a range of cells.  
GridControl1.ChangeCells(GridRangeInfo.Cells(1, 1, 4, 5), style)
```

## 5.1.23 How to Set the Number of Rows / Columns

### Introduction

Dynamically changing the **RowCount** or **ColCount** properties while a **GridControl** is being displayed is an efficient way to add or remove rows and / or columns from a GridControl. Using the designer, set the grids RowCount and ColCount properties. From code, set these properties after the call to InitializeComponent in the form's constructor (or anytime later in your code after the GridControl has been created).

### Example

[C#]

```
public Form1()
{
    //
    // Required for Windows Form Designer support.
    //
    InitializeComponent();

    //
    // TODO: Add any constructor code after InitializeComponent
    // call.
    //

    // Set the number of rows.
    gridControl1.RowCount = 20;

    // Set the number of columns.
    gridControl1.ColCount = 200;
}
```

**[VB .NET]**

```
Public Sub New()
    MyBase.New()

    ' This call is required by the Windows Form Designer.
    InitializeComponent()

    ' Add any initialization after the InitializeComponent() call.

    ' Set the number of rows.
    GridControl1.RowCount = 20

    ' Set the number of columns.
    GridControl1.ColCount = 200

End Sub
```

## 5.1.24 How to Set the Height of a Row

### Introduction

Changing a row's height is simple whether you are using the designer or code. From the designer, use the **RowHeightEntries** collection. To explicitly set the height of the particular row from code, use the **GridControl.RowHeights** collection.

## Example

[C#]

```
// Set height of row 3 to 40.  
this.gridControl1.RowHeights[3] = 40;  
  
// Set height of header row 30.  
this.gridControl1.RowHeights[0] = 30;
```

[VB .NET]

```
' Set height of row 3 to 40.  
Me.GridControl1.RowHeights(3) = 40  
  
' Set height of header row 30.  
Me.GridControl1.RowHeights(0) = 30
```

## 5.1.25 How to Set a Value into a Cell

### Introduction

A style object holds all the information that affects the cells appearance. One property contained in the style object is its

CellValue. It is this property that you have to set in order to place a value in a cell. Another property called GridStyleInfo.Text, will allow you to set the value of a cell using a string. Text and [CellValue](#) are related properties and setting one of these properties will also set the other.

### Example

Use a two-parameter indexer (rowIndex, colIndex) on your **GridControl** object to get a reference to that particular cells style, a **GridStyleInfo** object.

[C#]

```
// Setting the CellValues of the cell.  
int rowIndex = 1;  
int colIndex = 2;  
gridControl1[rowIndex, colIndex].CellValue = "PI";  
rowIndex++;
```

```
// Sets a different value in the next row.  
gridControl1[rowIndex, colIndex].CellValue = 3.14159;
```

**[VB.NET]**

```
' Setting the CellValues of the cell.  
Dim rowIndex As Integer = 1  
Dim colIndex As Integer = 2  
GridControl1(rowIndex, colIndex).CellValue = "PI"  
rowIndex = rowIndex + 1  
  
' Sets a different value in the next row.  
GridControl1(rowIndex, colIndex).CellValue = 3.14159
```



**Note:**

*Using an indexer on the GridControl to set values will trigger of several notification events that listeners (and the GridControl itself) can use to monitor the state of the GridControl. These events may slow things down if you have a lot of data to move into the GridControl. In this case, you can want to employ a technique that will avoid these events.*

There are several ways to do this; if your data is in some form supported by the **GridControl.PopulateValues** method, you can use that method to move such data into the GridControl. Another option is to use the **GridControl.SetCellInfo** method, by passing it appropriate parameters to avoid the notification events. But, if you are only setting a few values or you need the notification events to be raised, then using an indexer is a straight-forward way to accomplish this task.

## 5.1.26 How to Set the Text in a Header Cell

### Introduction

In a **GridControl**, values in header cells are set just as in any other cell.

### Example

Use an indexer on your GridControl with the row index set to 0.

**[C#]**

```
// Setting Text property in the 5th column header cell.  
gridControl1[0, 5].Text = "HeaderTextForColumn5";
```

**[VB.NET]**

```
' Setting Text property in the 5th column header cell.  
GridControl1(0, 5).Text = "HeaderTextForColumn5"
```

## 5.1.27 How to Set the Width of a Column

### Introduction

Changing a column's width is simple whether you are using the designer or code. In the designer, use the **ColWidthsEntries** collection. In code, use the **GridControl.ColWidths** collection to specify the width of a column.

### Example

**[C#]**

```
// Set size of column 3 to 250.  
this.gridControl1.ColWidths[3] = 250;
```

**[VB.NET]**

```
' Set size of column 3 to 250.  
Me.GridControl1.ColWidths(3) = 250
```

## 5.1.28 How to Set the Text Color that Appears in a Cell

### Introduction

Harnessing the ability to customize **text** color to your application allows you endless possibilities. Use the **TextColor** property of the cells style and set it to color value.

### Example

**[C#]**

```
// Modifying the TextColor of a cell.  
gridControl1[rowIndex, colIndex].TextColor = Color.Red;
```

**[VB .NET]**

```
' Modifying the TextColor of a cell.  
GridControl1(rowIndex, colIndex).TextColor = Color.Red
```

## 5.1.29 How to Set Transparent Backcolor for a GridControl

### Introduction

Setting the transparent [backcolor](#) for a **GridControl** can be done easily with simple code.

### Example

**[C#]**

```
// Set up Transparent Background.  
this.gridControl1.SupportsTransparentBackColor = true;  
this.gridControl1.TransparentBackground = true;  
  
// Set Color for the Transparent Background.  
this.gridControl1.Properties.BackgroundColor = Color.FromArgb(0, 1, 1,  
1);  
this.gridControl1.BackColor = Color.FromArgb(0, SystemColors.Window);
```

**[VB .NET]**

```
' Set up Transparent Background.  
Me.gridControl1.SupportsTransparentBackColor = True  
Me.gridControl1.TransparentBackground = True  
  
' Set Color for the Transparent Background.  
Me.gridControl1.Properties.BackgroundColor = Color.FromArgb(0, 1, 1, 1)  
Me.gridControl1.BackColor = Color.FromArgb(0, SystemColors.Window)
```

### 5.1.30 How to Show a ContextMenu in the Center of CurrentCell or Selected Cells Irrespective of the Mouse Click

This can be achieved by handling the Grid's MouseDown event. In the event handler, the center location of the current cell or the selected range is calculated and the context menu is shown after moving the cursor to the center point. Below is the code snippet.

**[C#]**

```
private void gridControl1_MouseDown(object sender, MouseEventArgs e)
{
    if(e.Button == MouseButtons.Right)
    {
        GridCurrentCell cc = this.gridControl1.CurrentCell;
        Point pt = new Point(e.X, e.Y); // to retrieve mouse position
        if(this.gridControl1.Selections.Ranges.Count > 0) // Selecting more than one Cell
        {
            pt =
GridUtil.CenterPoint(this.gridControl1.RangeInfoToRectangle(this.gridControl1.Selections.Ranges.ActiveRange));
            Cursor.Position = this.gridControl1.PointToScreen(pt);
        }
        else if(cc.HasCurrentCell) // Single Cell
        {
            pt =
GridUtil.CenterPoint(this.gridControl1.RangeInfoToRectangle(cc.RangeInfo));
            Cursor.Position = this.gridControl1.PointToScreen(pt);
        }
        this.contextMenu1.Show(this.gridControl1, pt); // to Show ContextMenu
    }
}
```

**[VB.NET]**

```
Private Sub gridControl1_MouseDown(ByVal sender As Object, ByVal e As MouseEventArgs)
If e.Button = MouseButtons.Right Then
Dim cc As GridCurrentCell = Me.gridControl1.CurrentCell
Dim pt As Point = New Point(e.X, e.Y) 'To get the mouse position
If Me.gridControl1.Selections.Ranges.Count > 0 Then
pt =
GridUtil.CenterPoint(Me.gridControl1.RangeInfoToRectangle(Me.gridControl1.Selections.Ranges.ActiveRange))
```

```
Cursor.Position = Me.gridControl1.PointToScreen(pt)
ElseIf cc.HasCurrentCell Then 'Single Cell
pt =
GridUtil.CenterPoint(Me.gridControl1.RangeInfoToRectangle(cc.RangeInfo))
)
Cursor.Position = Me.gridControl1.PointToScreen(pt)
End If
Me.contextMenu1.Show(Me.gridControl1, pt) 'to Show ContextMenu
End If
End Sub
```

### 5.1.31 How to Show Multiple Images in a Cell

#### Introduction

To show multiple Images in a cell , you need to handle the **CellDrawn** Event. Follow the steps that are given below to achieve this.

1. Sets the **CellType** to GridCellTypeName.Image.

#### Example

##### [C#]

```
// To set the CellType to Image Type.
this.gridControl1[3,3].CellType = GridCellTypeName.Image;
```

##### [VB .NET]

```
' To set the CellType to Image Type.
Me.gridControl1(3,3).CellType = GridCellTypeName.Image
```

2. In the CellDrawn EventHandler, draw the Combinedbitmap using the **DrawImage** function.

#### Example

**[C#]**

```
private void gridControl1_CellDrawn(object sender,
Syncfusion.Windows.Forms.Grid.GridDrawCellEventArgs e)
{
    if( e.RowIndex == 3 && e.ColIndex == 3)
    {
        // Draws the image to the Cell.
        e.Graphics.DrawImage(bitmap ,e.Bounds.X,e.Bounds.Y );

    }
}
```

**[VB .NET]**

```
Private Sub gridControl1_CellDrawn(ByVal sender As Object, ByVal e As
Syncfusion.Windows.Forms.Grid.GridDrawCellEventArgs)
    If e.RowIndex = 3 And e.ColIndex = 3 Then
        ' Draws the image to the Cell.
        e.Graphics.DrawImage(bitmap,e.Bounds.X,e.Bounds.Y)
    End If
End Sub
```

## 5.1.32 How to Swap Rows and Columns

### Introduction

This can be done in the **GridControl** by handling the virtual events **QueryCellInfo**, **SaveCellInfo**, **QueryRowCount**, **QueryColCount**. Here the **GridControl.Data** property is used.

### Example

**[C#]**

```
// In the QueryCellInfo handler.
e.Style.ModifyStyle(this.gridControl1.Data[e.ColIndex, e.RowIndex],
Syncfusion.Styles.StyleModifyType.Override);

// In the SaveCellInfo handler.
this.gridControl1.Data[e.ColIndex, e.RowIndex] = e.Style.Store;

// In the QueryRowCount handler.
```

```
e.Count = this.gridControl1.Data.ColCount;  
  
// In the QueryColCount handler.  
e.Count = this.gridControl1.Data.RowCount;
```

**[VB.NET]**

```
' In the QueryCellInfo handler.  
e.Style.ModifyStyle(Me.gridControl1.Data(e.ColIndex, e.RowIndex),  
Syncfusion.Styles.StyleModifyType.Override)  
  
' In the SaveCellInfo handler.  
Me.gridControl1.Data(e.ColIndex, e.RowIndex) = e.Style.Store  
  
' In the QueryRowCount handler.  
e.Count = Me.gridControl1.Data.ColCount  
  
' In the QueryColCount handler.  
e.Count = Me.gridControl1.Data.RowCount
```

### 5.1.33 How to Synchronize the Selection of Multiple Grids

This can be achieved by using the SelectionChanged Event. In this event, the selection range of the source grids can be copied into the target grid in order to have synchronization between the grids. The TopRowChanged event and the LeftColChanged event can be used to synchronize the top row index and the left column index changes, when the grid is scrolled. Refer to the sample for more details. Here is the code snippet which, explains how selection change in one grid affects the other grid.

**[C#]**

```
void gridControl1_SelectionChanged(object sender,  
GridSelectionChangedEventArgs e)  
{  
    Synchronize_SelectionChanged(gridControl2, gridControl1, e);  
}  
void gridControl2_SelectionChanged(object sender,  
GridSelectionChangedEventArgs e)  
{  
    Synchronize_SelectionChanged(gridControl1, gridControl2, e);  
}  
void Synchronize_SelectionChanged(GridControl targetGrid, GridControl
```

```
sourceGrid, GridSelectionChangedEventArgs e)
{
    int sourceCount = sourceGrid.Selections.Ranges.Count;
    int targetCount = targetGrid.Selections.Ranges.Count;
    int maxCount = Math.Max(sourceCount, targetCount);

    // Invalidate any ranges that were changed.
    for (int n = 0; n < maxCount; n++)
    {
        GridRangeInfo newRange = n < sourceCount ?
sourceGrid.Selections.Ranges[n] : GridRangeInfo.Empty;
        GridRangeInfo oldRange = n < targetCount ?
targetGrid.Selections.Ranges[n] : GridRangeInfo.Empty;
        if (!oldRange.Equals(newRange))
        {
            Console.WriteLine("{0} - {1}", oldRange, newRange);
            targetGrid.InvalidateRange(oldRange);
            targetGrid.InvalidateRange(newRange);
        }
    }
    // Update underlying data structure - does not cause invalidate / paint
    ...
    targetGrid.Selections.Ranges.Clear();
    GridRangeInfo[] ranges = new GridRangeInfo[sourceCount];
    sourceGrid.Selections.Ranges.CopyTo(ranges, 0);
    targetGrid.Selections.Ranges.AddRange(ranges);
}
```

**[VB.NET]**

```
Sub gridControl1_SelectionChanged(ByVal sender As Object, ByVal e As
GridSelectionChangedEventArgs)
Synchronize_SelectionChanged(gridControl2, gridControl1, e)
End Sub 'gridControl1_SelectionChanged

Sub gridControl2_SelectionChanged(ByVal sender As Object, ByVal e As
GridSelectionChangedEventArgs)
Synchronize_SelectionChanged(gridControl1, gridControl2, e)
End Sub 'gridControl2_SelectionChanged

Sub Synchronize_SelectionChanged(ByVal targetGrid As GridControl, ByVal
sourceGrid As GridControl, ByVal e As GridSelectionChangedEventArgs)

Dim sourceCount As Integer = sourceGrid.Selections.Ranges.Count
Dim targetCount As Integer = targetGrid.Selections.Ranges.Count
Dim maxCount As Integer = Math.Max(sourceCount, targetCount)
```

```
' Invalidate any ranges that were changed.  
Dim n As Integer  
For n = 0 To maxCount - 1  
Dim newRange As GridRangeInfo  
Dim oldRange As GridRangeInfo  
If (n < sourceCount) Then  
newRange = sourceGrid.Selections.Ranges(n)  
Else  
newRange = GridRangeInfo.Empty  
End If  
If (n < targetCount) Then  
oldRange = targetGrid.Selections.Ranges(n)  
Else  
oldRange = GridRangeInfo.Empty  
End If  
If Not oldRange.Equals(newRange) Then  
Console.WriteLine("{0} - {1}", oldRange, newRange)  
targetGrid.InvalidateRange(oldRange)  
targetGrid.InvalidateRange(newRange)  
End If  
Next n  
' Update underlying data structure - does not cause invalidate / paint  
...  
targetGrid.Selections.Ranges.Clear()  
Dim ranges(sourceCount) As GridRangeInfo  
sourceGrid.Selections.Ranges.CopyTo(ranges, 0)  
targetGrid.Selections.Ranges.AddRange(ranges)  
End Sub 'Synchronize_SelectionChanged
```

### 5.1.34 How to Use a Combo Box in a Cell

#### Introduction

The control type of a cell is part of the cell style and is determined by the **GridStyleInfo.CellType** property. The items shown in the dropdown list can be provided in two ways.

- Create a StringCollection object holding your choices and then set this StringCollection in the **GridStyleInfo.ChoiceList** property for the cell.
- Have an IList object that holds object entries that have public properties (such as a **DataTable** object with its columns serving as public properties).

In the second case, use the `GridStyleInfo.DataSource`, `DisplayMember` and `ValueMember` properties to set the datasource for the drop list. In addition to setting the `CellType`, `ChoiceList`, `datasource`, `DisplayMember` and `ValueMember`, the `DropDownStyle` property of the [GridStyleInfo](#) controls the editing behavior of the combo box cell. You can also use the `GridStyleInfo.ShowButton` property to control when the combo box button is visible.

### Example

Here is the code that will set cells 4,2 to a combo box by setting the items in the combo box through the styles `ChoiceList` property.

#### [C#]

```
// Required to access the StringCollection.  
using System.Collections.Specialized;  
  
//...  
// Create the list.  
StringCollection items = new StringCollection();  
items.AddRange(new string[]{"One", "Two", "Three", "Four", "Five"});  
  
// Set the style properties.  
GridStyleInfo style = gridControl1[4, 2];  
style.CellType = "ComboBox";  
style.ChoiceList = items;  
styleCellValue = "Five";  
  
// True dropdown - no editing.  
style.DropDownStyle = GridDropDownStyle.Exclusive;
```

#### [VB.NET]

```
' Required to access the StringCollection.  
Imports System.Collections.Specialized  
  
'...  
' Create the list.  
Dim items As New StringCollection  
items.AddRange(New String() {"One", "Two", "Three", "Four", "Five"})  
  
' Set the style properties.  
Dim style As GridStyleInfo = GridControl1(4, 2)  
style.CellType = "ComboBox"  
style.ChoiceList = items  
styleCellValue = "Five"
```

```
' True dropdown - no editing.  
style.DropDownStyle = GridDropDownStyle.Exclusive
```

Here is the code that will set cells 4,2 to a combo box by setting the items in the combo box through a DataTable datasource.

**[C#]**

```
// Assume this.dt is a DataTable object with at least 2 columns named  
// "id" and "display".  
  
// Set the style properties.  
GridStyleInfo style = gridControl1[4, 2];  
style.CellType = "ComboBox";  
style.DataSource = dt;  
style.DisplayMember = "display";  
  
// Displayed in the grid cell.  
style.ValueMember = "id";  
  
// Value in the grid cell.  
style.DropDownStyle = GridDropDownStyle.AutoComplete;
```

**[VB.NET]**

```
' Assume this.dt is a DataTable object with at least 2 columns named  
// "id" and "display".  
  
' Set the style properties.  
Dim style As GridStyleInfo = GridControl1(4, 2)  
style.CellType = "ComboBox"  
style.DataSource = dt  
style.DisplayMember = "display"  
  
' Displayed in the grid cell.  
style.ValueMember = "id"  
  
' Value in the grid cell.  
style.DropDownStyle = GridDropDownStyle.AutoComplete
```

## 5.1.35 How to Use a ColorEdit Control in a Cell and Retrieve its Value

### Introduction

It is simple to use the ColorEdit control to specify a cell's value. Just set the [CellType](#) property in the cell style to "ColorEdit" and set the text property to an appropriate value.

### Example

#### [C#]

```
// Set the controltype.
gridControl1[4, 4].CellType = "ColorEdit";

// Set initial value to Color.Aqua or to set a RGB color, use something
// like
gridControl1[4, 4].Text = "2, 12, 255";

// Set initial value to RGB(2,12,255).
gridControl1[4, 4].Text = "Aqua";

//....
// To retrieve a color object from this cell, use code such as
Color c =
(Color)System.ComponentModel.TypeDescriptor.GetConverter(typeof(Color))
.ConvertFromString(gridControl1[4, 4].Text);
```

#### [VB.NET]

```
' Set controltype .
GridControl1(4, 4).CellType = "ColorEdit"

' Set initial value to Color.Aqua or to set a RGB color, use something
like
gridControl1[4, 4].Text = "2, 12, 255";

// Set initial value to RGB(2,12,255)
GridControl1(4, 4).Text = "Aqua"

'....
' To retrieve a color object from this cell, use code such as
Dim c As Color =
 CType(System.ComponentModel.TypeDescriptor.GetConverter(GetType(Color))
 .ConvertFromString(GridControl1(4, 4).Text), Color)
```

## 5.1.36 How to Use a PushButton in a Cell and Catch the User Clicking It

### Introduction

Set the [CellType](#) property in the cell style to "PushButton" and handle the grids CellButtonClicked event. Use the Description property of the cell style to specify the text that is to be displayed on the button.

### Example

#### [C#]

```
gridControl1[4, 6].CellType = "PushButton";
gridControl1[4, 6].Description = "Push Me!";

gridControl1.CellButtonClicked += new
GridCellButtonClickedEventHandler(gridControl1_CellButtonClicked);

//...
private void gridControl1_CellButtonClicked(object sender,
GridCellButtonClickedEventArgs e)
{
    string s = string.Format("You clicked ({0},{1}).", e.RowIndex,
e.ColIndex);
    MessageBox.Show(s);
}
```

#### [VB .NET]

```
GridControl1(4, 6).CellType = "PushButton"
GridControl1(4, 6).Description = "Push Me!"

AddHandler GridControl1.CellButtonClicked, AddressOf
gridControl1_CellButtonClicked

'...
Private Sub gridControl1_CellButtonClicked(sender As Object, e As
GridCellButtonClickedEventArgs)
    Dim s As String = String.Format("You clicked ({0},{1}).",
e.RowIndex, e.ColIndex)
    MessageBox.Show(s)
```

```
' GridControl1_CellButtonClicked  
End Sub
```

## 5.1.37 What is the Difference between TextAlign, HorizontalAlignment and VerticalAlignment?

### Introduction

**TextAlign** is set when the description of embedded controls are to be aligned to the left or right. **HorizontalAlignment** is set when the cell value is to be aligned either left or right or center of the cell. **VerticalAlignment** is set when the cell value is to be aligned either top or bottom or middle of the cell.

### Example

#### [C#]

```
// Right align the cell values of column 3 horizontally.  
this.gridControl1.ColStyles[3].HorizontalAlignment =  
GridHorizontalAlignment.Right;  
  
// Right align the embedded controls in column 6.  
this.gridControl1.ColStyles[6].TextAlign = GridTextAlign.Right;  
  
// Align the cell values of column 8 at the bottom.  
this.gridControl1.ColStyles[8].VerticalAlignment =  
GridVerticalAlignment.Bottom;
```

#### [VB.NET]

```
' Right align the cell values of column 3 horizontally.  
Me.gridControl1.ColStyles(3).HorizontalAlignment =  
GridHorizontalAlignment.Right  
  
' Right align the embedded controls in column 6.  
Me.gridControl1.ColStyles(6).TextAlign = GridTextAlign.Right  
  
' Align the cell values of column 8 at the bottom.  
Me.gridControl1.ColStyles(8).VerticalAlignment =  
GridVerticalAlignment.Bottom
```

### 5.1.38 How can we make the Current Row Bold?

To make the current row bold, use the below code sample.

[C#]

```
// There is no current row at this point, so the refresh will remove  
the bold text.  
private void gridControl1_CurrentCellDeactivated(object sender,  
GridCurrentCellEventArgs e)  
{  
    GridControlBase grid = gridControl1;  
    GridCurrentCell cc = gridControl1.CurrentCell;  
  
    // Check if Deactivate is called stand-alone or called from  
    MoveTo and the row is moving.  
    if (!cc.IsInMoveTo || cc.MoveToRowIndex != cc.MoveFromRowIndex)  
    {  
        grid.RefreshRange(GridRangeInfo.Row(e.RowIndex),  
GridRangeOptions.MergeAllSpannedCells);  
    }  
}  
  
// There is a current row at this point, so the refresh causes the  
current row to have bold text.  
private void gridControl1_CurrentCellActivated(object sender,  
System.EventArgs e)  
{  
    GridControlBase grid = gridControl1;  
    GridCurrentCell cc = gridControl1.CurrentCell;  
  
    // Check if Activate is called stand-alone or called from  
    MoveTo and the row is moving.  
    if (!cc.IsInMoveTo || cc.MoveToRowIndex != cc.MoveFromRowIndex  
        || !cc.MoveFromActiveState)  
    {  
        grid.RefreshRange(GridRangeInfo.Row(cc.RowIndex),  
GridRangeOptions.MergeAllSpannedCells);  
    }  
}  
  
// Dynamically modify the style just before it is used in drawing.  
private void gridControl1_PreparesViewStyleInfo(object sender,  
GridPrepareViewStyleInfoEventArgs e)  
{
```

```
GridControlBase grid = gridControl1;
GridCurrentCell cc = gridControl1.CurrentCell;

if (e.RowIndex > grid.Model.Rows.HeaderCount && e.ColumnIndex >
grid.Model.Cols.HeaderCount
    && cc.HasCurrentCellAt(e.RowIndex))
{
    e.Style.Font.Bold = true;
}
}
```

**[VB.NET]**

```
// Add the handlers...
AddHandler Me.gridControl1.CurrentCellDeactivated, New
GridCurrentCellDeactivatedEventHandler(AddressOf
gridControl1_CurrentCellDeactivated)
AddHandler Me.gridControl1.PrepareViewStyleInfo, New
GridPrepareViewStyleInfoEventHandler(AddressOf
gridControl1_PreparesViewStyleInfo)
AddHandler Me.gridControl1.CurrentCellActivated, New
EventHandler(AddressOf gridControl1_CurrentCellActivated)

        ' There is no current row at this point, so the refresh will
remove the bold text.
Private Sub gridControl1_CurrentCellDeactivated(ByVal sender As
Object, ByVal e As GridCurrentCellDeactivatedEventArgs)
    Dim grid As GridControlBase
    grid = gridControl1
    Dim cc As GridCurrentCell
    cc = gridControl1.CurrentCell
    If (Not (cc.IsInMoveTo) _
        OrElse (cc.MoveToRowIndex <>
cc.MoveFromRowIndex)) Then
        grid.RefreshRange(GridRangeInfo.Row(e.RowIndex),
GridRangeOptions.MergeAllSpannedCells)
    End If
End Sub

        ' There is a current row at this point, so the refresh causes
the current row to have bold text.
Private Sub gridControl1_CurrentCellActivated(ByVal sender As
Object, ByVal e As EventArgs)
    Dim grid As GridControlBase
    grid = gridControl1
    Dim cc As GridCurrentCell
    cc = gridControl1.CurrentCell
```

```

    If ((Not (cc.IsInMoveTo) _
        OrElse (cc.MoveToRowIndex <> cc.MoveFromRowIndex))

    --
        OrElse Not (cc.MoveFromActiveState)) Then
            grid.RefreshRange(GridRangeInfo.Row(cc.RowIndex),
GridRangeOptions.MergeAllSpannedCells)
    End If
End Sub

    ' Dynamically modify the style just before it is used in
drawing.

    Private Sub gridControl1_PrepAreaViewStyleInfo(ByVal sender As
Object, ByVal e As GridPrepareViewStyleInfoEventArgs)
        Dim grid As GridControlBase
        grid = gridControl1
        Dim cc As GridCurrentCell
        cc = gridControl1.CurrentCell
        If (((e.RowIndex > grid.Model.Rows.HeaderCount) _
            AndAlso (e.ColumnIndex >
grid.Model.Cols.HeaderCount)) _
            AndAlso cc.HasCurrentCellAt(e.RowIndex)) Then
            e.Style.Font.Bold = True
        End If
    End Sub

```

### 5.1.39 How to save the ComboBox cell value instantly after the dropdown is closed?

To save the ComboBox cell value immediately after the dropdown is closed; the **SaveCellInfo** event must be triggered. To initiate the SaveCellInfo event, **EndEdit()** has to be called. Also, in the CurrentCellCloseDropDown event the **CurrentCell.EndEdit()** is called.

```

[C#]
//Code...
this.gridControl1.CurrentCellCloseDropDown += new
PopupClosedEventHandler(grid_CurrentCellCloseDropDown);
}

void grid_CurrentCellCloseDropDown(object sender, PopupClosedEventArgs
e)
{
    this.grid.CurrentCell.EndEdit();
}
//Code...

```

**[VB]**

```
'code...
AddHandler gridControl1.CurrentCellCloseDropDown, AddressOf
grid_CurrentCellCloseDropDown
End Sub

Private Sub grid_CurrentCellCloseDropDown(ByVal sender As Object, ByVal e
As PopupClosedEventArgs)
Me.grid.CurrentCell.EndEdit()
End Sub

'code...
```

## 5.2 GridDataBoundGrid

The GridDataBoundGrid is a column oriented grid bound to a datasource. In this grid, you can easily set the properties on a column by column basis but, making changes on a cell by cell basis will require the use of events. The tasks and solutions discussed in this section are specific to the GridDataBoundGrid.

### 5.2.1 How to Change the Appearance of a Single Header Cell

#### Introduction

To make changes to individual cells (header cells or otherwise), you must implement the **PrepareViewStyleInfo** event.

#### Example

**[C#]**

```
private void gridDataBoundGrid1_PreparesViewStyleInfo(object sender,
GridPrepareViewStyleInfoEventArgs e)
{
    if(e.ColumnIndex == 2 && e.RowIndex == 0)
    {
        // Change Font style, Font size, orientation, Text color
        // of column header 3.
```

```
        e.Style.Font.Italic = true;
        e.Style.Font.Bold = true;
        e.Style.Font.Orientation = 270;
        e.Style.TextColor = Color.Red;
    }
}
```

**[VB.NET]**

```
Private Sub gridDataBoundGrid1_PreparesViewStyleInfo(ByVal sender As Object, ByVal e As GridPrepareViewStyleInfoEventArgs)
    If e.ColumnIndex = 2 And e.RowIndex = 0 Then

        ' Change Font style, Font size, orientation, Text color of
        ' column header 3.
        e.Style.Font.Italic = True
        e.Style.Font.Bold = True
        e.Style.Font.Orientation = 270
        e.Style.TextColor = Color.Red
    End If
End Sub
```

## 5.2.2 How to Change the Backcolor of a Column

### Introduction

The GridDataBoundGrid maintains a collection of **GridBoundColumn** objects that will allow you to set **column** properties like **backcolor**, **textcolor**, **font**, etc. Either from code or at design-time, you can explicitly add GridBoundColumns to the `this.gridDataBoundGrid1.GridBoundColumns` property. If you do not explicitly add GridBoundColumns to this collection, then the grid will generate an internal set of columns that you can use, `this.gridDataBoundGrid1.Binder.InternalColumns`.

### Example

To change the backcolor of a column named "Price", use the code given below.

**[C#]**

```
// If you have added GridBoundColumns.
this.gridDataBoundGrid1.GridBoundColumns["Price"].StyleInfo.BackColor =
Color.Red;
```

```
// If you haven't explicitly added GridBoundColumns.  
this.gridDataBoundGrid1.Binder.InternalColumns["Price"].StyleInfo.BackColor = Color.Red;
```

**[VB.NET]**

```
' If you have added GridBoundColumns.  
Me.gridDataBoundGrid1.GridBoundColumns("Price").StyleInfo.BackColor =  
Color.Red  
  
' If you haven't explicitly added GridBoundColumns.  
Me.gridDataBoundGrid1.Binder.InternalColumns("Price").StyleInfo.BackColor =  
Color.Red
```

### 5.2.3 How to Change the Backcolor of a Single Cell

#### Introduction

In a GridDataBoundGrid, you cannot set cell specific properties like **backcolor** (other than **CellValue** or **text**) using an indexer. The reason is that in a GridDataBoundGrid, the only data storage is the bound datasource which, only holds a single value per cell. It does not hold **textcolor**, **backcolor** or any of the other many cell specific properties that are found in a **GridStyleInfo** object.

This code does not work.

**[C#]**

```
// Set the backcolor of cell(8,10).  
this.gridDataBoundGrid1[8, 10].BackColor = Color.Red;
```

**[VB.NET]**

```
' Set the backcolor of cell(8,10).  
Me.gridDataBoundGrid1(8, 10).BackColor = Color.Red;
```

So, in order to set cell specific properties in a GridDataBoundGrid, you must catch the **PrepareViewStyleInfo** event (or **Model.QueryCellInfo** event). In your handler, check **e.RowIndex** and **e.ColumnIndex** and if these point to the cell you want to change, set **e.Style** to the value you want.

## Example

[C#]

```
private void gridDataBoundGrid1_PreparesViewStyleInfo(object sender,
GridPrepareViewStyleEventArgs e)
{
    if(e.ColumnIndex == 2 && e.RowIndex == 2)
    {
        e.Style.BackColor = Color.Red;
    }
}
```

[VB .NET]

```
Private Sub gridDataBoundGrid1_PreparesViewStyleInfo(ByVal sender As
Object, ByVal e As GridPrepareViewStyleEventArgs)
    If e.ColumnIndex = 2 And e.RowIndex = 2 Then
        e.Style.BackColor = Color.Red
    End If
End Sub
```

## 5.2.4 How to Change the Backcolor of a Single Row

### Introduction

In a GridDataBoundGrid, you cannot set row specific properties like **backcolor** using the Model.RowHeaders member. The reason is that in a GridDataBoundGrid, the only data storage is the bound datasource. There is no row specific storage allocated.

This code does not work.

[C#]

```
// Set backcolor of Row 3.
this.gridDataBoundGrid.Model.RowHeaders[8].BackColor=Color.Red;
```

[VB .NET]

```
' Set backcolor of Row 3.
Me.gridDataBoundGrid.Model.RowHeaders(8).BackColor=Color.Red;
```

So, in order to set row specific properties in a GridDataBoundGrid, you must catch the **PrepareViewStyleInfo** event (or **Model.QueryCellInfo** event). In your handler, check **e.RowIndex** and if it points to the row you want to color, set **e.Style** to the value you want (The **e.ColumnIndex > 0** check in the code avoids coloring the header cell in the row).

### Example

#### [C#]

```
private void gridDataBoundGrid1_PreparesViewStyleInfo(object sender,
GridPrepareViewStyleEventArgs e)
{
    if(e.ColumnIndex == 0 && e.RowIndex == 8)
    {
        e.Style.BackColor = Color.Red;
    }
}
```

#### [VB .NET]

```
Private Sub gridDataBoundGrid1_PreparesViewStyleInfo(ByVal sender As
Object, ByVal e As GridPrepareViewStyleEventArgs)
    If e.ColumnIndex = 0 And e.RowIndex = 8 Then
        e.Style.BackColor = Color.Red
    End If
End Sub
```

## 5.2.5 How to Change the Column Header Text

### Introduction

The GridDataBoundGrid by default uses the property name of the content of the columns as the header text. For a **DataTable**, this is the **DataTable.Column.ColumnName** property.

To specify a header other than this default one, set the **GridBoundColumn.HeaderText** property for the column. If you have explicitly added the **GridBoundColumns** to the **GridDataBoundGrid.GridBoundColumns** collection, then you must use the **GridBoundColumns** in that collection. If you did not add the **GridBoundColumns**, then you can use the **GridBoundColumns** that are autogenerated when you set the datasource to the grid, **GridDataBoundGrid.Binder.InternalColumns**.

### Example

**[C#]**

```
// Change the Header Text of 'Attribute' Column.  
this.gridDataBoundGrid1.Binder.InternalColumns["Attribute"].HeaderText  
= "NewHeaderText";
```

**[VB .NET]**

```
' Change the Header Text of 'Attribute' Column.  
Me.GridDataBoundGrid1.Binder.InternalColumns("Attribute").HeaderText =  
"NewHeaderText"
```

## 5.2.6 How to Change the Look of a Column's Border

### Introduction

You can use the **Borders** property of the **GridStyleInfo** to change the style and the appearance of the grid cell's border. Each border side of the cell can be configured individually with a **GridBorder** value. There is a **BorderMargins** property to control the margins on all four sides. In a GridDataBoundGrid, you can set the style properties column by column using **GridDataBoundGrid.GridBoundColumns** or **GridDataBoundGrid.Binder.InternalColumns** depending upon whether you had explicitly added the **GridBoundColumns** or not.

### Example

**[C#]**

```
// Borders on all four sides of the cell.  
GridStyleInfo style =  
this.gridDataBoundGrid1.GridBoundColumns[1].StyleInfo;  
style.Borders.All = new GridBorder(GridBorderStyle.Solid, Color.Red,  
GridBorderWeight.Thin);  
  
style = this.gridDataBoundGrid1.GridBoundColumns[2].StyleInfo;  
style.Borders.Right = new GridBorder(GridBorderStyle.Dotted,  
Color.LightBlue, GridBorderWeight.ExtraThick);  
style.Borders.Bottom = new GridBorder(GridBorderStyle.Solid,  
Color.Pink, GridBorderWeight.Medium);  
style.Borders.Left = new GridBorder(GridBorderStyle.DashDot,  
Color.LightGreen, GridBorderWeight.ExtraThick);  
style.Borders.Top = new GridBorder(GridBorderStyle.DashDotDot,  
Color.Purple, GridBorderWeight.ExtraExtraThick);
```

```
// Border Margins  
this.gridDataBoundGrid1.GridBoundColumns[4].StyleInfo.BorderMargins.Right = 20;  
this.gridDataBoundGrid1.GridBoundColumns[4].StyleInfo.BorderMargins.Left = 22;  
this.gridDataBoundGrid1.GridBoundColumns[4].StyleInfo.BorderMargins.Top = 24;  
this.gridDataBoundGrid1.GridBoundColumns[4].StyleInfo.BorderMargins.Bottom = 26;
```

**[VB.NET]**

```
' Borders on all four sides of the cell.  
Dim style As GridStyleInfo =  
Me.gridDataBoundGrid1.GridBoundColumns(1).StyleInfo  
style.Borders.All = New GridBorder(GridBorderStyle.Solid, Color.Red,  
GridBorderWeight.Thin)  
  
style = Me.gridDataBoundGrid1.GridBoundColumns(2).StyleInfo  
style.Borders.Right = New GridBorder(GridBorderStyle.Dotted,  
Color.LightBlue, GridBorderWeight.ExtraThick)  
style.Borders.Bottom = New GridBorder(GridBorderStyle.Solid,  
Color.Pink, GridBorderWeight.Medium)  
style.Borders.Left = New GridBorder(GridBorderStyle.DashDot,  
Color.LightGreen, GridBorderWeight.ExtraThick)  
style.Borders.Top = New GridBorder(GridBorderStyle.DashDotDot,  
Color.Purple, GridBorderWeight.ExtraExtraThick)  
  
' Bordermargins  
Me.gridDataBoundGrid1.GridBoundColumns(4).StyleInfo.BorderMargins.Right = 20  
Me.gridDataBoundGrid1.GridBoundColumns(4).StyleInfo.BorderMargins.Left = 22  
Me.gridDataBoundGrid1.GridBoundColumns(4).StyleInfo.BorderMargins.Top = 24  
Me.gridDataBoundGrid1.GridBoundColumns(4).StyleInfo.BorderMargins.Bottom = 26
```

## 5.2.7 How to change the row header to display line numbers instead of the black triangle in GridDataBoundGrid

You can achieve this by setting the row header base style to *Header*, and handling the **PrepareViewStyleInfo** event handler to set the line numbers. Refer the below code snippet which illustrates this.

**[C#]**

```
// In the Form Load
private void Form1_Load(object sender, System.EventArgs e)
{
    this.gridDataBoundGrid1.DataSource = GetTable();
    this.gridDataBoundGrid1.BaseStylesMap["Row Header"].StyleInfo.CellType =
= "Header";
}

// GridPrepareViewStyleInfo
private void grid_PreparesViewStyleInfo(object sender,
GridPrepareViewStyleInfoEventArgs e)
{
    if (e.ColIndex == 0 && e.RowIndex > 0)
    {
        e.Style.Text = e.RowIndex.ToString();
        e.Style.Font.Bold = false;
    }
}
```

**[VB .NET]**

```
' In the Form Load
Private Sub Form1_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles MyBase.Load
Me.gridDataBoundGrid1.DataSource = GetTable()
Me.gridDataBoundGrid1.BaseStylesMap("Row Header").StyleInfo.CellType =
= "Header"
End Sub 'Form1_Load

' GridPrepareViewStyleInfo
Private Sub gridDataBoundGrid1_PreparesViewStyleInfo(ByVal sender As
Object, ByVal e As
Syncfusion.Windows.Forms.Grid.GridPrepareViewStyleInfoEventArgs)
Handles gridDataBoundGrid1.PreparesViewStyleInfo
If e.ColIndex = 0 AndAlso e.RowIndex > 0 Then
    e.Style.Text = e.RowIndex.ToString()
    e.Style.Font.Bold = False
End If
End Sub
```

## 5.2.8 How to Copy a Range of Cells to the Clipboard

### Introduction

You can use the **CopyTextToClipboard** method to copy the text from a selected range of cells to the clipboard.

### Example

#### [C#]

```
// To copy selected range of cells to clipboard in a GridControl.  
bool val =  
this.gridControl1.CutPaste.CopyTextToClipboard(this.gridControl1.Selections.Ranges);  
Console.WriteLine(" Selected Range of cells(GridControl)are in  
Clipboard. This is"+val);  
MessageBox.Show("Data copied");  
  
// To copy selected range of cells to clipboard in a GridDataBoundGrid.  
bool val =  
this.gridDataBoundGrid1.Model.CutPaste.CopyTextToClipboard(this.gridDataBoundGrid1.Selections.Ranges);  
Console.WriteLine(" Selected Range of cells(GridDataBoundGrid)are in  
Clipboard. This is"+val);  
MessageBox.Show("Data copied");
```

#### [VB .NET]

```
' To copy a selected range of cells to the clipboard in a GridControl.  
Dim val As Boolean =  
Me.gridControl1.CutPaste.CopyTextToClipboard(Me.gridControl1.Selections.Ranges)  
Console.WriteLine(" Selected Range of cells(GridControl)are in  
Clipboard. This is", val)  
MessageBox.Show("Data copied")  
  
' To copy a selected range of cells to the clipboard in a  
GridDataBoundGrid.  
Dim val As Boolean =  
Me.gridDataBoundGrid1.Model.CutPaste.CopyTextToClipboard(Me.gridDataBoundGrid1.Selections.Ranges)  
Console.WriteLine("Selected Range of cells(GridDataBoundGrid)are in  
Clipboard. This is", val)  
MessageBox.Show("Data copied")
```

## 5.2.9 How to draw a check box cell in a **GridDataBoundGrid** header

The following scenario explains how to add a check box cell in the grid header, which when selected or deselected orders its particular column of check boxes to respond. In the **SetupCheckBox()** method, the concerned location (header position) of the check box has been provided. In the form, load the following events: **Layout**, **ResizingRows**, **ResizingColumns**. These events should be handled for the check box to update correctly. In the **checkBox.CheckedChanged** event, the required customized code for header check box operation is provided.

[C#]

```
private void SetupCheckBox()

{
    Rectangle cellRect =
this.gridDataBoundGrid1.ViewLayout.RangeInfoToRectangle(GridRange
Info.Cell(0, 1), GridCellSizeKind.ActualSize);

checkBox.Location = new Point(cellRect.X + cellRect.Width / 2 -
13 / 2, cellRect.Y + cellRect.Height / 2 - 13 / 2); ///
cellRect.Location;

checkBox.Size = new Size(13, 13);

}

void gridDataBoundGrid1_Layout(object sender, LayoutEventArgs e)

{
    SetupCheckBox();
}

void gridDataBoundGrid1_ResizingColumns(object sender,
GridResizingColumnsEventArgs e)

{
    this.gridDataBoundGrid1.BeginUpdate();

    SetupCheckBox();
}
```

```
this.gridDataBoundGrid1.EndUpdate();

this.gridDataBoundGrid1.Model.Refresh();

}
```

**Sample C#:** [GDBG header check box c#.zip](#)

**[VB]**

```
Private Sub SetupCheckBox()

Dim cellRect As Rectangle =
Me.gridDataBoundGrid1.ViewLayout.RangeInfoToRectangle(GridRangeIn
fo.Cell(0, 1), GridCellSizeKind.ActualSize)

checkBox.Location = New Point(cellRect.X + cellRect.Width / 2 -
13 \ 2, cellRect.Y + cellRect.Height / 2 - 13 \ 2) '
cellRect.Location;

checkBox.Size = New Size(13, 13)

End Sub

Private Sub gridDataBoundGrid1_Layout(sender As Object, e As
LayoutEventArgs)

    SetupCheckBox()

End Sub

Private Sub gridDataBoundGrid1_ResizingColumns(sender As Object,
e As GridResizingColumnsEventArgs)

    Me.gridDataBoundGrid1.BeginUpdate()

    SetupCheckBox()

    Me.gridDataBoundGrid1.EndUpdate()

    Me.gridDataBoundGrid1.Model.Refresh()

End Sub
```

**Sample VB:** [GDBG header check box VB.zip](#)

## 5.2.10 How to Insert an Unbound CheckBox Column

### Introduction

There are three steps needed to have an unbound checkbox column in the GridDataBoundGrid.

1. Add the unbound column to the GridDataBoundGrid.
2. A bounded column has its own datastore to store the values that are entered in it. But, in an unbound column there is no datastore present in it, so it has to be handled manually by adding any collection like a hashtable. To make an unbound column work together with sort, assign a key as the corresponding primary key value in the row in the Key / Value pair of the external collection.
3. Then **Model.QueryCellInfo** event is used to display the values in the checkbox. The **Model.SaveCellInfo** event is used to save the changes that are made by the user in the checkbox.

### Example

[C#]

```
private void Form1_Load(object sender, System.EventArgs e)
{
    // Create a DataTable.
    DataTable dt = new DataTable("Table");

    dt.Columns.Add("Product",typeof(System.String));

    dt.Columns.Add("Product_ID",typeof(System.Int32));

        // Add a primary key.
    dt.PrimaryKey = new DataColumn[]
{dt.Columns["Product_ID"]};

    dt.Columns.Add("QtyPerUnit",typeof(System.String));

    dt.Columns.Add("UnitPrice",typeof(System.Double));

        dt.Rows.Add(new Object[] { "Coffee",1,"1
kg",200.00 });
        dt.Rows.Add(new Object[] { "Tea",2,"1
kg",120.00 });
        dt.Rows.Add(new Object[] { "Milk Powder",3,"1
kg",260.00 });

}
```

```
        dt.Rows.Add(new Object[] { "Biscuits", 4, "1
Packet", 20.00 });
        dt.Rows.Add(new Object[] { "Pepsi", 5, "1
Lit", 60.00 });
        dt.Rows.Add(new Object[] { "Coke", 6, "1
Lit", 55.00 });

        // Set DataSource property.
this.gridDataBoundGrid1.DataSource = dt;

        // Create an unbound column for checkbox.
GridBoundColumnsCollection column5 =
(GridBoundColumnsCollection)
this.gridDataBoundGrid1.Binder.InternalColumns.Clone();
GridBoundColumn Ucolumn5 = new
GridBoundColumn();

        // Set Header Text and assign Mapping Name.
Ucolumn5.HeaderText = "CheckBox";
Ucolumn5.MappingName = "CheckBox";

        // Bind the unbound column to the grid.
column5.Add(Ucolumn5);
this.gridDataBoundGrid1.Binder.GridBoundColumns
= column5;

        // Handle Query Cell Info to display the values
in CheckBox column.
this.gridDataBoundGrid1.Model.QueryCellInfo +=
new GridQueryCellInfoEventHandler(Model_QueryCellInfo);

        // Handle SaveCellInfo to save changes that are
made by the user in the CheckBox.
this.gridDataBoundGrid1.Model.SaveCellInfo +=
new GridSaveCellInfoEventHandler(Model_SaveCellInfo);
}

private void Model_QueryCellInfo(object sender,
GridQueryCellInfoEventArgs e)
{
    GridModel model =
this.gridDataBoundGrid1.Model;
    int ColIndex =
this.gridDataBoundGrid1.Model.NameToColIndex("CheckBox");
    if(e.RowIndex > 0 && e.ColIndex == ColIndex)
    {
        // Set up a CheckBox control.
        e.Style.CellType = "CheckBox";
    }
}
```

```
        e.Style.HorizontalAlignment =
GridHorizontalAlignment.Center;
        e.Style.VerticalAlignment =
GridVerticalAlignment.Middle;
        e.Style.CellValueType = typeof(bool);

        // Determine the Checked and Unchecked
values of CheckBox.
        e.Style.CheckBoxOptions.CheckedValue =
"True";
        e.Style.CheckBoxOptions.UncheckedValue =
"False";
        e.Style.Enabled = true;
        int keyColIndex =
model.NameToColIndex("Product_ID");
        string key = model[e.RowIndex,
keyColIndex].Text;
        if (key != null)
{
    object value =
CheckBoxValues[key];
        // Display the value in
checkbox.
    if (value != null)
        e.StyleCellValue =
value;
}
}

private void Model_SaveCellInfo(object sender,
GridSaveCellInfoEventArgs e)
{
    GridModel model =
this.gridDataBoundGrid1.Model;
    int ColIndex =
this.gridDataBoundGrid1.Model.NameToColIndex("CheckBox");
    if(e.RowIndex > 0 && e.ColIndex == ColIndex)
{
    int keyColIndex =
model.NameToColIndex("Product_ID");
    string key = model[e.RowIndex,
keyColIndex].Text;

    // Save the checkbox value that is
modified by the user.
    if (key != null)
```

```
        CheckBoxValues[key] =
e.Style.CellValue;
    }
}
}
}
```

**[VB.NET]**

```
Private Sub Form1 Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    ' Create a DataTable.
    Dim dt As DataTable = New DataTable("Table")
    dt.Columns.Add("Product", GetType(System.String))
    dt.Columns.Add("Product_ID", GetType(System.Int32))

    ' Add a primary key.
    dt.PrimaryKey = New DataColumn() {dt.Columns("Product_ID")}
    dt.Columns.Add("QtyPerUnit", GetType(System.String))
    dt.Columns.Add("UnitPrice", GetType(System.Double))

    dt.Rows.Add(New Object() { "Coffee", 1, "1 kg", 200.00})
    dt.Rows.Add(New Object() { "Tea", 2, "1 kg", 120.00})
    dt.Rows.Add(New Object() { "Milk Powder", 3, "1 kg", 260.00})
    dt.Rows.Add(New Object() { "Biscuits", 4, "1 Packet", 20.00})
    dt.Rows.Add(New Object() { "Pepsi", 5, "1 Lit", 60.00})
    dt.Rows.Add(New Object() { "Coke", 6, "1 Lit", 55.00})

    ' Set DataSource property.
    Me.gridDataBoundGrid1.DataSource = dt

    ' Create an unbound column for checkbox.
    Dim column5 As GridBoundColumnsCollection =
CType(Me.gridDataBoundGrid1.Binder.InternalColumns.Clone(), GridBoundColumnsCollection)
    Dim Ucolumn5 As GridBoundColumn = New GridBoundColumn()

    ' Set Header Text and assign Mapping Name.
    Ucolumn5.HeaderText = "CheckBox"
    Ucolumn5.MappingName = "CheckBox"

    ' Bind the unbound column to the grid.
    column5.Add(Ucolumn5)
    Me.gridDataBoundGrid1.Binder.GridBoundColumns = column5
```

```

' Handle Query Cell Info to display the values in CheckBox column.
AddHandler gridDataBoundGrid1.Model.QueryCellInfo, AddressOf
Model_QueryCellInfo

' Handle SaveCellInfo to save changes that are made by the user in
the CheckBox.
AddHandler gridDataBoundGrid1.Model.SaveCellInfo, AddressOf
Model_SaveCellInfo
End Sub

Private Sub Model_QueryCellInfo(ByVal sender As Object, ByVal e As
GridQueryCellInfoEventArgs)
    Dim model As GridModel = Me.gridDataBoundGrid1.Model
    Dim ColIndex As Integer =
Me.gridDataBoundGrid1.Model.NameToColIndex("CheckBox")
    If e.RowIndex > 0 AndAlso e.ColumnIndex = ColIndex Then

        ' Set up a CheckBox control.
        e.Style.CellType = "CheckBox"
        e.Style.HorizontalAlignment = GridHorizontalAlignment.Center
        e.Style.VerticalAlignment = GridVerticalAlignment.Middle
        e.Style.CellValueType = GetType(Boolean)

        ' Determine the Checked and Unchecked values of CheckBox.
        e.Style.CheckBoxOptions.CheckedValue = "True"
        e.Style.CheckBoxOptions.UncheckedValue = "False"
        e.Style.Enabled = True
        Dim keyColIndex As Integer = model.NameToColIndex("Product_ID")
        Dim key As String = model(e.RowIndex, keyColIndex).Text
        If Not key Is Nothing Then
            Dim value As Object = CheckBoxValues(key)

            ' Display the value in checkbox.
            If Not value Is Nothing Then
                e.StyleCellValue = value
            End If
        End If
    End If
End Sub

Private Sub Model_SaveCellInfo(ByVal sender As Object, ByVal e As
GridSaveCellInfoEventArgs)
    Dim model As GridModel = Me.gridDataBoundGrid1.Model
    Dim ColIndex As Integer =
Me.gridDataBoundGrid1.Model.NameToColIndex("CheckBox")
    If e.RowIndex > 0 AndAlso e.ColumnIndex = ColIndex Then
        Dim keyColIndex As Integer = model.NameToColIndex("Product_ID")

```

```
Dim key As String = model(e.RowIndex, keyColIndex).Text

    ' Save the checkbox value that is modified by the user.
    If Not key Is Nothing Then
        CheckBoxValues(key) = e.Style.CellValue
    End If
End If
End Sub
End Class
End Namespace
```

## 5.2.11 How to Include an Icon in the Column Header

### Introduction

You can place images in cells by specifying style.ImageIndex and style.ImageList values for the cell, provided the style.CellType is either "Static" or "text box". To make your header cell hold an icon, you can make it "Static" and set these properties appropriately. For a **GridDataBoundGrid**, setting such properties on a single cell will require you to use the **PrepareViewStyleInfo** event.

### Example

[C#]

```
private void gridDataBoundGrid1_PreparesViewStyleInfo(object sender,
GridPrepareViewStyleInfoEventArgs e)
{
    if(e.RowIndex == 0 )
    {
        // Set Cell as Static.
        e.Style.CellType = "Static";
        e.Style.CellAppearance = GridCellAppearance.Raised;

        // Assign an icon image to the cell.
        if(e.ColIndex == 3)
        {
            e.Style.ImageList = imageList;
            e.Style.ImageIndex = 1;
        }
    }
}
```

[VB.NET]

```
Private Sub gridDataBoundGrid1_PreparesViewStyleInfo(ByVal sender As Object, ByVal e As GridPrepareViewStyleEventArgs)

    ' Set Cell as Static.
    If e.RowIndex = 0 Then
        e.Style.CellType = "Static"
        e.Style.CellAppearance = GridCellAppearance.Raised

    ' Assign an icon image to the cell.
    If e.ColumnIndex = 3 Then
        e.Style.ImageList = Me.imageList
        e.Style.ImageIndex = 1
    End If
End If

' GridDataBoundGrid1_PreparesViewStyleInfo
End Sub
```

## 5.2.12 How to Make the Particular Cells ReadOnly

### Introduction

In general, cell specific style settings (other than CellValue or text) in a GridDataBoundGrid need to be done through an event like **PrepareViewStyleInfo**. Functional properties like **Read-only** that are used to determine the cell's functionality need to be set in **Model.QueryCellInfo**. But, visual properties like the **font** and **backcolor** can be set in either the PrepareViewStyleInfo or the Model.QueryCellInfo.

### Example

[C#]

```
this.gridDataBoundGrid1.Model.QueryCellInfo += new
GridQueryCellInfoEventHandler(Model_QueryCellInfo);

private void Model_QueryCellInfo(object sender,
GridQueryCellInfoEventArgs e)
{
    // Set ReadOnly property of the Cell(2,2) to true.
    if(e.ColumnIndex == 2 && e.RowIndex == 2)
    {
```

```
        e.Style.ReadOnly = true;
    }
}
```

**[VB.NET]**

```
AddHandler Me.gridDataBoundGrid1.Model.QueryCellInfo, AddressOf
Model_QueryCellInfo

    Private Sub Model_QueryCellInfo(sender As Object, e As
GridQueryCellInfoEventArgs)

        ' Set ReadOnly property of the Cell(2,2) to true.
        If e.ColumnIndex = 2 And e.RowIndex = 2 Then
            e.Style.ReadOnly = True
        End If
    End Sub
```

## 5.2.13 How to Make the GridFilterBar Use an Autocomplete Combo Box

### Introduction

To get autocomplete using the standard **GridFilterBar**, try the following code.

### Example

**[C#]**

```
// Bind FilterBar to the grid.
theFilterBar.WireGrid(this.gridDataBoundGrid1);
```

**[VB.NET]**

```
' Bind FilterBar to the grid.
theFilterBar.WireGrid(Me.gridDataBoundGrid1)
```

**[C#]**

```
// Set up a GridStyleInfo object.
GridStyleInfo style = new GridStyleInfo();
```

```
style.ModifyStyle(this.gridDataBoundGrid1.BaseStylesMap["Header"].StyleInfo, StyleModifyType.Copy);

    // Set cell type of the style object to ComboBox.
    style.CellType = "ComboBox";
    style.ExclusiveChoiceList = true;

    // Apply style settings.
    style.BaseStyle = "Standard";
    style.Font.Bold = false;
    style.BackColor = this.gridDataBoundGrid1.TableStyle.BackColor;
    style.Borders.Bottom = new GridBorder(GridBorderStyle.Dashed);

    // Set DropDownStyle to Autocomplete.
    style.DropDownStyle = GridDropDownStyle.AutoComplete;

    // Apply the StyleInfo object to FilterBar.
    theFilterBar.WireGrid(this.gridDataBoundGrid1, style);
```

And replace it with this code.

**[VB.NET]**

```
' Set up a GridStyleInfo object.
Dim style As New GridStyleInfo
style.ModifyStyle(Me.gridDataBoundGrid1.BaseStylesMap("Header").StyleInfo, StyleModifyType.Copy)

' Set cell type of the style object to ComboBox.
style.CellType = "ComboBox"
style.ExclusiveChoiceList = True

' Apply style settings.
style.BaseStyle = "Standard"
style.Font.Bold = False
style.BackColor = Me.gridDataBoundGrid1.TableStyle.BackColor
style.Borders.Bottom = New GridBorder(GridBorderStyle.Dashed)

' Set DropDownStyle to Autocomplete.
style.DropDownStyle = GridDropDownStyle.AutoComplete

' Apply the StyleInfo object to FilterBar.
theFilterBar.WireGrid(Me.gridDataBoundGrid1, style)
```

This adds the **DropDownStyle** = Autocomplete, setting it to the default combobox.

## 5.2.14 How to merge two columns in a GridDataBoundGrid

You can make use of the **CoveredRanges** property to merge two or more columns in the GridDataBoundGrid.

[C#]

```
this.gridDataBoundGrid1.Model.CoveredRanges.Add(GridRangeInfo.Cells(0, 3, 0, 4));  
  
Private void gridDataBoundGrid1_CellClick(object sender, GridCellEventArgs e)  
{  
    if (e.RowIndex == 0 && e.ColumnIndex > 0)  
    {  
        GridRangeInfo range;  
        bool check =  
this.gridDataBoundGrid1.Model.CoveredRanges.Find(e.RowIndex, e.ColumnIndex, out range);  
        if (check)  
this.gridDataBoundGrid1.Model.Selections.Add(GridRangeInfo.Cols(range.Left, range.Right));  
    }  
}  
  
// On Mouse down, select both the column which is merged.  
Private void gridDataBoundGrid1_MouseDown(object sender, MouseEventArgs e)  
{  
    int row, col;  
    this.gridDataBoundGrid1.PointToRowCol(new Point(e.X, e.Y), out row, out col);  
    if (row == 0 && col > 0)  
    {  
        GridRangeInfo range;  
        bool check = this.gridDataBoundGrid1.Model.CoveredRanges.Find(row, col, out range);  
        if (check)  
this.gridDataBoundGrid1.Model.Selections.Add(GridRangeInfo.Cols(range.Left, range.Right));  
    }  
}
```

**[VB.NET]**

```
Me.gridDataBoundGrid1.Model.CoveredRanges.Add(GridRangeInfo.Cells(0,3,0,4));  
  
Private Sub GridDataBoundGrid1_CellClick(ByVal sender As System.Object, ByVal e As Syncfusion.Windows.Forms.Grid.GridCellEventArgs) Handles GridDataBoundGrid1.CellClick  
If e.RowIndex = 0 AndAlso e.ColumnIndex > 0 Then  
Dim range As GridRangeInfo  
Dim check As Boolean =  
Me.gridDataBoundGrid1.Model.CoveredRanges.Find(e.RowIndex, e.ColumnIndex, range)  
If check Then  
Me.gridDataBoundGrid1.Model.Selections.Add(GridRangeInfo.Cols(range.Left, range.Right))  
End If  
End If  
' On Mouse down, select both the column which is merged.  
Private Sub GridDataBoundGrid1_MouseDown(ByVal sender As Object, ByVal e As System.Windows.Forms.MouseEventArgs) Handles GridDataBoundGrid1.MouseDown  
Dim row, col As Integer  
Me.gridDataBoundGrid1.PointToRowCol(New Point(e.X, e.Y), row, col)  
If row = 0 AndAlso col > 0 Then  
Dim range As GridRangeInfo  
Dim check As Boolean =  
Me.gridDataBoundGrid1.Model.CoveredRanges.Find(row, col, range)  
If check Then  
Me.gridDataBoundGrid1.Model.Selections.Add(GridRangeInfo.Cols(range.Left, range.Right))  
End If  
End If
```

## 5.2.15 How to Paste Clipboard Contents Bigger than GridDataBoundGrid Row and Column Count

### Introduction

The GridDataBoundGrid does not increment the row and column count as a **GridControl**. The reason for this is that the GridDataBoundGrid does not have its own datastore, it has to store these in the relevant datatable. To add rows and columns in the datatable, the Model.ClipboardPaste handler must be used.

In this handler, the clipboard contents that are stored in text format of the **DataObject** will be a single string. This has to be split with '\n' and '\t' to refer the rows and columns fashion and this count is compared with the existing row and column

count. The necessary extra rows are added then and makes pasting possible.

### Example

[C#]

```
this.gridDataBoundGrid1.UseListChangedEvent = false;

DataObject data = (DataObject) Clipboard.GetDataObject();

// Gets the size of the data object in rows.
string s = (string)data.GetData(DataFormats.Text);
string[] rows = s.Split(new char[]{'\n'});
int numRows = rows.GetLength(0);
if(numRows > 0 && rows[numRows - 1].Length == 0)

// Removes extra empty row if present.
numRows--;

// Gets the size of the data object in Columns.
string[] cols = rows[0].Split(new char[]{'\t'});
int numCols = cols.GetLength(0);
if(numCols > 0 && cols[numCols - 1].Length == 0)

// Removes extra empty column if present.
numCols--;

int extraRowIfStartAtAddNewRow =
this.gridDataBoundGrid1.Binder.IsAddNew ? 1 : 0;
while(currentCell.RowIndex + numRows + extraRowIfStartAtAddNewRow >
grid.Model.RowCount)
{
    DataRow dr = dt.NewRow();
    dt.Rows.Add(dr);
    row--;
}
while(currentCell.COLIndex + numCols > grid.Model.ColCount+1)
{
    dt.Columns.Add();
    col--;
}
this.gridDataBoundGrid1.Binder.ResumeBinding();
```

```
this.gridDataBoundGrid1.EndUpdate();
```

**[VB.NET]**

```
Me.gridDataBoundGrid1.UseListChangedEvent = False

data As DataObject = CType(Clipboard.GetDataObject(), DataObject)

' Gets the size of the data object in rows.
s As String = CStr(data.GetData(DataFormats.Text))
rows As String() = s.Split(New Char(){ControlChars.Lf})
numRows As Integer = rows.GetLength(0)
If numRows > 0 AndAlso rows(numRows - 1).Length = 0 Then

    ' Removes extra empty row if present.
    numRows -= 1
End If

' Gets the size of the data object in Columns.
Dim cols As String() = rows(0).Split(New Char(){ControlChars.Tab})
Dim numCols As Integer = cols.GetLength(0)
If numCols > 0 AndAlso cols(numCols - 1).Length = 0 Then

    ' Removes extra empty column if present.
    numCols -= 1
End If

Dim extraRowIfStartAtAddNewRow As Integer =
If(Me.gridDataBoundGrid1.Binder.IsAddNew, 1, 0)
Do While currentCell.RowIndex + numRows + extraRowIfStartAtAddNewRow >
grid.Model.RowCount
Dim dr As DataRow = dt.NewRow()
dt.Rows.Add(dr)
row -= 1
Loop
Do While currentCell.ColumnIndex + numCols > grid.Model.ColumnCount + 1
dt.Columns.Add()
col -= 1
Loop
Me.gridDataBoundGrid1.Binder.ResumeBinding()
Me.gridDataBoundGrid1.EndUpdate()
```

## 5.2.16 How to Prevent Showing the '+' Sign Next to the Parent Rows with no Children

### Introduction

To hide the expansion indicator that reflects that no children are present at the parent row in a hierarchical grid, you will need to handle the **CellDrawn** and **CellButtonClick** Event.

Follow the steps given below.

1. In the CellDrawn Event Handler ,to assign the number of children in the parent row, to count and to paint the current cell with **BackgroundColor** if count equals to zero.

### Example

[C#]

```
private void gridDataBoundGrid1_CellDrawn(object sender,
Syncfusion.Windows.Forms.Grid.GridDrawCellEventArgs e)

{

    if( flag )

    {

        if(e.ColumnIndex == 1)

        {

            // Retrieve the record state of a row.
            GridBoundRecordState state =
this.gridDataBoundGrid1.Binder.GetRecordStateAtRowIndex(e.RowIndex);

            count = 0;

            // Check its Hierarchy Level Index.
            if( state.LevelIndex == 0 )

            {

                this.gridDataBoundGrid1.BeginUpdate();

this.gridDataBoundGrid1.Model.SuspendChangeEvents();

                // To expand the rowIndex for finding the
            }
        }
    }
}
```

```
childCount.

this.gridDataBoundGrid1.ExpandAtRowIndex(e.RowIndex);

        // Get the number of child.
        count = state.ChildCount;

        // To collapse the expanded row.

this.gridDataBoundGrid1.CollapseAtRowIndex(e.RowIndex);

this.gridDataBoundGrid1.Model.ResumeChangeEvent();

        this.gridDataBoundGrid1.EndUpdate();

    }

}

// If no. of child is zero, fill with backcolor to
// avoid plus sign.
if( count == 0 && e.ColumnIndex == 1 )

{

    Brush brush = new SolidBrush(e.Style.BackColor);

    try

    {

        // Paint the current to BackColor.
        e.Graphics.FillRectangle(brush,e.Bounds);

        e.Cancel = true;

    }

    finally

    {

        brush.Dispose();

    }

}
```

```
    }  
  
}  
  
}
```

**[VB.NET]**

```
Private Sub gridDataBoundGrid1_CellDrawn(ByVal sender As Object, ByVal e As Syncfusion.Windows.Forms.Grid.GridDrawCellEventArgs)  
  
    If flag Then  
  
        If e.ColumnIndex = 0 Then  
  
            ' Retrieve the record state of a row.  
            Dim state As GridBoundRecordState =  
Me.gridDataBoundGrid1.Binder.GetRecordStateAtRowIndex(e.RowIndex)  
  
            count = 0  
  
            ' Check its Hierarchy Level Index.  
            If state.LevelIndex = 0 Then  
  
                Me.gridDataBoundGrid1.BeginUpdate()  
  
                Me.gridDataBoundGrid1.Model.SuspendChangeEvents()  
  
                ' To expand the rowIndex for finding the childCount.  
                Me.gridDataBoundGrid1.ExpandAtRowIndex(e.RowIndex)  
  
                ' Get the no. of childs.  
                count = state.ChildCount  
  
                ' To collapse the expanded row.  
                Me.gridDataBoundGrid1.CollapseAtRowIndex(e.RowIndex)  
  
                Me.gridDataBoundGrid1.Model.ResumeChangeEvents()  
  
                Me.gridDataBoundGrid1.EndUpdate()  
  
            End If  
  
        End If  
  
    End If
```

```
' If no. of childs is zero, fill with backcolor to avoid plus sign.  
If count = 0 And e.ColumnIndex = 1 Then  
  
    Dim brush As Brush = New SolidBrush(e.Style.BackColor)  
  
    Try  
  
        ' Paint the current to BackColor.  
        e.Graphics.FillRectangle(brush, e.Bounds)  
  
        e.Cancel = True  
  
    Finally  
  
        brush.Dispose()  
  
    End Try  
  
End If  
  
End If  
  
End Sub
```

2. In the CellButtonClick EventHandler,to set the flag to control the Current Cell Drawing.

### **Example**

#### **[C#]**

```
// Set flag to false to control the Current Cell Drawing.  
private void gridDataBoundGrid1_CellButtonClicked(object sender,  
Syncfusion.Windows.Forms.Grid.GridCellClickedEventArgs e)  
{  
    flag = false;  
}
```

#### **[VB.NET]**

```
' Set flag to false to control the Current Cell Drawing.  
Private Sub gridDataBoundGrid1_CellButtonClicked(ByVal sender As  
Object, ByVal e As  
Syncfusion.Windows.Forms.Grid.GridCellClickedEventArgs)
```

```
    flag = False  
  
End Sub
```

## 5.2.17 How to QueryCellFormattedText and SaveCellFormattedText

### Introduction

The purpose of the **QueryCellFormattedText** is to take the data which, is present in the Total Marks and convert it to Percentage for display. The **SaveCellFormattedText** takes the units that are entered by the user in Percentage, and converts them to Total Marks so that they can be saved as Total Marks.

### Example

```
[C#]  
  
private void Model_QueryCellFormattedText(object sender,  
Syncfusion.Windows.Forms.Grid.GridCellTextEventArgs e)  
{  
  
    if(e.Style.CellValueType==typeof(double) && e.Style.Text.Length  
> 0)  
    {  
        // Convert the cell value to percentage for display.  
        double dVal = (double)e.StyleCellValue/500 *100;  
        e.Text = dVal.ToString("##.##");  
        e.Handled = true;  
    }  
}  
  
private void Model_SaveCellFormattedText(object sender,  
Syncfusion.Windows.Forms.Grid.GridCellTextEventArgs e)  
{  
    if(e.Style.CellValueType==typeof(double) && e.Style.Text.Length  
> 0)  
    {  
        // Convert the Cell Value again from percentage to its  
        // original format.  
        double dVal = double.Parse(e.Text)/100 *500;
```

```
// Save the converted cell value.  
e.Style.CellValue = dVal;  
e.Handled = true;  
}  
}
```

**[VB.NET]**

```
Private Sub Model_QueryCellFormattedText(ByVal sender As Object, ByVal e As Syncfusion.Windows.Forms.Grid.GridCellTextEventArgs)  
If e.Style.CellValueType Is GetType(Double) AndAlso  
e.Style.Text.Length > 0 Then  
  
    ' Convert the cell value to percentage for display.  
    Dim dVal As Double = CDbl(e.StyleCellValue)/500 *100  
    e.Text = dVal.ToString("##.##")  
    e.Handled = True  
End If  
End Sub  
  
Private Sub Model_SaveCellFormattedText(ByVal sender As Object, ByVal e As Syncfusion.Windows.Forms.Grid.GridCellTextEventArgs)  
If e.Style.CellValueType Is GetType(Double) AndAlso  
e.Style.Text.Length > 0 Then  
  
    ' Convert the Cell Value again from percentage to its original  
    format.  
    Dim dVal As Double = Double.Parse(e.Text)/100 *500  
  
    ' Save the converted cell value.  
    e.StyleCellValue = dVal  
    e.Handled = True  
End If  
End Sub
```

## 5.2.18 How to Retrieve the DataRow from the GridDataBoundGrid with the RowIndex

### Introduction

The GridDataBoundGrid has to be bound to the datasource using the **CurrencyManager**. Using the CurrencyManager, the record corresponding to the row index can be retrieved.

## Example

### [C#]

```
CurrencyManager  
cm=(CurrencyManager)BindingContext[gridDataBoundGrid1.DataSource,  
gridDataBoundGrid1.DataMember];  
DataRow row;  
DataView dv=(DataView)cm.List;  
  
// The 2 is the rowindex.  
int position = this.gridDataBoundGrid1.Binder.RowIndexToPosition(2);  
row=dv[position].Row;
```

### [VB .NET]

```
Dim cm As CurrencyManager=  
CType(BindingContext(gridDataBoundGrid1.DataSource,  
gridDataBoundGrid1.DataMember), CurrencyManager)  
Dim row As DataRow  
Dim dv As DataView= CType(cm.List, DataView)  
  
' The 2 is the rowindex.  
Dim position As Integer =  
Me.gridDataBoundGrid1.Binder.RowIndexToPosition(2)  
row=dv(position).Row
```

## 5.2.19 How to Set the Width of a Column

### Introduction

In order to change the width of the columns in the **GridDataBoundGrid**, you must do the following.

- Set the property **grid.AllowResizeToFit** to False. This can be done once in the forms constructor or Form.Load event. It will turn off the grids default sizing behavior so that your explicit sizing will work. Otherwise, the default sizing will take precedence. The default sizing uses the width of the header text to size the columns.
- Then, explicitly set the width of the particular columns by using the **Model.ColWidths** collection.

## Example

[C#]

```
// Set size of column 3 to 250.  
this.gridDataBoundGrid1.AllowResizeToFit = false;  
this.gridDataBoundGrid1.Model.ColWidths[3] = 250;
```

[VB .NET]

```
' Set size of column 3 to 250.  
Me.GridDataBoundGrid1.AllowResizeToFit = False  
Me.GridDataBoundGrid1.Model.ColWidths(3) = 250
```

## 5.2.20 How to Set the Height of a Row

### Introduction

To explicitly set the height of a particular row, use the **Model.RowHeights** collection.

## Example

[C#]

```
// Set height of row 3 to 40.  
this.gridDataBoundGrid1.Model.RowHeights[3] = 40;  
  
// Set height of header row 30.  
this.gridDataBoundGrid1.Model.RowHeights[0] = 30;
```

[VB .NET]

```
' Set height of row 3 to 40.  
Me.GridDataBoundGrid1.Model.RowHeights(3) = 40  
  
' Set height of header row 30.  
Me.GridDataBoundGrid1.Model.RowHeights(0) = 30
```

## 5.2.21 How to Stop the Errors Thrown when Pasting Larger Clipboard Contents in a GridDataBoundGrid

### Introduction

A `IndexOutOfRangeException` is thrown when trying to paste clipboard contents that are greater than the number of columns and rows that are available to accommodate the contents. To stop the exception, a condition check at `Model.PasteCellText` handler will do good.

### Example

#### [C#]

```
private void Model_PasteCellText(object sender,
Syncfusion.Windows.Forms.Grid.GridPasteCellTextEventArgs e)
{
    // Check Whether the RowIndex crosses the upper bound.
    if(e.RowIndex >= this.gridDataBoundGrid1.Model.RowCount)
    {
        // If rowindex falls beyond upper bound, Cancel and Abort the
        // Paste process.
        MessageBox.Show("There is no enough rows to paste the rest of
the contents");
        e.Cancel = true;
        e.Abort = true;
    }
}
```

#### [VB .NET]

```
Private Sub Model_PasteCellText(ByVal sender As Object, ByVal e As
Syncfusion.Windows.Forms.Grid.GridPasteCellTextEventArgs)

    ' Check Whether the RowIndex crosses the upper bound.
    If e.RowIndex >= Me.gridDataBoundGrid1.Model.RowCount Then

        ' If rowindex falls beyond upper bound, Cancel and Abort the
        ' Paste process.
        MessageBox.Show("There is no enough rows to paste the rest of
the contents")
        e.Cancel = True
        e.Abort = True
    End If
End Sub
```

## 5.2.22 How to Support Tri-State Sorting in a GridDataBoundGridControl

### Introduction

The standard sorting never removes the sorting from a column; so once a column is sorted it will either be in ascending or descending order. By using the GridDataBoundGridControl's **CellClick** event, it is possible to alter this behavior so that when a column is clicked once, it will be sorted in an ascending order, when it is clicked a second time, the column will be sorted in a descending order and when the column is clicked a third time, it will have its sorting removed.

The tri-state sort behavior is accomplished by storing the sort indicator which, is in the **Tag** property of the column and by handling the CellClick event. When a column is sorted in descending order, then the sorting is removed the next time the column is clicked.

1. There are a couple of problems that you will have to work around. One is that the standard column header cell uses the Tag for its sorthead and it explicitly sets the non-sorted header's Tag to ascending /descending only .It does not support the none option for unsorting the GridDataBoundGridControl. So, you need to hide the HeaderCell and add the New Header.

### Example

[C#]

```
// Set Datasource.
this.gridDataBoundGrid1.DataSource = dt;
this.gridDataBoundGrid1.Model.BaseStylesMap["Column
Header"].StyleInfo.CellType = "Header";
this.gridDataBoundGrid1.Model.Data.RowCount = 1;
this.gridDataBoundGrid1.Model.Rows.HeaderCount = 1;
this.gridDataBoundGrid1.Model.Rows.FrozenCount = 1;

// Hide the Column Header Cells.
this.gridDataBoundGrid1.Model.HideRows[0] = true;

// Add New Header Cells.
for(int col = 1; col <= 3; ++col)
{
    this.gridDataBoundGrid1[1, col] = this.gridDataBoundGrid1[0,1];
    this.gridDataBoundGrid1[1, col].CellType = "ColumnHeaderCell";
}
```

**[VB.NET]**

```
' Set Datasource.  
Me.gridDataBoundGrid1.DataSource = dt  
Me.gridDataBoundGrid1.Model.BaseStylesMap("Column  
Header").StyleInfo.CellType = "Header"  
Me.gridDataBoundGrid1.Model.Data.RowCount = 1  
Me.gridDataBoundGrid1.Model.Rows.HeaderCount = 1  
Me.gridDataBoundGrid1.Model.Rows.FrozenCount = 1  
  
' Hide the Column Header Cells.  
Me.gridDataBoundGrid1.Model.HideRows(0) = True  
  
' Add New Header Cells.  
Dim col As Integer  
For col = 1 To 3 Step + 1  
    Me.gridDataBoundGrid1(1, col) = Me.gridDataBoundGrid1(0,1)  
    Me.gridDataBoundGrid1(1, col).CellType = "ColumnHeaderCell"  
Next
```

2. In the CellClick Event perform the sorting.

**Example**

**[C#]**

```
private void gridDataBoundGrid1_CellClick(object sender,  
Syncfusion.Windows.Forms.Grid.GridCellEventArgs e)  
{  
    // Check for Column Header Cell.  
    if(this.gridDataBoundGrid1[e.RowIndex,e.ColIndex].CellType ==  
"ColumnHeaderCell")  
    {  
        if(this.gridDataBoundGrid1[e.RowIndex,e.ColIndex].HasTag )  
        {  
            // If Column values are not in sorted order, then Sort  
            them in ascending order.  
  
        if(this.gridDataBoundGrid1[e.RowIndex,e.ColIndex].Tag.ToString() ==  
"None")  
        {  
  
this.gridDataBoundGrid1[e.RowIndex,e.ColIndex].Tag =  
ListSortDirection.Ascending;  
                SortColumn(e.RowIndex,e.ColIndex);  
        }  
    }
```

```
// If Column values are sorted in ascending order,  
then Sort them in descending order.  
else if((ListSortDirection  
)this.gridDataBoundGrid1[e.RowIndex, e.ColIndex].Tag ==  
ListSortDirection.Ascending)  
{  
  
    this.gridDataBoundGrid1[e.RowIndex, e.ColIndex].Tag  
= ListSortDirection.Descending ;  
    SortColumn(e.RowIndex, e.ColIndex);  
}  
  
// If Column values are sorted in descending order,  
then remove Sorting.  
else  
if((ListSortDirection)this.gridDataBoundGrid1[e.RowIndex,  
e.ColIndex].Tag == ListSortDirection.Descending)  
{  
  
this.gridDataBoundGrid1[e.RowIndex, e.ColIndex].Tag = "None";  
    SortColumn(e.RowIndex, e.ColIndex);  
}  
}  
else  
{  
    this.gridDataBoundGrid1[e.RowIndex, e.ColIndex].Tag  
= ListSortDirection.Ascending ;  
    SortColumn(e.RowIndex, e.ColIndex);  
}  
}  
}  
}
```

**[VB.NET]**

```
Private Sub gridDataBoundGrid1_CellClick(ByVal sender As Object, ByVal  
e As Syncfusion.Windows.Forms.Grid.GridCellEventArgs)  
  
    ' Check for Column Header Cell.  
    If Me.gridDataBoundGrid1(e.RowIndex, e.ColIndex).CellType =  
"ColumnHeaderCell" Then  
        If Me.gridDataBoundGrid1(e.RowIndex, e.ColIndex).HasTag Then  
  
            ' If Column values are not in sorted order, then Sort them in  
ascending order.  
            If  
Me.gridDataBoundGrid1(e.RowIndex, e.ColIndex).Tag.ToString() = "None"  
Then
```

```

        Me.gridDataBoundGrid1(e.RowIndex,e.ColumnIndex).Tag =
ListSortDirection.Ascending
            SortColumn(e.RowIndex,e.ColumnIndex)

        ' If Column values are sorted in ascending order, then Sort
        them in descending order.
        Else If
CType(Me.gridDataBoundGrid1(e.RowIndex,e.ColumnIndex).Tag =
ListSortDirection.Ascending,ListSortDirection) Then
            Me.gridDataBoundGrid1(e.RowIndex,e.ColumnIndex).Tag =
ListSortDirection.Descending
            SortColumn(e.RowIndex,e.ColumnIndex)

        ' If Column values are sorted in descending order, then
        remove Sorting.
        Else If
CType(Me.gridDataBoundGrid1(e.RowIndex,e.ColumnIndex).Tag =
ListSortDirection.Descending,ListSortDirection) Then
            Me.gridDataBoundGrid1(e.RowIndex,e.ColumnIndex).Tag = "None"
            SortColumn(e.RowIndex,e.ColumnIndex)
        End If
        Else
            Me.gridDataBoundGrid1(e.RowIndex,e.ColumnIndex).Tag =
ListSortDirection.Ascending
            SortColumn(e.RowIndex,e.ColumnIndex)
        End If
    End If

End Sub

```

## 5.2.23 How to Use a Combo Box in a Column

### Introduction

The control type of a cell is part of the cell style and is determined by the **GridStyleInfo.CellType** property. The items shown in the dropdown list can be provided in two ways.

- Create a StringCollection object that will hold your choices and then set this StringCollection in the **GridStyleInfo.ChoiceList** property for the cell.
- Have an IList object that will hold object entries that have public properties (such as a DataTable object with its columns serving as public properties).

In the second case, use the **GridStyleInfo.DataSource**, **DisplayMember** and **ValueMember** properties to set the datasource for the drop list. In addition to setting the cell type, **ChoiceList**, **datasource**, **DisplayMember** and **ValueMember**, the **DropDownStyle** property of **GridStyleInfo** controls the editing behavior of the combobox cell. You can also use the **GridStyleInfo.ShowButton** property to control when the combo box button is visible.

### Example

Here is the code that sets column 2 to be a combo box with the dropdownlist being set through the styles **ChoiceList** property. To access a column's style, you must use either the **GridDataBoundGrid.GridBoundColumns** or **GridDataBoundGrid.Binder.InternalColumn** depending upon whether you have explicitly added the **GridBoundColumns** or not.

#### [C#]

```
// Required to access StringCollection.  
using System.Collections.Specialized;  
  
//...  
// Create the list.  
StringCollection items = new StringCollection();  
items.AddRange(new string[]{"One", "Two", "Three", "Four", "Five"});  
  
// Set the style properties.  
GridStyleInfo style =  
this.gridDataBoundGrid1.GridBoundColumns[1].StyleInfo;  
style.CellType = "ComboBox";  
style.ChoiceList = items;  
styleCellValue = "Five";  
  
// True dropdown - no editing.  
style.DropDownStyle = GridDropDownStyle.Exclusive;
```

#### [VB .NET]

```
' Required to access StringCollection.  
Imports System.Collections.Specialized  
  
'...  
' Create the list.  
Dim items As New StringCollection  
items.AddRange(New String() {"One", "Two", "Three", "Four", "Five"})  
  
' Set the style properties.  
Dim style As GridStyleInfo =  
Me.gridDataBoundGrid1.GridBoundColumns(1).StyleInfo
```

```
style.CellType = "ComboBox"
style.ChoiceList = items
styleCellValue = "Five"

' True dropdown - no editing.
style.DropDownStyle = GridDropDownStyle.Exclusive
```

Here is the code that will set column 2 to a combobox setting the items in the combobox through a **DataTable** datasource.

**[C#]**

```
// Assume this.dt is a DataTable object with at least 2 columns named
// "id" and "display".

// Set the style properties.
GridStyleInfo style =
this.gridDataBoundGrid1.GridBoundColumns[1].StyleInfo;
style.CellType = "ComboBox";
style.DataSource = dt;

// Displayed in the grid cell.
style.DisplayMember = "display";

// Value in the grid cell.
style.ValueMember = "id";
style.DropDownStyle = GridDropDownStyle.AutoComplete;
```

**[VB .NET]**

```
' Assume this.dt is a DataTable object with at least 2 columns named
// "id" and "display".

' Set the style properties.
Dim style As GridStyleInfo =
Me.gridDataBoundGrid1.GridBoundColumns(1).StyleInfo
style.CellType = "ComboBox"
style.DataSource = dt

' Displayed in the grid cell.
style.DisplayMember = "display"

' Value in the grid cell.
style.ValueMember = "id"
style.DropDownStyle = GridDropDownStyle.AutoComplete
```

## 5.2.24 How to Use a ColorEdit Control in a Column and Retrieve its Value

### Introduction

Set the **CellType** property in the cell style to "ColorEdit" and the **text** property to the appropriate value. To access a column's style, use either the **GridDataBoundGrid.GridBoundColumns** or **GridDataBoundGrid.Binder.InternalColumn** depending upon whether you have explicitly added the **GridBoundColumns** or not.

### Example

#### [C#]

```
GridStyleInfo style = gridDataBoundGrid1.GridBoundColumns[1].StyleInfo;

// Set controltype.
style.CellType = "ColorEdit";

// Set the initial value to Color.Aqua or to set a RGB color, use
// something like style.Text = "2, 12, 255";
// Set initial value to
RGB(2,12,255)
style.Text = "Aqua";

//....
// To retrieve a color object from this cell, use code such as
Color c =
(Color)System.ComponentModel.TypeDescriptor.GetConverter(typeof(Color))
.ConvertFromString(gridDataBoundGrid1[2, 2].Text);
```

#### [VB .NET]

```
Dim style As GridStyleInfo =
gridDataBoundGrid1.GridBoundColumns(1).StyleInfo

' Set controltype.
style.CellType = "ColorEdit"

' Set the initial value to Color.Aqua or to set a RGB color, use
// something like style.Text = "2, 12, 255"
' Set initial value to
RGB(2,12,255)
```

```

style.Text = "Aqua"

'....
' To retrieve a color object from this cell, use code such as
Dim c As Color =
CType(System.ComponentModel.TypeDescriptor.GetConverter(GetType(Color)))
.ConvertFromString(gridDataBoundGrid1(2, 2).Text), Color)

```

## 5.2.25 How to Use a PushButton in a Column and Catch the User Clicking It

### Introduction

Set the [CellType](#) property in the column style to "PushButton" and handle the grids CellButtonClicked event. Use the Description property of the column style to specify the text that is displayed on the button. To access a column's style, use either the [GridDataBoundGrid.GridBoundColumns](#) or [GridDataBoundGrid.Binder.InternalColumn](#) depending upon whether you have explicitly added the GridBoundColumns or not.

### Example

#### [C#]

```

GridStyleInfo style = gridDataBoundGrid1.GridBoundColumns[1].StyleInfo;
style.CellType = "PushButton";
style.Description = "Push Me!";

gridDataBoundGrid1.CellButtonClicked += new
GridCellButtonClickedEventHandler(grid_CellButtonClicked);

//...
private void grid_CellButtonClicked(object sender,
GridCellButtonClickedEventArgs e)
{
    string s = string.Format("You clicked ({0},{1}).", e.RowIndex,
e.ColIndex);
    MessageBox.Show(s);
}

```

#### [VB .NET]

```

Dim style As GridStyleInfo =

```

```
gridDataBoundGrid1.GridBoundColumns(1).StyleInfo
    style.CellType = "PushButton"
    style.Description = "Push Me!"

    AddHandler gridDataBoundGrid1.CellButtonClicked, AddressOf
grid_CellButtonClicked

    ...
    Private Sub grid_CellButtonClicked(sender As Object, e As
GridCellButtonClickedEventArgs)
        Dim s As String = String.Format("You clicked ({0},{1}).",
e.RowIndex, e.ColumnIndex)
        MessageBox.Show(s)

    End Sub
    ' Grid_CellButtonClicked
```

## 5.2.26 How to Get Selected Rows in GridDataBoundGrid

To get the selected row use the `GetSelectedRows` method .It returns a `GridRangeInfoList` object, an array of `GridRangeInfo` objects. This method has two arguments,

- `bRangeRowsOnly`

**True** - Only selected row will be returned,

**False** - If you want to treat single range selection as a full row selection.

- `ConsiderCurrentCell`

**True** - If current cell should be returned as selected range.

### Example

[C#]

```
GridRangeInfoList ranges = this.gridDataBoundGrid1.Selections.
GetSelectedRows(true, false);

foreach (GridRangeInfo range in ranges)
```

```
{  
  
for (int i = range.Top; i <= range.Bottom; ++i)  
  
{  
  
Console.WriteLine(string.Format("Row {0} selected", i));  
  
}  
  
}
```

**[VB]**

```
Dim ranges As GridRangeInfoList = Me.gridDataBoundGrid1 .Selections.  
GetSelectedRows(True, False)  
  
For Each range As GridRangeInfo In ranges  
  
    For i As Integer = range.Top To range.Bottom  
        Console.WriteLine(String.Format("Row {0}  
selected", i))  
    Next i  
Next range
```

## 5.2.27 How to Get Selected Ranges in GridDataBoundGrid

To get the selected range of cells, use the **GetSelectedRanges** method .It returns the list with selected cells. This method has two arguments,

- **ranges** - It gets the range of cells to be selected.
- **ConsiderCurrentCell**  
**True** - If current cell should be treated as selected range.

## Example

[C#]

```
GridRangeInfoList rangeList = null;  
this.gridDataBoundGrid1.Selections.GetSelectedRanges(out  
    rangeList, false);  
  
if (rangeList.Count > 0)  
{  
    foreach (GridRangeInfo range in rangeList)  
    {  
        for (int row = range.Top; row <= range.Bottom; ++row)  
        {  
            for (int col = range.Top; col <= range.Bottom;  
++col)  
            {  
  
                Console.WriteLine(String.Format("Cell {0}, {1} selected", row,  
                    col));  
            }  
        }  
    }  
}
```

[VB]

```
Dim rangeList As GridRangeInfoList = Nothing  
  
Me.gridDataBoundGrid1.Selections.GetSelectedRanges(rangeList, False)  
  
If rangeList.Count > 0 Then  
    For Each range As GridRangeInfo In rangeList  
        For row As Integer = range.Top To  
            range.Bottom  
                For col As Integer = range.Left To  
                    range.Right
```

```
Console.WriteLine(String.Format("Cell {0}, {1} selected", row,
col))

    Next col

    Next row

    Next range

End If
```

### 5.2.28 How to Determine that No Cell is Selected

To determine whether the cell is selected or not, use the **GetSelectedRange** method. It returns the list with selected range. If it returns the range as zero, then no cell is selected.

- **ranges** - It sets the range of cells to be selected.
- **ConsiderCurrentCell**

**True** - If the current cell should be treated as selected range.

#### Example

[C#]

```
GridRangeInfoList rangeList = null;

    this.gridDataBoundGrid1.Selections.GetSelectedRanges(out rangeList,
false);

    if (rangeList.Count == 0)

    {

        Console.WriteLine("no selection");

    }
```

[VB]

```
Dim rangeList As GridRangeInfoList = Nothing
Me.gridDataBoundGrid1.Selections.GetSelectedRanges(rangeList, False)
If rangeList.Count = 0 Then
    Console.WriteLine("no Selection")
End If
```

## 5.2.29 How to Disable Sorting While Record Added

When GridDataBoundGrid's data source is *BindingList*, you can perform sorting only using the *WrapperClasses*.

In the following example the *CellClick* event is used to customize sorting with the *WrapperClasses*:

[C#]

```
void gridDataBoundGrid1_CellClick(object sender, GridCellEventArgs e)
{
    if (e.RowIndex == 0 && e.ColumnIndex > 0)
    {
        int filed =
this.gridDataBoundGrid1.Binder.ColumnIndexToField(e.ColumnIndex);
        string name =
this.gridDataBoundGrid1.Binder.InternalColumns[filed].MappingName;
        WrapperClass lw = this.gridDataBoundGrid1.DataSource as
WrapperClass;
        lw.Sort(name, true);
        e.Cancel = true;
        this.gridDataBoundGrid1.Refresh();
    }
}
```

**[VB]**

```

Private Sub gridDataBoundGrid1_CellClick(ByVal sender As Object, ByVal e As
GridCellEventArgs)
    If e.RowIndex = 0 AndAlso e.ColumnIndex > 0 Then
        Dim filed As Integer =
Me.gridDataBoundGrid1.Binder.ColumnIndexToField(e.ColumnIndex)
        Dim name As String =
Me.gridDataBoundGrid1.Binder.InternalColumns(filed).MappingName
        Dim lw As WrapperClass = TryCast(Me.gridDataBoundGrid1.DataSource,
WrapperClass)
        lw.Sort(name, True)
        e.Cancel = True
        Me.gridDataBoundGrid1.Refresh()
    End If
End Sub

```

### 5.2.30 How to Apply Filtering on the GridDataBound Grid based on a Node Selected in the Tree View?

The GridDataBound grid is designed as a grid to bind data sources such as data table, string collection, data set, and data view. The filter bar can be enabled and hidden in the grid. Before selecting a node in the Tree view, the grid will load all the data from the data source. The **TreeView\_AfterSelect** event is used to determine the tree view node and perform filter operation on the grid control corresponding to the node that you selected in the Tree view.

The following code example illustrates how to apply filtering on the GridDataBound grid based on a node selected in the Tree view.

**[C#]**

```

//Include the below four lines of code in the constructor

treeViewAdv1.ExpandAll();
filter = new GridFilterBar();
//Associates Grid with the filter
filter.WireGrid(this.gridDataBoundGrid1);
//Hide the filter bar row
this.gridDataBoundGrid1.Model.Rows.Hidden[1] = true;

```

```
//Event Handler

private void treeViewAdv1_AfterSelect(object sender, EventArgs e)
{
    TreeNodeAdv node = treeViewAdv1.SelectedNode;
    if (node != null)
    {
        string filterString = string.Format("CategoryId = {0}",
Convert.ToInt32(node.Tag));

//Apply filter on grid

filter.RowFilter = filterString;
this.gridDataBoundGrid1.Refresh();
    }
}
```

**[VB.NET]**

```
//Include the below four lines of code in the constructor

treeViewAdv1.ExpandAll()
filter = New GridFilterBar()
'Associates Grid with the filter
filter.WireGrid(Me.gridDataBoundGrid1)
'Hide the filter bar row
Me.gridDataBoundGrid1.Model.Rows.Hidden(1) = True

//Event Handler

Private Sub treeViewAdv1_AfterSelect(ByVal sender As Object, ByVal e As
EventArgs)
    Dim node As TreeNodeAdv = treeViewAdv1.SelectedNode
```

```
If node IsNot Nothing Then  
    Dim filterString As String = String.Format("CategoryId = {0}",  
    Convert.ToInt32(node.Tag))  
  
    'Apply filter on grid  
  
    filter.RowFilter = filterString  
    Me.gridDataBoundGrid1.Refresh()  
End If  
End Sub
```

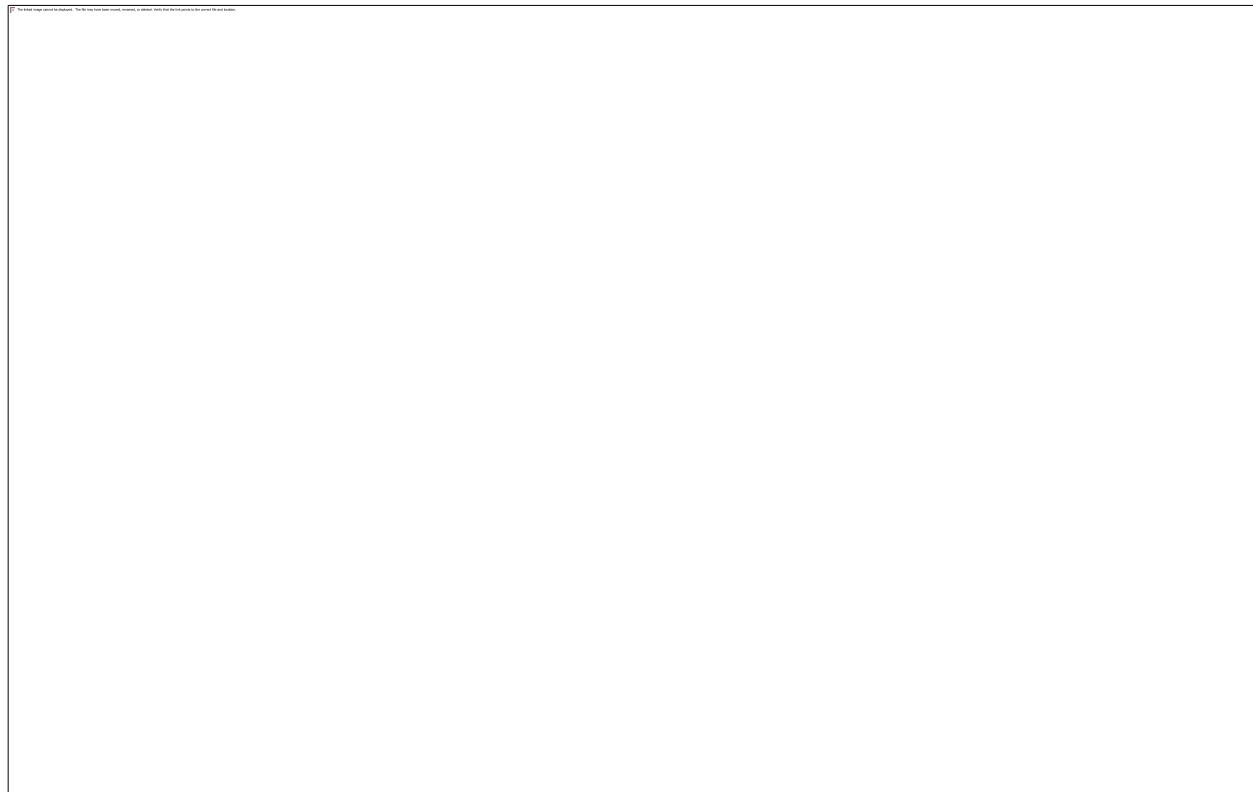


Figure 491: Apply filtering for the Data View and performing filter operation on the Grid

### Sample Link

A sample can be downloaded from the following location:

[http://www.syncfusion.com/downloads/Support/DirectTrac/91763/Syncfusion\\_FTP1-989352783.zip](http://www.syncfusion.com/downloads/Support/DirectTrac/91763/Syncfusion_FTP1-989352783.zip)

### 5.2.31 How to select all the contents in a cell after double-clicking on the contents

The **CurrentCellControlDoubleClick** event can be used to select the contents of a cell by double-clicking it.

The current cell is selected by using **gridControl1.CurrentCell.Renderer**. The length of the content is measured and the contents are selected using the **Select()** method of the inner textbox cell, if it is a textbox cell type.

Here is the sample code in C# and in VB:

[C#]

```
private void gridControl1_CurrentCellControlDoubleClick(object sender,
ControlEventArgs e)
{
    GridTextBoxCellRenderer ren = this.gridControl1.CurrentCell.Renderer as
GridTextBoxCellRenderer;

    if (ren.TextBox.TextLength > 0)
    {
        ren.TextBox.Select(0, ren.TextBox.TextLength);
    }
}
```

[VB]

```
Private Sub gridControl1_CurrentCellControlDoubleClick(ByVal sender As Object,
ByVal e As ControlEventArgs)

Dim ren As GridTextBoxCellRenderer =
TryCast(Me.gridControl1.CurrentCell.Renderer, GridTextBoxCellRenderer)

    If ren.TextBox.TextLength > 0 Then

        ren.TextBox.Select(0, ren.TextBox.TextLength)

    End If
End Sub
```

If the contents have to be selected using the F2 key, the **CurrentCellKeyDown** event can be handled and the contents can be selected.

Sample code:

```
[C#]
private void gridControl1_CurrentCellKeyDown(object sender,
System.Windows.Forms.KeyEventArgs e)

{
    if (e.KeyCode == Keys.F2)
    {

        ((GridTextBoxCellRenderer)this.gridControl1.CurrentCell.Renderer)
.TextBox.SelectAll();
    }
}
```

```
[VB]
```

```
Private Sub gridControl1_CurrentCellKeyDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyEventArgs)

If e.KeyCode = Keys.F2 Then

    CType(Me.gridControl1.CurrentCell.Renderer,
GridTextBoxCellRenderer).TextBox.SelectAll()

End If

End Sub
```

### 5.2.32 How to resolve the “Object reference not set to an instance of an object” error generation in GridDataBoundGrid Designer

The GridDataBoundGrid is a grid that is bound to data sources such as Data Tables, String Collections, Data Sets, and Data Views. The Hidden, Position and Width properties of GridDataBoundColumn generate the “Object reference not set to an instance of an object” error on designer code generation. This issue has been resolved for version 10.1.0.44 and following versions by setting the **DesignerSerializationVisibility** for these codes to *Hidden*.

However, if the application is migrated from version 9.4.0.62 or earlier versions to the latest version, the designer code throws this exception. This can be fixed by removing the “DesignerSerializationVisibility” part of code from the designer.

### 5.2.33 How to achieve Numeric Column Sorting in GridDataBoundGrid

To perform a custom sort through the DataView, the object in the column should implement **IComparable** interface. Once this is done, the DataView will use the IComparable interface implementation to sort the column instead of using the default sort of the control.

**[C#]**

```
public class StringLenComparer : IComparer
{
    #region IComparer Members
    public int Compare(object x, object y)
    {
        string xs = x.ToString(); ;
        string ys = y.ToString(); ;
        return xs.Length - ys.Length;
    }
    #endregion
}
public class IntComparer : IComparer
{
    #region IComparer Members
    public int Compare(object x, object y)
    {
        if(x == null && y == null)
            return 0;
        else if(x == null)
            return -1;
        else if(y == null)
            return 1;
        else
            return int.Parse(x.ToString()) - int.Parse(y.ToString());
    }
    #endregion
}
```

**[VB]**

```
Public Class StringLenComparer Implements IComparer
    #Region "IComparer Members"
    Public Function Compare(ByVal x As Object, ByVal y As
Object) As Integer
        Dim xs As String = x.ToString()
        Dim ys As String = y.ToString()
        Return xs.Length - ys.Length
    End Function
End Class
```

```
End Function
#End Region
End Class

Public Class IntComparer Implements IComparer
#Region "IComparer Members"
    Public Function Compare(ByVal x As Object, ByVal y As
Object) As Integer
        If x Is Nothing AndAlso y Is Nothing Then
            Return 0
        ElseIf x Is Nothing Then
            Return -1
        ElseIf y Is Nothing Then
            Return 1
        Else
            Return Integer.Parse(x.ToString()) -
                Integer.Parse(y.ToString())
        End If
    End Function
#End Region
End Class
```

#### **Sample Link**

To view a sample on this, see [Numeric Column Sorting](#).

## 5.3 GridGrouping Control

This section discusses specific solutions for very specific tasks under the below sections:

### 5.3.1 General

This section comprises the following topics:

#### 5.3.1.1 How to Access the Corresponding Parent Table's Row from the Child

Get the table's current element. Then cast this element to the **GridNestedTable**. You can now access the parent table for the current element. This is a recursive process.

##### **Example**

[C#]

```
Element el =
this.gridGroupingControl1.TableControl.Table.CurrentElement;
```

```
if(el != null)
{
    if(el is GridRecord)
    {
        Console.WriteLine("no parent row...");
    }

    // Check for a child table.
    else if(el is GridNestedTable)
    {
        GridNestedTable gnt = el as GridNestedTable;
        GridNestedTable gnt1 = gnt;
        while(gnt1 != null && gnt1.ChildTable != null)
        {
            gnt = gnt1;

            // Get the handle for parent table.
            gnt1 = gnt.ChildTable.ParentTable.CurrentElement as
GridNestedTable;
        }

        // Retrieve the corresponding parent table's record.
        DataRowView drv = gnt.ParentRecord.GetData() as
DataRowView;

        // Show column 2.
        Console.WriteLine(drv[1].ToString());
    }
}
```

**[VB.NET]**

```
Dim el As Element =
Me.GridGroupingControl1.TableControl.Table.CurrentElement
If Not (el Is Nothing) Then
    If TypeOf el Is GridRecord Then
        Console.WriteLine("no parent row...")
    Else

        ' Check for a child table.
        If TypeOf el Is GridNestedTable Then
            Dim gnt As GridNestedTable = el
            Dim gnt1 As GridNestedTable
            If TypeOf gnt Is GridNestedTable Then
                gnt1 = gnt
            Else
```

```
        gnt1 = Nothing
    End If

    While Not (gnt1 Is Nothing) AndAlso Not
(gnt1.ChildTable Is Nothing)
        gnt = gnt1
        If TypeOf gnt.ChildTable.ParentTable.CurrentElement
Is GridNestedTable Then

            ' Get the handle for parent table.
            gnt1 =
gnt.ChildTable.ParentTable.CurrentElement
        Else
            gnt1 = Nothing
        End If
    End While

    ' Retrieve the corresponding parent table's record.
    Dim drv As DataRowView = gnt.ParentRecord.GetData()

    ' Show column 2.
    Console.WriteLine(drv(1).ToString())
End If
End If
End If
```

### 5.3.1.2 How to access the current record

To access the current record, use the following code.

#### [C#]

```
Record rec = this.gridGroupingControl1.Table.CurrentRecord;
Trace.WriteLine(rec.ToString());
```

#### [VB.NET]

```
Dim rec As Record = Me.gridGroupingControl1.Table.CurrentRecord
Trace.WriteLine(rec.ToString())
```

### 5.3.1.3 How to add and format unbound columns in GridGrouping control

The unbound columns can be added and formatted using the below code.

[C#]

```
// Adding unbound column in the GridGroupingControl.  
this.gridGroupingControl1.TableDescriptor.UnboundFields.Add("UnboundCol  
umn1");  
  
//Formatting the Unbound column  
this.gridGroupingControl1.TableDescriptor.Columns["UnboundColumn1"].App  
earance.AnyRecordFieldCell.CellType = "CheckBox";  
this.gridGroupingControl1.TableDescriptor.Columns["UnboundColumn1"].App  
earance.AnyRecordFieldCell.BackColor = Color.LightSteelBlue;
```

[VB]

```
'Adding an Unbound column in a GridGroupingControl  
Me.gridGroupingControl1.TableDescriptor.UnboundFields.Add("UnboundCol  
umn1")  
  
'Formatting an Unbound column in a GridGroupingControl  
Me.gridGroupingControl1.TableDescriptor.Columns("UnboundColumn1").Appe  
arance.AnyRecordFieldCell.CellType = "CheckBox"  
Me.gridGroupingControl1.TableDescriptor.Columns("UnboundColumn1").Appe  
arance.AnyRecordFieldCell.BackColor = Color.LightSteelBlue
```

### 5.3.1.4 How to change the caption text

This can be done using the below code.

[C#]

```
// Set the caption text  
// {TableName} - Displays the CaptionSection.ParentTableDescriptor.Name  
//{CategoryName} - Displays the CaptionSection.ParentGroup.Name  
//{Category} - Displays the CaptionSection.ParentGroup.Category  
//{RecordCount} - Displays the  
CaptionSection.ParentGroup.GetFilteredRecordCount()  
this.gridGroupingControl1.TopLevelGroupOptions.CaptionText = "Tablename"
```

```
is {TableName} : {Category} : {RecordCount}";
```

**[VB]**

```
' Set the caption text
' {TableName} - Displays the CaptionSection.ParentTableDescriptor.Name
'{CategoryName} - Displays the CaptionSection.ParentGroup.Name
'{Category} - Displays the CaptionSection.ParentGroup.Category
'{RecordCount} - Displays the
CaptionSection.ParentGroup.GetFilteredRecordCount()
Me.gridGroupingControll1.TopLevelGroupOptions.CaptionText = "Tablename
is {TableName} : {Category} : {RecordCount}"
```

### 5.3.1.5 How to disable the resizing of rows and columns

This can be done using the below code.

**[C#]**

```
// Code to disable the resizing of rows.
this.gridGroupingControll1.TableModel.Options.ResizeRowsBehavior =
Syncfusion.Windows.Forms.Grid.GridResizeCellsBehavior.None;

//Code to disable the column resizing
this.gridGroupingControll1.TableModel.Options.ResizeColsBehavior =
Syncfusion.Windows.Forms.Grid.GridResizeCellsBehavior.None;
```

**[VB]**

```
'Code to disable the column resizing
Me.gridGroupingControll1.TableModel.Options.ResizeColsBehavior =
Syncfusion.Windows.Forms.Grid.GridResizeCellsBehavior.None

'Code to disable the resizing of rows.
Me.gridGroupingControll1.TableModel.Options.ResizeRowsBehavior =
Syncfusion.Windows.Forms.Grid.GridResizeCellsBehavior.None
```

### 5.3.1.6 How to Efficiently Customize the Child Table / Group using a Custom Engine

When customizing the GridChildTable / GridGroup by deriving the GridChildTable / GridGroup in the custom engine, the OnInitializeVisibleCounters method and the OnEnsureInitialized method must also be overridden along with the other overrides. Otherwise the GridGroup calls into the GridGroup extend methods and sometimes bypasses the methods like IsChildVisible that you have overridden.

In the OnInitializeVisibleCounters override, the total visible elements count, the total vertical scroll distance of elements in pixels and the total custom count of the visible elements must be calculated and set to the CachedVisibleCount, CachedYamountCount and CachedVisibleCustomCount respectively. The OnEnsureInitialized override method must return false (i.e changes were detected and the object was not updated) to ensure that the object is up to date.

### 5.3.1.7 How to group a column programmatically

To group a column programmatically, use the following code.

[C#]

```
//Show the GroupDropArea  
this.gridGroupingControl1.ShowGroupDropArea = true;  
// Group by "Col1"  
this.gridGroupingControl1.TableDescriptor.GroupedColumns.Add("Col1",  
ListSortDirection.Ascending);
```

[VB.NET]

```
'Show the GroupDropArea  
Me.gridGroupingControl1.ShowGroupDropArea = True  
' Group by "Col1"  
Me.gridGroupingControl1.TableDescriptor.GroupedColumns.Add("Col1",  
ListSortDirection.Ascending)
```

### 5.3.1.8 How to Hide the Custom Option in the FilterBar DropDown

To achieve this, you need to implement a custom filterbar cell deriving the cellmodel / cellrenderer from the **GridTableFilterBarCellModel** / **GridTableFilterBarCellRenderer**.

- The **GridTableFilterBarCellModel.FillWithChoices** method is overridden to remove the string "Custom" from the list.
- The **GridTableFilterBarCellModel.Select** method is rewritten with the "Select\_withoutcustom" method, which specifies the index for the selected item. You can pass the index for the "Custom" selected item as 0.
- The **GridTableFilterBarCellModel.ListBoxMouseUp()** method is overridden to invoke the Select\_withoutcustom method instead of the **Select** method.

[C#]

```
public class GridTableFilterBarCellModel1: GridTableFilterBarCellModel
{
    public
GridTableFilterBarCellModel1(Syncfusion.Windows.Forms.Grid.GridModel
gm):base(gm)
    {
    }

    // Override FillWithChoices to remove 'Custom' option from the
list.
    public override void FillWithChoices(ListBox listBox, GridStyleInfo
style, out bool exclusive)
    {
        exclusive = true;
        GridTableCellStyleInfo tableStyleInfo = (GridTableCellStyleInfo)
style;
        object[] items = (object[])
GetFilterBarChoices(tableStyleInfo.TableCellIdentity);
        listBox.Items.Clear();
        if (items != null)
        {
            listBox.Items.Add(SelectAllText);
            foreach (object item in items)
            {
                if (item is DBNull)
                    listBox.Items.Add(SelectEmptyText);
                else if (item != null)
                    listBox.Items.Add(style.GetFormattedText(item));
            }
        }
    }

    // Customize the 'Select' method to hide the Custom option.
    public void Select_WithoutCustom(GridTableCellStyleInfoIdentity
```

```

tableCellIdentity, int index)
{
    if (index >= 0)
    {
        if (index == 0)
        {
            ResetFilterBar(tableCellIdentity);
        }
        else if (index == 1)
        {
            SelectItem(tableCellIdentity, 0);
        }
        else
        {
            SelectItem(tableCellIdentity, index-1);
        }
    }
}

public override GridCellRendererBase CreateRenderer(GridControlBase control)
{
    return new Cellrenderer(control, this);
}

}

public class Cellrenderer : GridTableFilterBarCellRenderer
{
    public Cellrenderer(GridControlBase grid, GridCellModelBase cellModel): base(grid , cellModel)
    {

    }

    public new Model.Form1.GridTableFilterBarCellModel1 Model
    {
        get
        {
            return (Model.Form1.GridTableFilterBarCellModel1)base.Model;
        }
    }

    // Override ListBoxMouseUP method to call Customized Select
    // method instead of the usual 'Select' method.
    protected override void ListBoxMouseUp(object sender,
    MouseEventArgs e)
    {
        CurrentCell.CloseDropDown(PopupCloseType.Done);
        GridTableCellStyleInfo tableStyleInfo =
}

```

```
(GridTableCellStyleInfo) this.StyleInfo;
        GridTableCellStyleInfoIdentity tableCellIdentity =
tableStyleInfo.TableCellIdentity;
        Model.Select_WithoutCustom(tableCellIdentity,
this.ListBoxPart.SelectedIndex);

        // Don't call base class - ignore.
        SetTextBoxText(GetFilterBarText(StyleInfo), false);
    }
}
```

**[VB.NET]**

```
Public Class GridTableFilterBarCellModel : Inherits
GridTableFilterBarCellModel

    Public Sub New(ByVal gm As
Syncfusion.Windows.Forms.Grid.GridModel)
        MyBase.New(gm)
    End Sub

    ' Override FillWithChoices to remove 'Custom' option from the
list.
    Public Overrides Sub FillWithChoices(ByVal listBox As ListBox,
 ByVal style As GridStyleInfo, <System.Runtime.InteropServices.Out()>
ByRef exclusive As Boolean)

        exclusive = True
        Dim tableStyleInfo As GridTableCellStyleInfo = CType(style,
GridTableCellStyleInfo)
        Dim items As Object() =
CType(GetFilterBarChoices(tableStyleInfo.TableCellIdentity), Object())
        listBox.Items.Clear()
        If Not items Is Nothing Then

            listBox.Items.Add>SelectAllText)
            For Each item As Object In items

                If TypeOf item Is DBNull Then
                    listBox.Items.Add>SelectEmptyText)
                Else If Not item Is Nothing Then

listBox.Items.Add(style.GetFormattedText(item))
                End If
            Next item
        End If
    End Sub
}
```

```
' Customize the 'Select' method to hide the Custom option.
Public Sub Select_WithoutCustom(ByVal tableCellIdentity As
GridTableCellStyleInfoIdentity, ByVal index As Integer)

    If index >= 0 Then

        If index = 0 Then

            ResetFilterBar(tableCellIdentity)

        Else If index = 1 Then

            SelectItem(tableCellIdentity, 0)

        Else

            SelectItem(tableCellIdentity, index-1)
        End If
    End If
End Sub
Public Overrides Function CreateRenderer(ByVal control As
GridControlBase) As GridCellRendererBase

    Return New Cellrenderer(control, Me)
End Function
End Class
}
Public Class Cellrenderer : Inherits GridTableFilterBarCellRenderer

    Public Sub New(ByVal grid As GridControlBase, ByVal
cellModel As GridCellModelBase)
        MyBase.New(grid, cellModel)
    End Sub

    Public Shadows ReadOnly Property Model() As
Model.Form1.GridTableFilterBarCellModel1

        Get

            Return CType(MyBase.Model,
Model.Form1.GridTableFilterBarCellModel1)
        End Get
    End Property

    ' Override ListBoxMouseUp method to call Customized Select
method instead of the usual 'Select' method.
    Protected Overrides Sub ListBoxMouseUp(ByVal sender As
Object, ByVal e As MouseEventArgs)
```

```
        CurrentCell.CloseDropDown(PopupCloseType.Done)
        Dim tableStyleInfo As GridTableCellStyleInfo =
 CType(Me.StyleInfo, GridTableCellStyleInfo)
        Dim tableCellIdentity As
GridTableCellStyleInfo.Identity = tableStyleInfo.TableCellIdentity

        Model.Select_WithoutCustom(tableCellIdentity,
Me.ListBoxPart.SelectedIndex)

        ' Don't call base class - ignore.
        SetTextBoxText(GetFilterBarText(StyleInfo), False)
    End Sub
End Class
```

### 5.3.1.9 How to Hide / Unhide the Columns in a Grouping Grid

The TableDescriptor object has a **VisibleColumns** collection that you can use to control which columns are visible. You can hide / unhide columns using the following code.

#### Example

[C#]

```
//Hide
this.gridGroupingControl1.TableDescriptor.VisibleColumns.Remove("Col1");
;

//Unhide
this.gridGroupingControl1.TableDescriptor.VisibleColumns.Add("Col1");
```

[VB]

```
'Hide
Me.gridGroupingControl1.TableDescriptor.VisibleColumns.Remove("Col1")

'Unhide
Me.gridGroupingControl1.TableDescriptor.VisibleColumns.Add("Col1")
```

### 5.3.1.10 How to Place a Checkbox in a Header Cell

A CheckBox can be placed in a header cell of the GridGroupingControl by handling the QueryCellStyleInfo event. The value of the check box has to be stored in any datatype and the SaveCellText handler saves the value from the grid.

To get this working, you need to cancel the sorting of the header cell by handling the TableControlCellClick event.

In the TableControl.MouseUp event handler the CellRenderer's mouseup has to be raised. By default the selectcells mousecontroller calls the Cellrenderer's MouseUp which in turn raises the checkboxclick. The DragGroupHeader mousecontroller which is for the header cells, doesn't call the Renderer's mouseup. So you have to explicitly call the Renderer's Mouseup from the TableControl's Mouseup checking for the MouseController type. Refer the attached sample for more details.

### Example

#### [C#]

```
this.gridGroupingControl1.TableControl.MouseUp += new  
MouseEventHandler(TableControl_MouseUp);  
  
private void TableControl_MouseUp(object sender, MouseEventArgs e)  
{  
    int row, col;  
    this.gridGroupingControl1.TableControl.PointToRowCol(new  
Point(e.X, e.Y), out row, out col);  
    GridTableCellStyleInfo style =  
this.gridGroupingControl1.TableControl.Model[row, col];  
    IMouseController controller;  
  
    this.gridGroupingControl1.TableControl.MouseControllerDispatcher.HitTes  
t(new Point(e.X, e.Y), e.Button, e.Clicks, out controller);  
    if(controller.Name == "DragGroupHeader")  
    {  
        this.gridGroupingControl1.TableControl.GetCellRenderer(row,  
col).RaiseMouseUp(row, col, e);  
    }  
}
```

#### [VB]

```
AddHandler gridGroupingControl1.TableControl.MouseUp, AddressOf  
TableControl_MouseUp  
Private Sub TableControl_MouseUp(ByVal sender As Object, ByVal e As  
MouseEventArgs)  
    Dim row, col As Integer  
    Me.gridGroupingControl1.TableControl.PointToRowCol(New
```

```
Point(e.X, e.Y), row, col)
    Dim style As GridTableCellStyleInfo =
Me.gridGroupingControl1.TableControl.Model(row, col)
    Dim controller As IMouseController

Me.gridGroupingControl1.TableControl.MouseControllerDispatcher.HitTest(
New Point(e.X, e.Y), e.Button, e.Clicks, controller)
    If controller.Name = "DragGroupHeader" Then
        Me.gridGroupingControl1.TableControl.GetCellRenderer(row,
col).RaiseMouseUp(row, col, e)
    End If
End Sub
```

### 5.3.1.11 How to Place a UserControl in a Header Cell

To have a User Control in the GridGroupingControl, the GridGenericCellModel and GridGenericCellRenderer classes must be derived and the User Control designed must be made as a cellmodel in the GridGroupingControl using these derived classes.

The following are the steps that need to be followed to do this.

**Step 1:** The UserControl form is added to the project and the required controls are dropped into it.

**Step 2:** The GridGenericCellModel and GridGenericCellRenderer classes are derived and the UserControl is drawn accordingly to fit into the required cell header.

[C#]

```
public class CustomControlCellRenderer: GridGenericControlCellRenderer
{
    Control host = null;
    public CustomControlCellRenderer(GridControlBase grid,
GridCellModelBase cellModel)
        : base(grid, cellModel)
    {
        SupportsFocusControl = true;
    }
    protected override void OnDraw(Graphics g, Rectangle
clientRectangle, int rowIndex, int colIndex, GridStyleInfo style)
    {
```

```
        if (host == null)
    {
        host = new Control();
        Grid.Controls.Add(host);
    }
    Button c = CustomControlCellModel.CustomControl;
    FixControlParent(c);
    c.Bounds = clientRectangle;
    c.Refresh();
}
}
```

**[VB]**

```
Public Class CustomControlCellRenderer Inherits
GridGenericControlCellRenderer
    Private host As Control = Nothing
    Public Sub New(ByVal grid As GridControlBase, ByVal cellModel As
GridCellModelBase)
        MyBase.New(grid, cellModel)
        SupportsFocusControl = true
    End Sub
    Protected Overrides Sub OnDraw(ByVal g As Graphics, ByVal
clientRectangle As Rectangle, ByVal rowIndex As Integer, ByVal colIndex
As Integer, ByVal style As GridStyleInfo)
        If (host = Nothing) Then
            host = New Control
            Grid.Controls.Add(host)
        End If
        Dim c As Button = CustomControlCellModel.CustomControl
        FixControlParent(c)
        c.Bounds = clientRectangle
        c.Refresh()
    End Sub
End Class
```

**Step 3:** A new CellType is created by adding the instance of the derived generic cell model class into the GridGroupingControl's cellmodels.

**[C#]**

```
CustomControlCellModel.CustomControl = new Button();
this.gridGroupingControl1.Table.TableModel.CellModels.Add("UserControl"
,new CustomControlCellModel(this.gridGroupingControl1.TableModel));
```

**[VB]**

```
CustomControlCellModel.CustomControl = New Button  
Me.gridGroupingControl1.Table.TableModel.CellModels.Add("UserControl",  
New CustomControlCellModel(Me.gridGroupingControl1.TableModel))
```

### 5.3.1.12 How to Prevent Columns Resizing for Child Tables

To prevent the GridGroupingControl's child table / grandchild table's columns from getting resized, you must handle the TableControlResizingColumns event of the grid. In the event handler check for the e.TableDescriptor.Name and cancel the event by setting the e.Cancel to true. The following code snippet cancels the re sizing of the Child table.

**[C#]**

```
// To prevent re sizing columns for all child tables.  
private void gridGroupingControl1_TableControlResizingColumns(object  
sender, Syncfusion.Windows.Forms.Grid.Grouping.GridTableControlResizing  
ColumnsEventArgs e)  
{  
    if(this.gridGroupingControl1.TableDescriptor.Name !=  
e.TableControl.TableDescriptor.Name)  
    {  
        e.Inner.Cancel = true;  
    }  
}  
//To prevent re sizing columns for any particular child table.  
private void gridGroupingControl1_TableControlResizingColumns(object  
sender, Syncfusion.Windows.Forms.Grid.Grouping.GridTableControlResizing  
ColumnsEventArgs e)  
{  
    if(e.TableControl.TableDescriptor.Name = "ChildTable")  
    {  
        e.Inner.Cancel = true;  
    }  
}
```

**[VB]**

```
' To prevent re sizing columns for all child tables.  
Private Sub gridGroupingControl1_TableControlResizingColumns(ByVal  
sender As Object, ByVal e As  
Syncfusion.Windows.Forms.Grid.Grouping.GridTableControlResizingColumns
```

```
EventArgs) Handles gridGroupingControl1.TableControlResizingColumns
    If Me.gridGroupingControl1.TableDescriptor.Name <>
e.TableControl.TableDescriptor.Name Then
        e.Inner.Cancel = True
    End If
End Sub
' To prevent re sizing columns for any particular child table.
Private Sub gridGroupingControl1_TableControlResizingColumns(ByVal
sender As Object, ByVal e As
Syncfusion.Windows.Forms.Grid.Grouping.GridTableControlResizingColumns
EventArgs)
    If e.TableControl.TableDescriptor.Name = "ChildTable" Then
        e.Inner.Cancel = True
    End If
End Sub
```

### 5.3.1.13 How to reject the changes made to the GridGroupingControl

By default, any changes made to the GridGroupingControl will affect the underlying data source. In order to cancel the changes, you can make use of the **RejectChanges** method to reject the recent changes made to the data source. Also, ensure that the **AcceptChanges** method is called after the date source is filled, as the RejectChanges method will roll back all the changes made to the data source, since the last time the AcceptChanges method was called.

[C#]

```
private void button1_Click(object sender, EventArgs e)
{
    DataTable dt = this.gridGroupingControl1.DataSource as DataTable;
    dt.RejectChanges();
}
```

[VB]

```
Private Sub button1_Click(ByVal sender As Object, ByVal e As EventArgs)
Dim dt As DataTable = TryCast(Me.gridGroupingControl1.DataSource,
DataTable)
dt.RejectChanges()
End Sub
```

### 5.3.1.14 How to resize the columns to fit in a page while exporting to PDF

To resize columns to fit in a page, you can use the property **ExportRange** of the **GridPDFConverter**. The property is used to set number of rows to be exported in a page. The **ExportToPdfWithMerge** method is used to export the grid.

#### Example

C#

```
GridPDFConverter pdfConvertor = new GridPDFConverter();

// Range of rows to be exported in a PDF file
pdfConvertor.ExportRange = 40;

//Merged Export
pdfConvertor.ExportToPdfWithMerge(ref doc,
this.gridGroupingControl1.TableControl );

//To resize the column to fit in PDF
float tempPercent = currentPercent;
this.gridGroupingControl1.BeginUpdate();
PdfDocument doc = new PdfDocument();
Hashtable ht = new Hashtable();
for (int i = 0; i <
gridGroupingControl1.TableDescriptor.Columns.Count; i++)
{
    ht.Add(i,
gridGroupingControl1.TableDescriptor.Columns[i].Width)
    float gridWidth =
this.gridGroupingControl1.TableModel.ColWidths.GetTotal(0,
gridGroupingControl1.TableDescriptor.Columns.Count);
    if (gridWidth > doc.PageSettings.Width + 80)
    {
        float scale = (gridWidth / (doc.PageSettings.Width + 80))
- currentPercent;
        zoomGrid(currentPercent - scale);
    }
}
```

**VB**

```
Dim pdfConvertor As New GridPDFConverter()

'Range of rows to be exported in a PDF file
pdfConvertor.ExportRange = 40

'Merged Export
pdfConvertor.ExportToPdfWithMerge(doc,
Me.gridGroupingControl1.TableControl)

'To Resize the column to fit in PDF
Dim tempPercent As Single = currentPercent
Me.gridGroupingControl1.BeginUpdate()
Dim doc As New PdfDocument()
Dim ht As New Hashtable()
For i As Integer = 0 To
gridGroupingControl1.TableDescriptor.Columns.Count - 1
    ht.Add(i, gridGroupingControl1.TableDescriptor.Columns(i).Width)
Next i
Dim gridWidth As Single =
Me.gridGroupingControl1.TableModel.ColWidths.GetTotal(0,
gridGroupingControl1.TableDescriptor.Columns.Count)
If gridWidth > doc.PageSettings.Width + 80 Then
    Dim scale As Single = (gridWidth /
(doc.PageSettings.Width + 80)) - currentPercent
    zoomGrid(currentPercent - scale)
End If
```

### 5.3.1.15 How to Make Use of the Journal Control using GridGrouping Controls

You can use the GridGroupingControl, TextBox, RecordNavigationBar and Button control to make use of the Journal control. The GridGroupingControl is designed as a grid to bind the data source. The navigation bar is used to browse the records in the grid. The navigation bar can be enabled by setting the **ShowNavigationBar** property as **true**. Preview operation can be performed through the Button control. The **QueryCellInfo** event is used to display a text box value in the preview cell.

### Properties

Property	Description
ShowNavigationBar	Describes whether to show the Navigation Bar.
ShowRecordPreviewRow	Indicates whether the nested table has preview row.

The following code example illustrates how to make use of Journal control using GridGrouping controls.

[C#]

```
this.gridGroupingControll.ShowNavigationBar = true;

// Button control

this.gridGroupingControll.TableOptions.ShowRecordPreviewRow =
!this.gridGroupingControll.TableOptions.ShowRecordPreviewRow;

// Event Handler

this.gridGroupingControll.QueryCellStyleInfo += new
Syncfusion.Windows.Forms.Grid.Grouping.GridTableCellStyleInfoEventHandler(gridGroupingControll_QueryCellStyleInfo);

void gridGroupingControll_QueryCellStyleInfo(object sender,
Syncfusion.Windows.Forms.Grid.Grouping.GridTableCellStyleInfoEventArgs e)
{
    if (e.TableCellIdentity.TableCellType ==
GridTableCellType.RecordPreviewCell)
    {
```

```
        Element el = e.TableCellIdentity.DisplayElement;
        e.Style.CellValue = richTextBox1.Text;
    }
}
```

**[VB.NET]**

```
Private Me.gridGroupingControl1.ShowNavigationBar = True

// Button control

Me.gridGroupingControl1.TableOptions.ShowRecordPreviewRow = Not
Me.gridGroupingControl1.TableOptions.ShowRecordPreviewRow

// Event Handler

Private Me.gridGroupingControl1.QueryCellStyleInfo += New
Syncfusion.Windows.Forms.Grid.Grouping.GridTableCellStyleInfoEventHandler(AddressOf gridGroupingControl1_QueryCellStyleInfo)

Private Sub gridGroupingControl1_QueryCellStyleInfo(ByVal sender As
Object, ByVal e As
Syncfusion.Windows.Forms.Grid.Grouping.GridTableCellStyleInfoEventArgs)
    If e.TableCellIdentity.TableCellType =
GridTableCellType.RecordPreviewCell Then
        Dim el As Element =
e.TableCellIdentity.DisplayElement
        e.Style.CellValue = richTextBox1.Text
    End If
End Sub
```

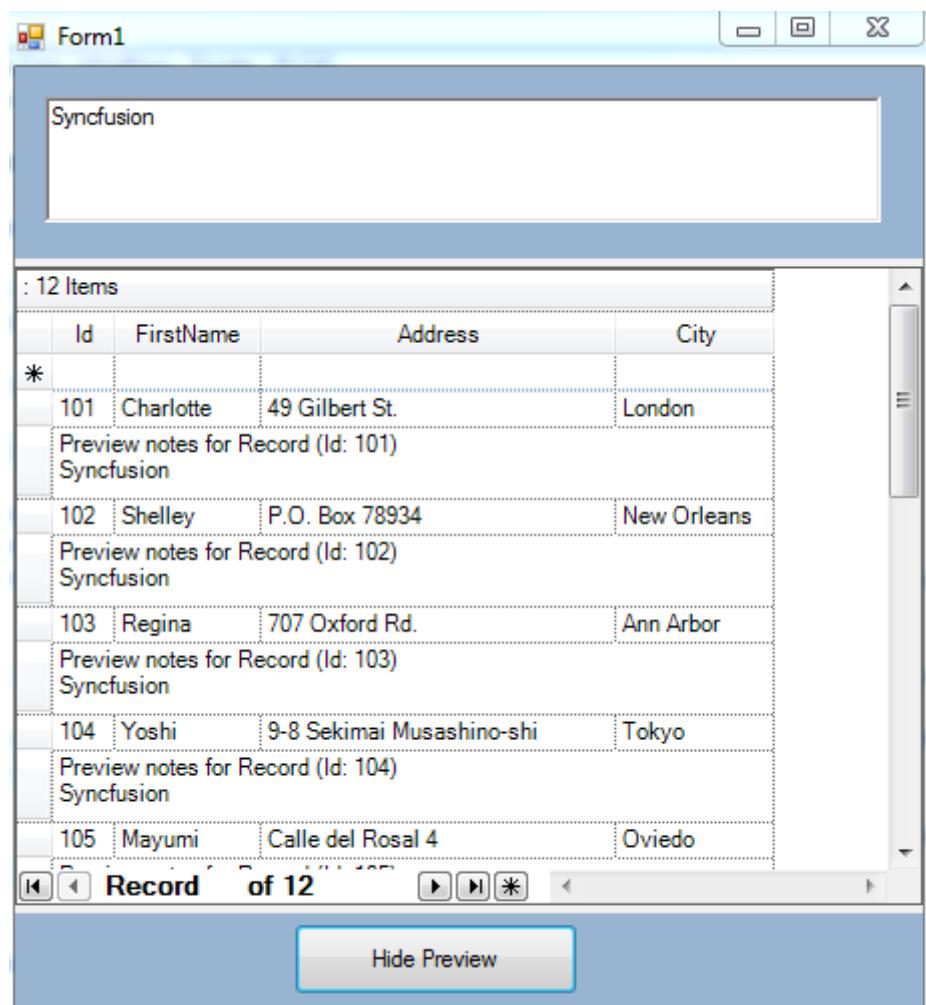


Figure 492: Journal control designed using GridGrouping controls

**Sample link:**

<http://www.syncfusion.com/uploads/redirect.aspx?&team=support&file=168039-490391191.zip>

### 5.3.1.16 How to Get New Form Window in Front of an Active Window while Double Clicking a Cell

To display a new form window while double clicking a cell, **TableControlCellDoubleClick** event is handled. The **window.BringToFront()** method is used to bring the new window in front of the active window, but this is effective only if the **FocusOnMouseDown** property is set to *false*.

[C#]

```
//Form load...
gridGroupingControll.TableControlCellDoubleClick +=
    GridGroupingControllOnTableControlCellDoubleClick;
gridGroupingControll.TableOptions.ListBoxSelectionMode = SelectionMode.One;
```

```
gridGroupingControll.TableControl.FocusOnMouseDown = false;
}
private void GridGroupingControllOnTableControlCellDoubleClick(object
sender, GridTableCellClickEventArgs e)
{
var window = new Form1();
window.Show();
window.BringToFront();
}
```

**[VB]**

```
'Form load...
AddHandler gridGroupingControll.TableControlCellDoubleClick, AddressOf
GridGroupingControllOnTableControlCellDoubleClick
gridGroupingControll.TableOptions.ListBoxSelectionMode = SelectionMode.One
gridGroupingControll.TableControl.FocusOnMouseDown = False
End Sub
Private Sub GridGroupingControllOnTableControlCellDoubleClick(ByVal sender
As Object, ByVal e As GridTableCellClickEventArgs)
Dim window = New Form1()
window.Show()
window.BringToFront()
End Sub
```

## 5.3.2 Nested Tables

This section comprises the following topics:

### 5.3.2.1 How to access child table's display elements

This can be done using the below code.

**[C#]**

```
//Loop for the number of elements in the display elements.
for(int i = 0; i <
this.gridGroupingControll.Table.DisplayElements.Count; i++)
{
    Element el = this.gridGroupingControll.Table.DisplayElements[i];
    // If the element is a nested table
    if(el.Kind == DisplayElementKind.NestedTable)
    {
        // Displaying the nested table elements.
        GridNestedTable nt = (GridNestedTable) el;
        foreach(Element ell in nt.ChildTable.NestedDisplayElements)
        {
```

```
        System.Diagnostics.Trace.WriteLine(el1);
    }
    i += (nt.ChildTable.NestedDisplayElements.Count - 1);
}
}
```

**[VB]**

```
'Loop for the number of elements in the display elements.
Dim i As Integer = 0
Do While i < Me.gridGroupingControl1.Table.DisplayElements.Count
Dim el As Element = Me.gridGroupingControl1.Table.DisplayElements(i)
' If the element is a nested table
    If el.Kind = DisplayElementKind.NestedTable Then
' Displaying the nested table elements.
Dim nt As GridNestedTable = CType(el, GridNestedTable)
    For Each el1 As Element In nt.ChildTable.NestedDisplayElements
        System.Diagnostics.Trace.WriteLine(el1)
    Next el1
    i += (nt.ChildTable.NestedDisplayElements.Count - 1)
End If
i += 1
Loop
```

### 5.3.2.2 How to Access the Nested Tables in a GridGroupingControl

You can get access to a nested table inside a record using the following code.

#### Example

**[C#]**

```
// Access a nested table.
NestedTable nt = groupingGrid.Table.Records[n].NestedTables[0];
```

**[VB.NET]**

```
' Access a nested table.
Dim nt As NestedTable = groupingGrid.Table.Records(n).NestedTables(0)
```

Whereas n is the specific record that owns the nested table.

The NestedTable has a **ChildTable** property.

**[C#]**

```
// Get the child of the nested table.  
ChildTable ct = nt.ChildTable;
```

**[VB .NET]**

```
' Get the child of the nested table.  
Dim ct As ChildTable = nt.ChildTable
```

To get access to a specific record in that child table, use the below given code snippet.

**[VB .NET]**

```
' Access the first record.  
Dim recordInNestedTable As Record = ct.Records(0)  
or  
Dim recordInNestedTable As Record = ct.FilteredRecords(0)
```

If you need access to the **TableDescriptor** for a nested relation, use the below given code snippet.

**[C#]**

```
// To access the ChildTableDescriptor.  
GridTableDescriptor td =  
this.gridGroupingControl1.TableDescriptor.Relations[0].ChildTableDescriptor;
```

**[VB .NET]**

```
' To access the ChildTableDescriptor.  
Dim td As GridTableDescriptor =  
Me.gridGroupingControl1.TableDescriptor.Relations(0).ChildTableDescriptor  
or
```

### 5.3.2.3 How to Get a Value from the Parent Record of the Nested Table when the Dropdown List is Clicked

Through the nested table's ParentRecord.GetValue() method the value of the parent record of the nested table can be retrieved when we are in the nested table. In the QueryCellStyleInfo event, the parent record's value is retrieved as an object. With the object, the data sources of the nested tables are changed. The code for the QueryCellStyleInfo event is as follows.

**[C#]**

```
private void gridGroupingControll1_QueryCellStyleInfo(object sender,
GridTableCellStyleEventArgs e)
{
    if(e.TableCellIdentity.Column != null &&
e.TableCellIdentity.Column.MappingName == "dropCol" &&
e.TableCellIdentity.DisplayElement is GridRecordRow)
    {
        object o =
e.TableCellIdentity.DisplayElement.ParentChildTable.ParentNestedTable.P
arentRecord.GetValue("check");
        this.gridGroupingControll1.TableModel.ResetVolatileData();
        if(o!=System.DBNull.Value && o != null && (bool)o)
        {
            e.Style.DataSource = colors;
            e.Style.DisplayMember = "color";
            e.Style.ValueMember = "color";
            e.Style.CellType = "GridListControl";
        }
        else
        {
            e.Style.DataSource = shapes;
            e.Style.DisplayMember = "shape";
            e.Style.ValueMember = "shape";
            e.Style.CellType = "GridListControl";
        }
    }
}
```

**[VB]**

```
Private Sub gridGroupingControll1_QueryCellStyleInfo(ByVal sender As
Object, ByVal e As GridTableCellStyleEventArgs)
    If Not e.TableCellIdentity.Column Is Nothing AndAlso
e.TableCellIdentity.Column.MappingName = "dropCol" AndAlso TypeOf
e.TableCellIdentity.DisplayElement Is GridRecordRow Then
        Dim o As Object =
e.TableCellIdentity.DisplayElement.ParentChildTable.ParentNestedTable.P
arentRecord.GetValue("check")
        Me.gridGroupingControll1.TableModel.ResetVolatileData()
        If Not o Is System.DBNull.Value AndAlso Not o Is Nothing
```

```
AndAlso CBool(o) Then
    e.Style.DataSource = colors
    e.Style.DisplayMember = "color"
    e.Style.ValueMember = "color"
    e.Style.CellType = "GridListControl"
Else
    e.Style.DataSource = shapes
    e.Style.DisplayMember = "shape"
    e.Style.ValueMember = "shape"
    e.Style.CellType = "GridListControl"
End If
End If
End Sub
```

#### 5.3.2.4 How to hide the row headers of a child table in the GridGroupingControl

You can do this by accessing the Child Table of the GridGroupingControl using the **GridTableModel**. Then handle the **QueryColWidth** event handler of the Child Table, and hide the Row Header (which is column zero) by setting the **Size** property to *Zero*.

[C#]

```
GridTableModel tbl =
this.gridGroupingControl1.GetTableModel("ChildTable");
tbl.QueryColWidth += new GridRowColSizeEventHandler(tbl_QueryColWidth);

void tbl_QueryColWidth(object sender, GridRowColSizeEventArgs e)
{
if (e.Index == 0)
{
e.Size = 0;
e.Handled = true;
}
}
```

[VB]

```
Dim tbl As GridTableModel =
Me.GridGroupingControl1.GetTableModel("ChildTable")
AddHandler tbl.QueryColWidth, AddressOf tbl_QueryColWidth
```

```
Private Sub tbl_QueryColWidth(ByVal sender As Object, ByVal e As  
GridRowColSizeEventArgs)  
If e.Index = 0 Then  
e.Size = 0  
e.Handled = True  
End If  
End Sub 'tbl_QueryColWidth
```

### 5.3.2.5 How to Set Column Style Properties for the Nested Tables in a Grouping Grid

To set the **column** properties for nested tables, you'll need to use the **TableDescriptor** property for the different tables. Here is the code that will get the **TableDescriptors** for the parent table, child table and grandchild table from the **GroupingControl's GridEngine** and which, sets some column style properties for each table.

#### Example

```
[C#]  
  
GridEngine engine = this.gridGroupingControll.Engine;  
  
// Set some parent properties.  
    GridTableDescriptor parentTD = (GridTableDescriptor)  
engine.TableDescriptor;  
  
parentTD.Columns["parentID"].Appearance.AnyRecordFieldCell.BackColor =  
Color.LightGoldenrodYellow;  
    parentTD.Columns["ParentName"].Width = 120;  
  
// Set some child properties.  
    RelationDescriptor childRD =  
parentTD.Relations[childTable.TableName];  
    GridTableDescriptor childTD = (GridTableDescriptor)  
childRD.ChildTableDescriptor;  
    childTD.Columns["Name"].Appearance.AnyRecordFieldCell.BackColor =  
Color.Pink;  
  
// Set some grandchild properties.  
    RelationDescriptor grandChildRD =  
childTD.Relations[grandChildTable.TableName];  
    GridTableDescriptor grandChildTD = (GridTableDescriptor)  
grandChildRD.ChildTableDescriptor;  
  
grandChildTD.Columns["GrandChildID"].Appearance.AnyRecordFieldCell.Back  
Color = Color.LightBlue;
```

```
grandChildTD.Columns["Name"].Appearance.AnyRecordFieldCell.BackColor =  
Color.LightCoral;  
  
grandChildTD.Columns["ChildID"].Appearance.AnyRecordFieldCell.BackColor  
= Color.LightYellow;
```

**[VB .NET]**

```
Dim engine As GridEngine = Me.GridGroupingControl1.Engine  
  
' Set some parent properties.  
Dim parentTD As GridTableDescriptor = CType(engine.TableDescriptor,  
GridTableDescriptor)  
  
parentTD.Columns("parentID").Appearance.AnyRecordFieldCell.BackColor =  
Color.LightGoldenrodYellow  
    parentTD.Columns("ParentName").Width = 120  
  
' Set some child properties.  
Dim childRD As RelationDescriptor =  
parentTD.Relations(childTable.TableName)  
  
    Dim childTD As GridTableDescriptor =  
CType(childRD.ChildTableDescriptor, GridTableDescriptor)  
  
        childTD.Columns("Name").Appearance.AnyRecordFieldCell.BackColor =  
Color.Pink '  
  
' Set some grandchild properties.  
Dim grandChildRD As RelationDescriptor =  
childTD.Relations(grandChildTable.TableName)  
  
    Dim grandChildTD As GridTableDescriptor =  
CType(grandChildRD.ChildTableDescriptor, GridTableDescriptor)  
  
grandChildTD.Columns("GrandChildID").Appearance.AnyRecordFieldCell.Back  
Color = Color.LightBlue  
  
grandChildTD.Columns("Name").Appearance.AnyRecordFieldCell.BackColor =  
Color.LightCoral '  
  
grandChildTD.Columns("ChildID").Appearance.AnyRecordFieldCell.BackColor  
= Color.LightYellow
```

### 5.3.2.6 How to set style properties of a nested table

This can be done using the below code.

[C#]

```
// 1. Changing the backcolor of all the record field cells in the child table.

// Get the child table descriptor for a particular relation
GridTableDescriptor child_tabledescriptor =
this.gridGroupingControl1.TableDescriptor.Relations["MyChildTable"].ChildTableDescriptor;
// Set the style properties
child_tabledescriptor.Columns["childID"].Appearance.AnyRecordFieldCell.
BackColor = Color.Pink;

// 2. Changing the cell property of all the record field cells in the child table.

// Get the child table descriptor for a particular relation
GridTableDescriptor child_tabledescriptor =
this.gridGroupingControl1.TableDescriptor.Relations["MyChildTable"].ChildTableDescriptor;
// Set the style properties
child_tabledescriptor.Columns["childID"].Appearance.AnyRecordFieldCell.
CellType="ComboBox";
```

[VB]

```
' 1. Changing the backcolor of all the record field cells in the child table.

' Get the child table descriptor for a particular relation
Dim child_tabledescriptor As GridTableDescriptor =
Me.gridGroupingControl1.TableDescriptor.Relations("ComSal").ChildTableDescriptor
' Set the style properties
child_tabledescriptor.Columns("Des").Appearance.AnyRecordFieldCell.Back
Color = Color.Pink

' 2. Changing the cell property of all the record field cells in the child table.
' Get the child table descriptor for a particular relation
```

```
Dim child_tabledescriptor As GridTableDescriptor =
Me.gridGroupingControl1.TableDescriptor.Relations("ComSal").ChildTableDescriptor
' Set the style properties
child_tabledescriptor.Columns("Des").Appearance.AnyRecordFieldCell.Cell
Type="ComboBox"
```

### 5.3.2.7 How does one add currency symbol for parent table and child table

You can set the currency symbol for both Parent and Child through **GridColumnDescriptor** object. Parent tables have it by default. For the Child table setting, you'll need to get the **GridColumnDescriptor** for each Child table, through recursion. The **AccessChild()** method is handled recursively, to set the currency symbol for Grandchild tables, using relations of the GridGroupingControl.

```
[C#]
//In Form_Load
//For Parent Table
this.gridGroupingControl1.TableDescriptor.Columns["Values"].Appearance.AnyRe
cordFieldCell.CellType = GridCellTypeName.Currency;
this.gridGroupingControl1.TableDescriptor.Columns["Values"].Appearance.AnyRe
cordFieldCell.CurrencyEdit.CurrencySymbol = "@";

//For Child Table
foreach (GridRelationDescriptor relation in
this.gridGroupingControl1.TableDescriptor.Relations)
{
    AccessChild(relation);
}

public void AccessChild(GridRelationDescriptor relation)
{
    relation.ChildTableDescriptor.Columns["Values"].Appearance.AnyRecordFi
eldCell.CellType = GridCellTypeName.Currency;

    relation.ChildTableDescriptor.Columns["Values"].Appearance.AnyRecordFi
eldCell.CurrencyEdit.CurrencySymbol = "!#";
    if (relation.ChildTableDescriptor.Relations.Count > 0)
```

```
{  
    foreach (GridRelationDescriptor rel in  
relation.ChildTableDescriptor.Relations)  
    {  
        AccessChild(rel); //Recursive function  
    }  
}  
}
```

**[VB]**

```
'In Form_Load  
'For Parent Table  
    Me.gridGroupingControl1.TableDescriptor.Columns("Values").Appearance.AnyRecordFieldCell.CellType = GridCellTypeName.Currency  
    Me.gridGroupingControl1.TableDescriptor.Columns("Values").Appearance.AnyRecordFieldCell.CurrencyEdit.CurrencySymbol = "@"  
  
'For Child Table  
For Each relation As GridRelationDescriptor In  
Me.gridGroupingControl1.TableDescriptor.Relations  
    AccessChild(relation)  
Next relation  
  
Public Sub AccessChild(ByVal relation As GridRelationDescriptor)  
  
    relation.ChildTableDescriptor.Columns("Values").Appearance.AnyRecordFieldCell.CellType = GridCellTypeName.Currency  
  
    relation.ChildTableDescriptor.Columns("Values").Appearance.AnyRecordFieldCell.CurrencyEdit.CurrencySymbol = "#"  
    If relation.ChildTableDescriptor.Relations.Count > 0 Then  
        For Each rel As GridRelationDescriptor In  
relation.ChildTableDescriptor.Relations  
            AccessChild(rel) 'Recursive function  
        Next rel  
    End If  
End Sub
```

### 5.3.2.8 Foreign Key Reference For Child Table

When you want to create a relation between the child and grandchild table with *ForeignKeyReference*, set the *RelationKind* property of *GridRelationDescriptor* to *ForeignKeyReference*.

The following code illustrates this:

```
[C#]

public Form1()
{
    // To Add relation to ParentTable
    GridRelationDescriptor parentToChildRelationDescriptor =
new GridRelationDescriptor();
    parentToChildRelationDescriptor.ChildTableName =
"MyChildTable";      // same as SourceListSetEntry.Name for childTable (see
below)
    parentToChildRelationDescriptor.RelationKind =
RelationKind.RelatedMasterDetails;

    parentToChildRelationDescriptor.RelationKeys.Add("parentID",
"ParentID");

parentToChildRelationDescriptor.ChildTableDescriptor.VisibleColumns.Add("Name");
parentToChildRelationDescriptor.ChildTableDescriptor.VisibleColumns.Add("MyG
randChildTable_Name");

gridGroupingControl1.TableDescriptor.Relations.Add(parentToChildRelati
onDescriptor);

    // To add Relation to GrandChildTable in Childtable like look up
    GridRelationDescriptor childToGrandChildRelationDescriptor = new
GridRelationDescriptor();
    childToGrandChildRelationDescriptor.ChildTableName =
"MyGrandChildTable"; // same as SourceListSetEntry.Name for
grandChhildTable (see below)
    childToGrandChildRelationDescriptor.RelationKind =
RelationKind.ForeignKeyReference;

    childToGrandChildRelationDescriptor.RelationKeys.Add("childID",
"GrandChildID");

childToGrandChildRelationDescriptor.ChildTableDescriptor.VisibleColumns.Add(
"GrandChildID");

childToGrandChildRelationDescriptor.ChildTableDescriptor.VisibleColumns.Add(
"Name");

//Add relation to ChildTable
```

```
parentToChildRelationDescriptor.ChildTableDescriptor.Relations.Add(childToGr  
andChildRelationDescriptor);  
}  
}
```

**[VB]**

```
Public Sub New()  
    ' To Add relation to ParentTable  
    Dim parentToChildRelationDescriptor As New  
    GridRelationDescriptor()  
        parentToChildRelationDescriptor.ChildTableName = "MyChildTable"  
        parentToChildRelationDescriptor.RelationKind =  
    RelationKind.RelatedMasterDetails  
        parentToChildRelationDescriptor.RelationKeys.Add("parentID",  
    "ParentID")  
  
    parentToChildRelationDescriptor.ChildTableDescriptor.VisibleColumns.Add("Nam  
e")  
  
    parentToChildRelationDescriptor.ChildTableDescriptor.VisibleColumns.Add("MyG  
randChildTable_Name")  
  
gridGroupingControll.TableDescriptor.Relations.Add(parentToChildRelationDesc  
riptor)  
  
    ' To add Relation to GrandChildTable in Childtable like look up  
    Dim childToGrandChildRelationDescriptor As New  
    GridRelationDescriptor()  
        childToGrandChildRelationDescriptor.ChildTableName =  
    "MyGrandChildTable"  
        childToGrandChildRelationDescriptor.RelationKind =  
    RelationKind.ForeignKeyReference  
        childToGrandChildRelationDescriptor.RelationKeys.Add("childID",  
    "GrandChildID")  
  
    childToGrandChildRelationDescriptor.ChildTableDescriptor.VisibleColumns.Add(  
    "GrandChildID")  
  
    childToGrandChildRelationDescriptor.ChildTableDescriptor.VisibleColumns.Add(  
    "Name")  
  
    'Add relation to ChildTable  
parentToChildRelationDescriptor.ChildTableDescriptor.Relations.Add(childToGr  
andChildRelationDescriptor)  
End Sub
```

### 5.3.3 Summaries

This section comprises the following topics:

#### 5.3.3.1 How to align the summary cells in the GridGroupingControl

You can align the summary cells of the GridGroupingControl by making use of the **Horizontal Alignment** property, as shown below.

[C#]

```
GridSummaryColumnDescriptor sumCol1 = new  
GridSummaryColumnDescriptor("sumCol1", SummaryType.DoubleAggregate,  
"Col1", "{Sum}");  
sumCol1.Appearance.AnyCell.HorizontalAlignment =  
GridHorizontalAlignment.Right;
```

[VB]

```
Dim sumCol1 As New GridSummaryColumnDescriptor("sumCol1",
SummaryType.DoubleAggregate, "Col1", "{Sum}")
sumCol1.Appearance.AnyCell.HorizontalAlignment =
GridHorizontalAlignment.Right
```

### 5.3.3.2 How to create weighted summaries in the GridGroupingControl

You can create custom summaries that perform weighted average calculations using a code naming convention to pass the column holding the weights to the summary descriptor. Refer the below code snippet which illustrates this.

[C#]

```
private void gridGroupingControl1_QueryCustomSummary(object sender,
GridQueryCustomSummaryEventArgs e)
{
if(e.SummaryDescriptor.SummaryType == SummaryType.Custom)
{
e.SummaryDescriptor.CreateSummaryMethod = new
CreateSummaryDelegate(WeightedSummary.CreateSummaryMethod);
}
e.Handled = true;
}

private GridSummaryColumnDescriptor
GetWeightedSummaryColumnDescriptor(string sourceCol, string weightCol)
{
GridSummaryColumnDescriptor wgtSumCol = new
GridSummaryColumnDescriptor();
wgtSumCol.Name = string.Format("{0} {1}", sourceCol, weightCol);
//special name following the convention above
wgtSumCol.DataMember = sourceCol; //the column this summary is applied
to
wgtSumCol.DisplayColumn = sourceCol; //where this summary is displayed
wgtSumCol.Format = "{WeightedAverage:#.##}"; //what is displayed in the
summary
wgtSumCol.SummaryType = SummaryType.Custom; //marks this as a Custom
Summary
wgtSumCol.Appearance.AnySummaryCell.HorizontalAlignment =
GridHorizontalAlignment.Right;
wgtSumCol.MaxLength = 6;
return wgtSumCol;
```

```
}
```

**[VB]**

```
'custom summary
Private Sub gridGroupingControll1_QueryCustomSummary(ByVal sender As Object, ByVal e As GridQueryCustomSummaryEventArgs)
If e.SummaryDescriptor.SummaryType = SummaryType.Custom Then
e.SummaryDescriptor.CreateSummaryMethod = New
CreateSummaryDelegate(AddressOf WeightedSummary.CreateSummaryMethod)
End If
e.Handled = True
End Sub

Private Function GetWeightedSummaryColumnDescriptor(ByVal sourceCol As String, ByVal weightCol As String) As GridSummaryColumnDescriptor
Dim wgtSumCol As GridSummaryColumnDescriptor = New
GridSummaryColumnDescriptor
wgtSumCol.Name = String.Format("{0}_{1}", sourceCol, weightCol)
'special name following the convention above
wgtSumCol.DataMember = sourceCol 'the column this summary is applied to
wgtSumCol.DisplayColumn = sourceCol 'where this summary is displayed
wgtSumCol.Format = "{WeightedAverage:#.##}" 'what is displayed in the summary
wgtSumCol.SummaryType = SummaryType.Custom 'marks this as a Custom Summary
wgtSumCol.Appearance.AnySummaryCell.HorizontalAlignment =
GridHorizontalAlignment.Right
wgtSumCol.MaxLength = 6
Return wgtSumCol
End Function
```

### 5.3.3.3 How to define a summary row in the grid

This can be done using the below code.

**[C#]**

```
// Defining a summary column descriptor.
GridSummaryColumnDescriptor sd = new GridSummaryColumnDescriptor();
// Summary for Col2
sd.DataMember= "Col2";
// Setting under which column you need to see the total.
```

```
sd.DisplayColumn = "Col2";
// Here you specify the format of the field to be displayed.
sd.Format = "{Count}";
// Here you set the type of the summary i.e totalling or average or
// count etc...
sd.SummaryType = SummaryType.DistinctCount;
// Here "Total" is the text that occurs as the header.
// Create a new SummaryRowDescriptor and add it to the SummaryRows
collection
this.gridGroupingControll1.TableDescriptor.SummaryRows.Add(new
GridSummaryRowDescriptor("Col2", "Total", sd));
```

**[VB]**

```
' Defining a summary column descriptor.
Private sd As GridSummaryColumnDescriptor = New
GridSummaryColumnDescriptor()
' Summary for Col2
sd.DataMember= "Col2"
' Setting under which column you need to see the total.
sd.DisplayColumn = "Col2"
' Here you specify the format of the field to be displayed.
sd.Format = "{Count}"
' Here you set the type of the summary i.e totalling or average or
// count etc...
sd.SummaryType = SummaryType.DistinctCount
' Here "Total" is the text that occurs as the header.
' Create a new SummaryRowDescriptor and add it to the SummaryRows
collection
Me.gridGroupingControll1.TableDescriptor.SummaryRows.Add(New
GridSummaryRowDescriptor("Col2", "Total", sd))
```

#### 5.3.3.4 How to format summary rows

The following code shows how to format summary rows.

**[C#]**

```
// sd is GridSummaryColumnDescriptor
// Changing the backcolor of the SummaryFieldCell.
sd.Appearance.SummaryFieldCell.BackColor = Color.LightBlue;
// Changing the type of the SummaryFieldCell.
sd.Appearance.SummaryFieldCell.CellType = "ComboBox";
```

**[VB]**

```
' sd is GridSummaryColumnDescriptor
' Changing the backcolor of the SummaryFieldCell.
sd.Appearance.SummaryFieldCell.BackColor = Color.LightBlue
' Changing the type of the SummaryFieldCell.
sd.Appearance.SummaryFieldCell.CellType = "ComboBox"
```

### 5.3.3.5 How to retrieve a summary item

The following code shows how to retrieve a summary item.

**[C#]**

```
// sd is GridSummaryColumnDescriptor
string
item=GridEngine.GetSummaryText(this.gridGroupingControll1.Table.TopLevel
Group, sd)
```

**[VB]**

```
'sd is GridSummaryColumnDescriptor
Dim item As String =
GridEngine.GetSummaryText(Me.gridGroupingControll1.Table.TopLevelGroup,
sd)
```

### 5.3.3.6 How to show Summary Results in the Inner Most Table on every Parent Level in Nested Related Tables

The summary results of the nested tables can be obtained by using the **GetSummaryText** method of the **GridEngine**.

#### Example

**[C#]**

```
private void gridGroupingControll1_QueryValue(object sender,
Syncfusion.Grouping.FieldValueEventArgs e)
```

```

{
    if(e.TableDescriptor.Name == this.parentTD.Name)
    {
        Group g = e.Record.NestedTables[0].ChildTable as Group;

        // Set SummaryText for ParentTable.
        e.Value = GridEngine.GetSummaryText(g, "c_ROW", "c_SUM");
    }
    else if
        (e.TableDescriptor == this.childTD)
    {
        Group g = e.Record.NestedTables[0].ChildTable as Group;

        // Set SummaryText for child table.
        e.Value = GridEngine.GetSummaryText(g, "gc_ROW",
"gc_SUM");
    }
    else if
        (e.TableDescriptor == this.gridChildTD)
    {
        // Retrieve the unbound column values.
        object o =
this.unboundValues[e.Record.GetValue("GrandChildID").ToString()];
        double d;
        if(o != null && double.TryParse(o.ToString(),
System.Globalization.NumberStyles.Any, null, out d))
        {
            // Display the values.
            e.Value = d;
        }
    }
}

```

**[VB.NET]**

```

Private Sub gridGroupingControl1_QueryValue(ByVal sender As Object,
ByVal e As Syncfusion.Grouping.FieldValueEventArgs)
    If e.TableDescriptor.Name = Me.parentTD.Name Then
        Dim g As Group = CType(IIf(TypeOf
e.Record.NestedTables(0).ChildTable Is Group,
e.Record.NestedTables(0).ChildTable, Nothing), Group)

        ' Set SummaryText for ParentTable.
        e.Value = GridEngine.GetSummaryText(g, "c_ROW",
"c_SUM")
    ElseIf e.TableDescriptor Is Me.childTD Then
        Dim g As Group = CType(IIf(TypeOf

```

```
e.Record.NestedTables(0).ChildTable Is Group,
e.Record.NestedTables(0).ChildTable, Nothing), Group)

        ' Set SummaryText for ChildTable.
        e.Value = GridEngine.GetSummaryText(g,
"gc_ROW", "gc_SUM")

        ElseIf e.TableDescriptor Is Me.gridChildTD Then

            ' Retrieve the unbound column values.
            Dim o As Object =
Me.unboundValues(e.Record.GetValue("GrandChildID").ToString())
            Dim d As Double
            If Not o Is Nothing AndAlso
Double.TryParse(o.ToString(), System.Globalization.NumberStyles.Any,
Nothing, d) Then

                ' Display the values.
                e.Value = d
            End If
        End If
    End Sub
```

### 5.3.3.7 How to Update Summaries Immediately Upon Changing a Field

By default, the summaries will be refreshed as you leave the row. But if you want the summaries to be updated as you leave the cell then, you must handle the CurrentRecordContextChange event and you can Invalidate the summary. Refer the following code snippet.

**[C#]**

```
private void gridGroupingControl1_CurrentRecordContextChange(object
sender, CurrentRecordContextChangeEvent e)
{
    if(e.Action == CurrentRecordAction.CurrentFieldChanged)
    {

this.gridGroupingControl1.CurrencyManager.EndCurrentEdit();
        this.gridGroupingControl1.Table.InvalidateSummary();
    }
}
```

**[VB]**

```
Private Sub gridGroupingControll_CurrentRecordContextChange(ByVal sender As Object, ByVal e As CurrentRecordContextChangeEventArgs)
    If e.Action = CurrentRecordAction.CurrentFieldChanged Then
        Me.gridGroupingControll.CurrencyManager.EndCurrentEdit()
        Me.gridGroupingControll.Table.InvalidateSummary()
    End If
End Sub
```

### 5.3.3.8 What are the options in the summary columns?

The options in the summary columns are illustrated using the below code.

[C#]

```
// Disabling the change of summary value during the filter criteria.
// sd is GridSummaryColumnDescriptor
// This ignores filtering of the grid. So, the summary value doesn't
// change.
sd.IgnoreRecordFilterCriteria=true;
```

[VB]

```
'Disabling the change of summary value during the filter criteria.
' sd is GridSummaryColumnDescriptor
' This ignores filtering of the grid. So, the summary value doesn't
// change.
sd.IgnoreRecordFilterCriteria=True
```

### 5.3.3.9 How to Get Summary Cell Values in sorting?

Or

#### How to Incorporate Summary Cells while Sorting?

Or

#### How to Sort the Grid based on the Summary Cell Values?

While sorting, the grid rearranges the data to match the current sort criteria, but it excludes the summary cells. To perform sorting based on the summary cell values, it should be passed as an argument in the “**SetGroupSummarySortOrder**” method.

[C#]

```
private void PeformSort(GridTableControl tableControl, string name)
{
    if (tableControl.TableDescriptor.IsGrouped)
    {
        //Code...
        if (gsc != null)
        {
            string caption = gsc.Format.Trim(new char[] {'{', '}'});
            foreach (SortColumnDescriptor col in
tableControl.TableDescriptor.GroupedColumns)
            {
                if (!flag)
                    group.Add(col.Name, col.Clone());
                col.SetGroupSummarySortOrder(gsc.GetSummaryDescriptorName(), caption,
                _summSort);
            }
            flag = true;
        }
        //Code...
    }
}
```

[VB]

```
Private Sub PeformSort(ByVal tableControl As GridTableControl, ByVal
name As String)
If tableControl.TableDescriptor.IsGrouped Then

    'Code...

    If gsc IsNot Nothing Then
        Dim caption As String = gsc.Format.Trim(New Char() {"{", "}"})
        For Each col As SortColumnDescriptor In
tableControl.TableDescriptor.GroupedColumns
            If Not flag Then
                group.Add(col.Name, col.Clone())
            End If
            col.SetGroupSummarySortOrder(gsc.GetSummaryDescriptorName(), caption,
            _summSort)
        Next col
        flag = True
    'Code...
End If
```

### 5.3.3.10 How to Change the Format of the Summary Cell in Group Caption

To change the summary cell's format, the **TableControlDrawCellDisplayText** event is handled. Using the **DisplayText** property the desired format can be applied to the summary cell.

**[C#]**

```
//In form load...

this.gridGroupingControl1.TableControlDrawCellDisplayText += new
GridTableControlDrawCellDisplayTextEventHandler(gridGroupingControl1_Ta
bleControlDrawCellDisplayText);
}
void gridGroupingControl1_TableControlDrawCellDisplayText(object
sender, GridTableControlDrawCellDisplayTextEventArgs e)
{
GridTableCellStyleInfo style = e.Inner.Style as GridTableCellStyleInfo;
if (style.TableCellIdentity.TableCellType ==
GridTableCellType.GroupCaptionSummaryCell)
{
    double value;
    if (double.TryParse(e.Inner.DisplayText, out value))
        e.Inner.DisplayText = value.ToString("#,###.00");
}
}

//Code...
```

**[VB]**

```
'In form load...
AddHandler gridGroupingControl1.TableControlDrawCellDisplayText,
AddressOf gridGroupingControl1_TableControlDrawCellDisplayText
End Sub

Private Sub gridGroupingControl1_TableControlDrawCellDisplayText(ByVal
sender As Object, ByVal e As
GridTableControlDrawCellDisplayTextEventArgs)

Dim style As GridTableCellStyleInfo = TryCast(e.Inner.Style,
GridTableCellStyleInfo)
If style.TableCellIdentity.TableCellType =
GridTableCellType.GroupCaptionSummaryCell Then
    value As Double
    If Double.TryParse(e.Inner.DisplayText, value) Then
        e.Inner.DisplayText = value.ToString("#,###.00")
    End If
End If
'Code
```

Refer to the following sample file for more details:

<http://files2.syncfusion.com/dtSupport/DirectTrac/98946/GGCSortingFormat345822923.zip>

### 5.3.4 DisplayElements

This section comprises the following topics:

#### 5.3.4.1 How to access all the DisplayElements or a particular DisplayElement in the GridGrouping control

You can access the DisplayElements by using the following code.

[C#]

```
// Accessing all the display elements
foreach (Element el in gridGroupingControl1.Table.DisplayElements)
{
    Console.WriteLine(el.Info);
}

// Accessing a particular display element
Console.WriteLine(this.gridGroupingControl1.Table.DisplayElements[index].Info);
```

[VB]

```
'Accessing all the display element
For Each el As Element In
Me.gridGroupingControl1.Table.DisplayElements
    Console.WriteLine(el.Info)
Next el

'Accessing a particular display element
Console.WriteLine(Me.gridGroupingControl1.Table.DisplayElements(index).Info)
```

#### 5.3.4.2 How to access a particular DisplayElement if RowIndex is provided

You can access the DisplayElements with the rowIndex, by using the following code.

[C#]

```
// Accessing a particular display element
Element
el=this.gridGroupingControl1.Table.DisplayElements[rowindex].ParentElement;
```

[VB]

```
' Accessing a particular display element
Dim el As Element =
Me.gridGroupingControl1.Table.DisplayElements(rowIndex).ParentElement
```

### 5.3.4.3 How to find a DisplayElement Type

You can find the type of a particular DisplayElement using the below code.

[C#]

```
// Accessing the type of display element
Console.WriteLine(this.gridGroupingControl1.Table.DisplayElements[rowindex].Kind);
```

[VB]

```
' Accessing the type of display element
Console.WriteLine(Me.gridGroupingControl1.Table.DisplayElements(rowIndex).Kind)
```

## 5.3.5 Layout and Appearance

This section comprises the following topics:

### 5.3.5.1 How to Allocate Equal Size for Each of the Columns in all the Tables

The parent/child table columns width can be set equally in proportion to the grid control's client width, by dynamically setting the columns width in the TableModel.QueryColWidth event handler. When it deals with a nested table, the QueryColWidth event of the entire nested table must be handled to set the respective nested table columns width.

In the QueryColWidth event handler, the available width for the columns can be calculated as follows,

```
availableArea = groupingGrid.ClientSize.Width - gridModel.ColWidths.GetTotal(0, gridModel.Cols.HeaderCount) - indentColsTotalWidth;
```

and the proportional columns width can be calculated as follows,

```
Size = (int) availableArea / (grid.TableDescriptor.VisibleColumns.Count);
```

### 5.3.5.2 How to create multiple rows per record in a GridGroupingControl

This can be done by using the **ColumnSets** property of the GridGrouping control.

[C#]

```
//Create a GridColumnSpanDescriptor and initialize to a column and
mention its location to display
GridColumnSpanDescriptor columnSpanDescriptor1 = new
GridColumnSpanDescriptor("Name", "R0C0");

//Create a GridColumnSetDescriptor
GridColumnSetDescriptor columnSetDescriptor1 = new
GridColumnSetDescriptor();

//Add the GridColumnSpanDescriptor to the GridColumnSetDescriptor
columnSetDescriptor1.ColumnSpans.Add(columnSpanDescriptor1);

//Add the GridColumnSetDescriptor to the ColumnSet of the
GridGroupingControl through the TableDescriptor.
this.gridGroupingControl1.TableDescriptor.ColumnSets.Add(columnSetDescriptor1);
```

[VB]

```
'Create a GridColumnSpanDescriptor and initialize to a column and  
mention its location to display  
Dim columnSpanDescriptor1 As GridColumnSpanDescriptor = New  
GridColumnSpanDescriptor("Name", "R0C0")  
  
'Create a GridColumnSetDescriptor  
Dim columnSetDescriptor1 As GridColumnSetDescriptor = New  
GridColumnSetDescriptor()  
  
'Add the GridColumnSpanDescriptor to the GridColumnSetDescriptor  
columnSetDescriptor1.ColumnSpans.Add(columnSpanDescriptor1)  
  
'Add the GridColumnSetDescriptor to the ColumnSet of the  
GridGroupingControl through the TableDescriptor.  
Me.gridGroupingControll1.TableDescriptor.ColumnSets.Add(columnSetDescriptor1)
```

### 5.3.5.3 How to freeze specified columns

You can freeze Specified columns by making use of the below given code.

[C#]

```
//Add a specified column index to freeze  
this.gridGroupingControll1.TableModel.Cols.FreezeRange(1, 1);  
  
//Add a range of columns to freeze.  
this.gridGroupingControll1.TableModel.Cols.FreezeRange(1, 3);
```

[VB]

```
'Add a specified column index to freeze  
Me.gridGroupingControll1.TableModel.Cols.FreezeRange(1, 1)  
  
'Add a range of columns to freeze.  
Me.gridGroupingControll1.TableModel.Cols.FreezeRange(1, 3)
```

#### 5.3.5.4 How to set conditional formatting in the GroupingGrid

To set up a conditional formatting in the GroupingGrid, use the following code.

[C#]

```
// Declaring a conditional format descriptor.  
GridConditionalFormatDescriptor gcfd = new  
GridConditionalFormatDescriptor();  
  
// Setting some properties  
gcfd.Appearance.AnyRecordFieldCell.Font.Bold = true;  
gcfd.Appearance.AnyRecordFieldCell.Interior = new  
BrushInfo(Color.LightGreen);  
  
//Expression which is applied on the table data.  
gcfd.Expression = "[ColumnName] like \\'true\\\'";  
  
// Setting name to the conditional format and adding it to the grid.  
gcfd.Name = "gcfd";  
this.gridGroupingControll1.TableDescriptor.ConditionalFormats.Add(gcfd);
```

[VB]

```
' Declaring a conditional format descriptor.  
Dim gcfd As GridConditionalFormatDescriptor = New  
GridConditionalFormatDescriptor()  
  
' Setting some properties  
gcfd.Appearance.AnyRecordFieldCell.Font.Bold = True  
gcfd.Appearance.AnyRecordFieldCell.Interior = New  
BrushInfo(Color.LightGreen)  
  
' Expression which is applied on the table data.  
gcfd.Expression = "[ColumnName] like 'true'"  
  
' Setting name to the conditional format and adding it to the grid.  
gcfd.Name = "gcfd"  
Me.gridGroupingControll1.TableDescriptor.ConditionalFormats.Add(gcfd)
```

#### 5.3.5.5 How to set texts in the preview record

This can be done by handling the QueryCellStyleInfo event.

[C#]

```
//Check for the preview record
if( e.TableCellIdentity.TableCellType ==
GridTableCellType.RecordPreviewCell)
{
//Set the text in the record.
e.Style.CellValue = " This is preview record".ToString();

//Change the default italic font to regular.
e.Style.Font.Italic = false;

//Set Bold to the text font.
e.Style.Font.Bold = true;

//Change the text color.
e.Style.TextColor = Color.Purple;
}
```

[VB]

```
'Check for the preview record
If e.TableCellIdentity.TableCellType =
GridTableCellType.RecordPreviewCell Then

'Set the text in the record.
    e.Style.CellValue = " This is preview record".ToString()

'Change the default italic font to regular.
    e.Style.Font.Italic = False

'Set Bold to the text font.
    e.Style.Font.Bold = True

'Change the text color.
    e.Style.TextColor = Color.Purple
End If
```

### 5.3.5.6 How to set the cursor to GridGroupingControl instead of the default one?

You need to handle the TableControlCellCursor event of GridGroupingControl and set the required cursor to the e.Inner.Cursor property.

[C#]

```
//Event invoker
    this.gridGroupingControl1.TableControlCellCursor +=new
Syncfusion.Windows.Forms.Grid.Grouping.GridTableControlCellCursorEventHandler(gridGroupingControl1_TableControlCellCursor);

//Event handler
void gridGroupingControl1_TableControlCellCursor(object sender,
Syncfusion.Windows.Forms.Grid.Grouping.GridTableControlCellCursorEventArgs e)
{
    e.Inner.Cancel = true;
    e.Inner.Cursor = Cursors.Hand;
}
```

[VB]

```
'Event invoker
AddHandler gridGroupingControl1.TableControlCellCursor, AddressOf
gridGroupingControl1_TableControlCellCursor

'Event handler
Private Sub gridGroupingControl1_TableControlCellCursor(ByVal
sender As Object, ByVal e As
Syncfusion.Windows.Forms.Grid.Grouping.GridTableControlCellCursorEventArgs)
    e.Inner.Cancel = True
    e.Inner.Cursor = Cursors.Hand
End Sub
```

### 5.3.5.7 How to apply text color for a RichText celltype in GridGroupingControl

You can use the `QueryCellStyleInfo` event to provide styles for the cells in the `GridGroupingControl`. This provides the `GridStyleInfo` object for a cell on demand. The `QueryCellStyleInfo` is triggered every time a request is made to access the style information for a cell. You can do any type of cell formatting with this event.

You can apply style settings for a given `CellType` using the `TableCellIdentity.TableCellType` property on the instances of the `GridTableCellStyleInfoEventArgs`.

The following code example illustrates how to apply styles to RichText `CellType` in the grouping grid:

```
[C#]  
  
// Hook up the event.  
  
this.gridGroupingControl1.QueryCellStyleInfo  
+= newGridTableCellStyleInfoEventHandler(gridGroupingControl1_QueryCe  
llStyleInfo);  
  
  
//QueryCellStyleInfo event: To Format Cell by Cell Basis on demand.  
  
private void gridGroupingControl1_QueryCellStyleInfo(object sender,  
Syncfusion.Windows.Forms.Grid.Grouping.GridTableCellStyleInfoEventArgs e)  
{  
    e.Style.ReadOnly = true;  
    if (e.TableCellIdentity.Column != null &&  
e.TableCellIdentity.Column.Name == "Values" &&  
e.TableCellIdentity.DisplayElement.Kind ==  
Syncfusion.Grouping.DisplayElementKind.Record)  
    {  
        e.Style.CellType = GridCellTypeName.RichText;  
        e.Style.TextColor = Color.Purple;  
    }  
}
```

```
[VB.NET]  
  
'Hook up the event.  
  
AddHandler GridGroupingControl1.QueryCellStyleInfo, AddressOf  
gridGroupingControl1_QueryCellStyleInfo  
  
'QueryCellStyleInfo event: To Format Cell by Cell Basis on demand.
```

```
Private Sub gridGroupingControl1_QueryCellStyleInfo(ByVal sender
As Object, ByVal e As
Syncfusion.Windows.Forms.Grid.Grouping.GridTableCellStyleInfoEventArgs)
    e.Style.ReadOnly = True
    If e.TableCellIdentity.Column IsNot Nothing AndAlso
e.TableCellIdentity.Column.Name = "Values" AndAlso
e.TableCellIdentity.DisplayElement.Kind =
Syncfusion.Grouping.DisplayElementKind.Record Then
        e.Style.CellType = GridCellTypeName.RichText
        e.Style.TextColor = Color.Purple
    End If
End Sub
```

### 5.3.6 Context Menu

This section comprises the following topics:

#### 5.3.6.1 How to attach a context menu to a cell in the GridGrouping control

By handling the TableControlCurrentCellControlGotFocus, and assigning the context menu to the cell's control, we can attach the context menu to a cell, when the cell is in the edit mode.

```
[C#]
// Subscribing to the events
this.gridGroupingControl1.TableControlCurrentCellControlGotFocus +=new
Syncfusion.Windows.Forms.Grid.Grouping.GridTableControlControlEventHandler(gridGroupingControl1_TableControlCurrentCellControlGotFocus);

this.gridGroupingControl1.TableControlCurrentCellControlLostFocus +=new
Syncfusion.Windows.Forms.Grid.Grouping.GridTableControlControlEventHandler(gridGroupingControl1_TableControlCurrentCellControlLostFocus);

// Attach a context menu when the cell is in edit mode.
private void
gridGroupingControl1_TableControlCurrentCellControlGotFocus(object
sender,
```

```
Syncfusion.Windows.Forms.Grid.Grouping.GridTableControlEventArgs  
e)  
{  
e.Inner.Control.ContextMenu = this.contextMenu1;  
}  
  
// Reset the context menu when the cell losses focus  
private void  
gridGroupingControl1_TableControlCurrentCellControlLostFocus(object  
sender,  
Syncfusion.Windows.Forms.Grid.Grouping.GridTableControlEventArgs  
e)  
{  
e.Inner.Control.ContextMenu = null;  
}
```

**[VB]**

```
' Attach a context menu when the cell is in edit mode  
Private Sub  
gridGroupingControl1_TableControlCurrentCellControlGotFocus(ByVal  
sender As Object, ByVal e As  
Syncfusion.Windows.Forms.Grid.Grouping.GridTableControlEventArgs  
) Handles gridGroupingControl1.TableControlCurrentCellControlGotFocus  
    If e.TableControl.CurrentCell.RowIndex = 7 AndAlso  
e.TableControl.CurrentCell.ColumnIndex = 2 Then  
        e.Inner.Control.ContextMenu = Me.contextMenu1  
    End If  
End Sub  
  
' Reset the context menu when the cell losses focus  
Private Sub  
gridGroupingControl1_TableControlCurrentCellControlLostFocus(ByVal  
sender As Object, ByVal e As  
Syncfusion.Windows.Forms.Grid.Grouping.GridTableControlEventArgs  
) Handles gridGroupingControl1.TableControlCurrentCellControlLostFocus  
    e.Inner.Control.ContextMenu = Nothing  
End Sub
```

### 5.3.6.2 How to attach a context menu to the GridGrouping control

This can be done using the below code snippet.

**[C#]**

```
//Add items to context menu
this.contextMenu1.MenuItems.Add("One");
this.contextMenu1.MenuItems.Add("Two");
this.contextMenu1.MenuItems.Add("Three");

//Assign it to the GridGroupingControl
this.gridGroupingControll.ContextMenu = this.contextMenu1;
```

**[VB]**

```
'Add items to context menu
Me.contextMenu1.MenuItems.Add("One")
Me.contextMenu1.MenuItems.Add("Two")
Me.contextMenu1.MenuItems.Add("Three")

'Assign it to the GridGroupingControl
Me.gridGroupingControll.ContextMenu = Me.contextMenu1
```

## 5.3.7 Expressions

This section comprises the following topics:

### 5.3.7.1 How to add Expression columns

Expression fields allows you to add a column that holds calculated values based on other fields in the same record. These expression columns can be used in grouping and sorting. This also can be employed as summary fields for summary rows.

**[C#]**

```
using Syncfusion.Grouping;
using Syncfusion.Windows.Forms.Grid;
using Syncfusion.Windows.Forms.Grid.Grouping;

// Declare an ExpressionFieldDescriptor
ExpressionFieldDescriptor expression1 = new
```

```
ExpressionFieldDescriptor();
expression1.Name = "Sum of all columns";

// Simple expression to add the values in the columns
// For all the valid Expression syntax, refer the
// EssentialGrid UserGuide --> Essential Grid Tutorials --->
// Adding Expression Fields ---> Valid Expression Syntax
expression1.Expression = "[Col0] + [Col1] + [Col2] + [Col3]";

//Add the Expression column to the grid
this.gridGroupingControl1.TableDescriptor.ExpressionFields.Add(expression1);
```

**[VB]**

```
Imports Syncfusion.Grouping
Imports Syncfusion.Windows.Forms.Grid
Imports Syncfusion.Windows.Forms.Grid.Grouping

' Declare an ExpressionFieldDescriptor
Dim expression1 As New ExpressionFieldDescriptor()
expression1.Name = "Sum of all columns"

' Simple expression to add the values in the columns
' For all the valid Expression syntax, refer the
' EssentialGrid UserGuide --> Essential Grid Tutorials --->
' Adding Expression Fields ---> Valid Expression Syntax
expression1.Expression = "[Col0] + [Col1] + [Col2] + [Col3]"

'Add the Expression column to the grid
Me.gridGroupingControl1.TableDescriptor.ExpressionFields.Add(expression1)
```

### 5.3.7.2 What are the various ExpressionColumn options?

The following code illustrates various expression column options.

**[C#]**

```
//1. Setting the operators in the expression.
// The computations are performed with level one operations done first.
```

```
// Alpha constants used with match and like should be enclosed in apostrophes (').
// *, / ,+, - ,<, >, =, <=, >=,match, like, in, between ,or, and
// * operator used here.
ExpressionFieldDescriptor exp= new
ExpressionFieldDescriptor("Expr1","[ColumnName] *2", "System.Int32");

//2. Setting the output/view type.
// Output in double type.
ExpressionFieldDescriptor exp= new
ExpressionFieldDescriptor("Expr1","[ColumnName] *2", "System.Double");
```

**[VB]**

```
' 1. Setting the operators in the expression.
' Setting operators in the expression.
' The computations are performed with level one operations done first.
' Alpha constants used with match and like should be enclosed in apostrophes (').
' *, / ,+, - ,<, >, =, <=, >=,match, like, in, between ,or, and
' * operator used here.
Dim exp As ExpressionFieldDescriptor = New
ExpressionFieldDescriptor("Expr1", "[Id] *2", "System.Int32")

' 2. Setting the output in double type.
Dim exp2 As ExpressionFieldDescriptor = New
ExpressionFieldDescriptor("Expr2", "[Expr1] *2", "System.Double")
```

## 5.3.8 Selections

This section comprises the following topics:

### 5.3.8.1 How to access selected records

The selected records can be accessed using the below code snippet.

**[C#]**

```
//Accessing Selected records in a GridGroupingcontrol.  
foreach(SelectedRecord rec in  
this.gridGroupingControll1.Table.SelectedRecords)  
System.Diagnostics.Trace.WriteLine(rec.Record.Info);
```

[VB]

```
'Accessing Selected records in a GridGroupingControl.  
For Each rec As SelectedRecord In  
Me.gridGroupingControll1.Table.SelectedRecords  
Console.WriteLine(rec.Record.Info)  
Next rec
```

### 5.3.8.2 How to select a record programmatically

The following code illustrates how to select a record programmatically.

[C#]

```
// select the 3 rd record  
this.gridGroupingControll1.Table.SelectedRecords.Add(this.gridGroupingCo  
ntroll1.Table.Records[3]);
```

[VB]

```
'select the 3 rd record  
Me.gridGroupingControll1.Table.SelectedRecords.Add(Me.gridGroupingContro  
ll1.Table.Records(3))
```

### 5.3.8.3 How to set ListBoxSelectionModes

To set the **ListBoxSelectionMode** property that determines the selection behavior, use the following code.

[C#]

```
//Selecting Single record  
this.gridGroupingControll1.TableOptions.ListBoxSelectionMode =
```

```
SelectionMode.One;

//Selecting MultiRecords
this.gridGroupingControl1.TableOptions.ListBoxSelectionMode =
SelectionMode.MultiSimple;

//Selecting MultiExtendedRecords
this.gridGroupingControl1.TableOptions.ListBoxSelectionMode =
SelectionMode.MultiExtended;
```

**[VB]**

```
'Selecting Single record
Me.gridGroupingControl1.TableOptions.ListBoxSelectionMode =
SelectionMode.One

'Selecting MultiRecords
Me.gridGroupingControl1.TableOptions.ListBoxSelectionMode =
SelectionMode.MultiSimple

'Selecting MultiExtendedRecords
Me.gridGroupingControl1.TableOptions.ListBoxSelectionMode =
SelectionMode.MultiExtended
```

#### 5.3.8.4 What are the events fired when the records are selected?

Following are the events fired when the records are selected.

**[C#]**

```
//SelectionChanging Event.
this.gridGroupingControl1.SelectedRecordsChanging += new
Syncfusion.Grouping.SelectedRecordsChangedEventHandler(this.gridGroupin
gControl1_SelectedRecordsChanging);

//SelectionChanged Event.
this.gridGroupingControl1.SelectedRecordsChanged += new
Syncfusion.Grouping.SelectedRecordsChangedEventHandler(this.gridGroupin
gControl1_SelectedRecordsChanged);
```

**[VB]**

```
'SelectionChanging Event
```

```
Private Sub gridGroupingControll_SelectedRecordsChanging(ByVal sender As Object, ByVal e As Syncfusion.Grouping.SelectedRecordsChangedEventArgs) Handles gridGroupingControll.SelectedRecordsChanging
    System.Diagnostics.Trace.WriteLine("Action in SelectedRecordsChanging-->" + e.Action)
End Sub

'SelectionChanged Event
Private Sub gridGroupingControll_SelectedRecordsChanged(ByVal sender As Object, ByVal e As Syncfusion.Grouping.SelectedRecordsChangedEventArgs) Handles gridGroupingControll.SelectedRecordsChanged
    System.Diagnostics.Trace.WriteLine("Action in SelectedRecordsChanged-->" + e.Action)
End Sub
```

### 5.3.9 Records

This section comprises the following topics:

#### 5.3.9.1 How to access a particular record from a table

This can be done using the following code snippet.

[C#]

```
// Using the record Index to access a particular record from a table.
Record r=this.gridGroupingControll.Table.Records[RecordIndex];
```

[VB]

```
'Using the record Index to access a particular record from a table.
Dim r As Record = Me.gridGroupingControll.Table.Records(RecordIndex)
```

### 5.3.9.2 How to access a record given the row index

This can be done using the following code snippet.

[C#]

```
// Using the DisplayElements property of the grid, we can find the
// corresponding record.
Record r =
gridGroupingControl1.Table.DisplayElements[rowindex].ParentRecord;
```

[VB]

```
' Using the DisplayElements property of the grid, we can find the
// corresponding record.
Dim r As Record =
gridGroupingControl1.Table.DisplayElements(rowIndex).ParentRecord
```

### 5.3.9.3 How to access sorted or filtered records

This can be done using the following code snippet.

[C#]

```
// looping through the filtered records.
foreach(Record fr in this.gridGroupingControl1.Table.FilteredRecords)
{
    Console.WriteLine(fr.Info);
}
```

[VB]

```
' looping through the filtered records.
For Each fr As Record In Me.gridGroupingControl1.Table.FilteredRecords
    Console.WriteLine(fr.Info)
Next fr
```

#### 5.3.9.4 How to access unfiltered records

This can be done using the following code snippet.

[C#]

```
foreach(Record r in this.gridGroupingControl1.Table.Records)
{
    foreach(Record fr in
this.gridGroupingControl1.Table.FilteredRecords)
    {
        if(r!=fr)
        {
            Console.WriteLine(r.Info);
        }
    }
}
```

[VB]

```
For Each r As Record In Me.gridGroupingControl1.Table.Records
    For Each fr As Record In
Me.gridGroupingControl1.Table.FilteredRecords
        If Not r Is fr Then
            Console.WriteLine(r.Info)
        End If
    Next fr
Next r
```

#### 5.3.9.5 How to get a record index given the rowindex

This can be done using the following code snippet.

[C#]

```
// Record index is being calculated.
Table table = e.TableCellIdentity.Table;
Element el = table.DisplayElements[RowIndex];
Record r = el.ParentRecord;
int RecordIndex= table.UnsortedRecords.IndexOf(r);
```

**[VB]**

```
' Record index is being calculated.  
Dim table As Table = e.TableCellIdentity.Table  
Dim el As Element = table.DisplayElements(RowIndex)  
Dim r As Record = el.ParentRecord  
Dim RecordIndex As Integer = table.UnsortedRecords.IndexOf(r)
```

### 5.3.9.6 How to get a rowindex given the record index

This can be done using the following code snippet.

**[C#]**

```
int position =  
gridGroupingControl1.Table.DisplayElements.IndexOf(record);
```

**[VB]**

```
Dim position As Integer =  
gridGroupingControl1.Table.DisplayElements.IndexOf(record)
```

### 5.3.9.7 How to get the row position of the AddNewRecord in the GridGroupingControl

You can get the row index of the AddNewRecord using the **SourceListRecordChanged** event handler. In the event handler, you can check the **e.Action** property and get the row position from the **FilteredRecordsCollection**.

**[C#]**

```
void gridGroupingControl1_SourceListRecordChanged(object sender,  
RecordChangedEventArgs e)  
{  
if (e.Action == RecordChangedType.Added)  
{  
int offset = gridGroupingControl1.TableControl.GetMinimumTopRowIndex();  
int newIndex =  
e.TableListChangedEventArgs.Table.FilteredRecords.IndexOf(e.Record) +
```

```
offset;
Console.WriteLine(newRowIndex);
}
}
```

**[VB]**

```
Private Sub gridGroupingControl1_SourceListRecordChanged(ByVal sender
As Object, ByVal e As RecordChangedEventArgs)
If e.Action = RecordChangedType.Added Then
Dim offset As Integer =
gridGroupingControl1.TableControl.GetMinimumTopRowIndex()
Dim newIndex As Integer =
e.TableListChangedEventArgs.Table.FilteredRecords.IndexOf(e.Record) +
offset
Console.WriteLine(newIndex)
End If
End Sub
```

### 5.3.9.8 What are the events fired when records are collapsing or collapsed?

When the records are collapsing or collapsed, the following events are fired.

**[C#]**

```
// The RecordCollapsed event gets fired after the record is collapsed
private void gridGroupingControl1_RecordCollapsed(object sender,
Syncfusion.Grouping.RecordEventArgs e)
{
    Console.WriteLine("collapsed"+ e.Record.Info);
}

// The RecordCollapsing event gets fired when the record is collapsing
private void gridGroupingControl1_RecordCollapsing(object sender,
Syncfusion.Grouping.RecordEventArgs e)
{
    Console.WriteLine("collapsing"+ e.Record.Info);
}
```

**[VB]**

```
' The RecordCollapsed event gets fired after the record is collapsed
Private Sub gridGroupingControll1_RecordCollapsed(ByVal sender As Object, ByVal e As Syncfusion.Grouping.RecordEventArgs) Handles gridGroupingControll1.RecordCollapsed
    Console.WriteLine("Collapsed" + e.Record.Info)
End Sub

' The RecordCollapsing event gets fired when the record is collapsing
Private Sub gridGroupingControll1_RecordCollapsing(ByVal sender As Object, ByVal e As Syncfusion.Grouping.RecordEventArgs) Handles gridGroupingControll1.RecordCollapsing
    Console.WriteLine("Collapsing" + e.Record.Info)
End Sub
```



**Note:** This applies only when nested tables are used.

### 5.3.9.9 What are the events fired when the records are expanded / expanding?

When the records are expanding or expanded, the following events are fired.

#### [C#]

```
// The RecordExpanded event gets fired after the record is expanded
private void gridGroupingControll1_RecordExpanded(object sender,
Syncfusion.Grouping.RecordEventArgs e)
{
    Console.WriteLine("Expanded"+ e.Record.Info);
}

// The RecordExpanding event gets fired when the record is expanding
private void gridGroupingControll1_RecordExpanding(object sender,
Syncfusion.Grouping.RecordEventArgs e)
{
    Console.WriteLine("Expanding"+ e.Record.Info);
}
```

#### [VB]

```
' The RecordExpanded event gets fired after the record is expanded
Private Sub gridGroupingControll1_RecordExpanded(ByVal sender As Object,
ByVal e As Syncfusion.Grouping.RecordEventArgs) Handles
```

```
gridGroupingControl1.RecordExpanded
    Console.WriteLine("Expanded" + e.Record.Info)
End Sub

' The RecordExpanding event gets fired when the record is expanding
Private Sub gridGroupingControl1_RecordExpanding(ByVal sender As
Object, ByVal e As Syncfusion.Grouping.RecordEventArgs) Handles
gridGroupingControl1.RecordExpanding
    Console.WriteLine("Expanding" + e.Record.Info)
End Sub
```



**Note:** This applies only when nested tables are used.

### 5.3.9.10 What are the events fired when the record values are changed?

When the record values are changed, the following events are fired.

#### [C#]

```
// The RecordValueChanged event gets fired after the record's value is
changed.
private void gridGroupingControl1_RecordValueChanged(object
sender, Syncfusion.Grouping.RecordValueChangedEventArgs e)
{
    Console.WriteLine("Changed " + e.Record.Info);
}

// The RecordValueChanging event gets fired while the record's value is
changing.
private void gridGroupingControl1_RecordValueChanging(object
sender, Syncfusion.Grouping.RecordValueChangingEventArgs e)
{
    Console.WriteLine("Changing " + e.Record.Info);
}
```

#### [VB]

```
' The RecordValueChanged event gets fired after the record's value is
changed.
Private Sub gridGroupingControl1_RecordValueChanged(ByVal sender As
Object, ByVal e As Syncfusion.Grouping.RecordValueChangedEventArgs)
```

```
Handles gridGroupingControl1.RecordValueChanged
    Console.WriteLine("Changed " + e.Record.Info)
End Sub

' The RecordValueChanging event gets fired while the record's value is
changing.
Private Sub gridGroupingControl1_RecordValueChanging(ByVal sender As
Object, ByVal e As Syncfusion.Grouping.RecordValueChangingEventArgs)
Handles gridGroupingControl1.RecordValueChanging
    Console.WriteLine("Changing " + e.Record.Info)
End Sub
```

### 5.3.10 Datasource

This section comprises the following topics:

#### 5.3.10.1 How to Get the Position of a Row in the DataSource from the Current Record

From the row index, you can get the element displayed at that row. If it is a record row, then the element's parent record's unsorted position will give the underlying **DataRow** position.

##### Example

```
[C#]

Table table = e.TableControl.Table;

// Get the current display element.
Element el = table.DisplayElements[e.rowIndex];

// Get the current record.
Record r = el.ParentRecord;

// Find its row position.
int dataRowPos = table.UnsortedRecords.IndexOf(r);

// Retrieve the corresponding data row from the datasource.
```

```
CustomersDataRow row = dataSoure.Rows[dataRowPos];  
  
// Access the CutomerId value of the current record.  
string id = row.CustomerId;
```

**[VB.NET]**

```
Dim table As Table = e.TableControl.Table  
  
' Get the current display element.  
Dim el As Element = table.DisplayElements(e.rowIndex)  
  
' Get the current record.  
Dim r As Record = el.ParentRecord  
  
' Find its row position.  
Dim dataRowPos As Integer = table.UnsortedRecords.IndexOf(r)  
  
' Retrieve the corresponding data row from the datasource.  
Dim row As CustomersDataRow = dataSoure.Rows(dataRowPos)  
  
' Access the CutomerId value of the current record.  
Dim id As String = row.CustomerId
```

### 5.3.10.2 How to set up a datasource to the grouping grid

You can set a datasource to the GroupingGrid using the following code.

**[C#]**

```
// Assign a datasource to the GroupingGrid  
this.gridGroupingControll1.DataSource = dataTable;
```

**[VB.NET]**

```
' Assign a datasource to the GroupingGrid  
Me.gridGroupingControll1.DataSource = dataTable
```

## 5.3.11 Groups

This section comprises the following topics:

### 5.3.11.1 How to access all the groups

To access all the groups and the records categorized under it, use the following code.

[C#]

```
this.iterate(this.gridGroupingControl1.Table.TopLevelGroup);

public void iterate(Group g)
{
    System.Diagnostics.Trace.WriteLine("GroupLevel = "+g.GroupLevel);
    System.Diagnostics.Trace.WriteLine(g.Info);
    foreach(Record r in g.Records)
    {
        System.Diagnostics.Trace.WriteLine(r.Info);
    }
    foreach(Group gr in g.Groups)
    {
        iterate(gr);
    }
}
```

[VB]

```
Me.iterate(Me.gridGroupingControl1.Table.TopLevelGroup)

Public Sub iterate(ByVal g As Group)
    System.Diagnostics.Trace.WriteLine("GroupLevel = " +
g.GroupLevel.ToString())
    System.Diagnostics.Trace.WriteLine(g.Info)
    For Each r As Record In g.Records
        System.Diagnostics.Trace.WriteLine(r.Info)
    Next r
    For Each gr As Group In g.Groups
        iterate(gr)
    Next gr
End Sub
```

### 5.3.11.2 How to access a particular group

To access a particular group and the records categorized under it, use the following code.

#### [C#]

```
//Accessing a particular group and the categorized records under it
this.gridGroupingControl1.TableDescriptor.GroupedColumns.Add("Col1");

// Using category
this.iterate(this.gridGroupingControl1.Table.TopLevelGroup.Groups["row6
col1"]);

//Or using index
//this.iterate(this.gridGroupingControl1.Table.TopLevelGroup.Groups[6])
;

public void iterate(Group g)
{
    System.Diagnostics.Trace.WriteLine("GroupLevel = "+g.GroupLevel);
    System.Diagnostics.Trace.WriteLine(g.Info);
    foreach(Record r in g.Records)
    {
        System.Diagnostics.Trace.WriteLine(r.Info);
    }
    foreach(Group gr in g.Groups)
    {
        iterate(gr);
    }
}
```

#### [VB]

```
'Accessing a particular group and the categorized records under it
    Me.gridGroupingControl1.TableDescriptor.GroupedColumns.Add("Col1")
' Using category
    Me.iterate(Me.gridGroupingControl1.Table.TopLevelGroup.Groups("row6
col1"))

Public Sub iterate(ByVal g As Group)
    System.Diagnostics.Trace.WriteLine("GroupLevel = " +
g.GroupLevel.ToString())
```

```
System.Diagnostics.Trace.WriteLine(g.Info)
For Each r As Record In g.Records
    System.Diagnostics.Trace.WriteLine(r.Info)
Next r
For Each gr As Group In g.Groups
    iterate(gr)
Next gr
End Sub
```

### 5.3.11.3 How to access the group from the DisplayElements

To access the group from the DisplayElements, use the following code snippet.

[C#]

```
// For all the display elements in the Table
foreach(Element el in gridGroupingControl1.Table.DisplayElements)
{
    // DisplayElementKind.Record or DisplayElementKind.Summary
    if(el.Kind==DisplayElementKind.Record ||
DisplayElementKind.Summary)
    {
        Group g = el.ParentGroup;
        System.Diagnostics.Trace.WriteLine(g.Info);
    }
}
```

[VB]

```
' For all the display elements in the Table
For Each el As Element In gridGroupingControl1.Table.DisplayElements
    ' DisplayElementKind.Record or DisplayElementKind.Summary
    If el.Kind=DisplayElementKind.Record OrElse
el.Kind=DisplayElementKind.Summary Then
        Dim g As Group = el.ParentGroup
        System.Diagnostics.Trace.WriteLine(g.Info)
    End If
Next el
```

#### 5.3.11.4 How to access the group from the Record

To access the group from the record, use the following code snippet.

[C#]

```
// For all the display elements in the Table
foreach(Element el in gridGroupingControl1.Table.DisplayElements)
{
    // DisplayElementKind.Record or DisplayElementKind.Summary
    if(el.Kind==DisplayElementKind.Record || 
DisplayElementKind.Summary)
    {
        Group g = el.ParentGroup;
        System.Diagnostics.Trace.WriteLine(g.Info);
    }
}
```

[VB]

```
' For all the display elements in the Table
For Each el As Element In gridGroupingControl1.Table.DisplayElements
    ' DisplayElementKind.Record or DisplayElementKind.Summary
    If el.Kind=DisplayElementKind.Record OrElse
el.Kind=DisplayElementKind.Summary Then
        Dim g As Group = el.ParentGroup
        System.Diagnostics.Trace.WriteLine(g.Info)
    End If
Next el
```

#### 5.3.11.5 How to apply grouping properties for a particular column

Grouping properties for a particular column can be applied using the below code snippet.

[C#]

```
// Setting the color of any group cell in column 'Col1'.
this.gridGroupingControl1.TableDescriptor.Columns["Col1"].GroupByAppearance.AnyGroupCell.BackColor=Color.LightBlue;
```

```
// Setting the FilterBar cell appearance to be raised.  
this.gridGroupingControl1.TableDescriptor.Columns["Col1"].GroupByAppearance.FilterBarCell.CellAppearance=GridCellAppearance.Raised;  
  
// Setting the cell type of any record cell.  
this.gridGroupingControl1.TableDescriptor.Columns["Col1"].GroupByAppearance.AnyRecordFieldCell.CellType="ComboBox";
```

**[VB]**

```
' Shows how to apply grouping properties for particular column.  
' Setting the color of any record cell in column 'Col1'.  
Me.gridGroupingControl1.TableDescriptor.Columns("Col1").GroupByAppearance.AnyRecordFieldCell.BackColor=Color.LightBlue  
  
' Setting the FilterBar cell appearance to be raised.  
Me.gridGroupingControl1.TableDescriptor.Columns("Col1").GroupByAppearance.FilterBarCell.CellAppearance=GridCellAppearance.Raised  
  
' Setting the cell type of any record cell.  
Me.gridGroupingControl1.TableDescriptor.Columns("Col1").GroupByAppearance.AnyRecordFieldCell.CellType="ComboBox"
```

### 5.3.11.6 How to apply grouping properties for ChildLevelGroups

Grouping properties for ChildLevelGroups can be applied using the below code snippet.

**[C#]**

```
//Hiding the caption text of the group.  
this.gridGroupingControl1.TableDescriptor.ChildGroupOptions.ShowCaption=false;  
  
//Hiding the plus / minus sign from the group.  
this.gridGroupingControl1.TableDescriptor.ChildGroupOptions.ShowCaptionPlusMinus=false;  
  
// Adding the Filter Bar to the Child level groups.  
this.gridGroupingControl1.TableDescriptor.ChildGroupOptions.ShowFilterBar=true;  
  
// Hiding the AddNewRecord field before details row.
```

```
this.gridGroupingControl1.TableDescriptor.ChildGroupOptions.ShowAddNewRecordBeforeDetails=false;

//Showing the footer of the group.
this.gridGroupingControl1.TableDescriptor.ChildGroupOptions.ShowGroupFooter=true;
```

**[VB]**

```
' Hiding the caption text of the group.
Me.gridGroupingControl1.TableDescriptor.ChildGroupOptions.ShowCaption=False

' Hiding the plus / minus symbols present in the group.
Me.gridGroupingControl1.TableDescriptor.ChildGroupOptions.ShowCaptionPlusMinus=False

' Showing the FilterBar.
Me.gridGroupingControl1.TableDescriptor.ChildGroupOptions.ShowFilterBar=True

' Hiding the AddNewRecord before details.
Me.gridGroupingControl1.TableDescriptor.ChildGroupOptions.ShowAddNewRecordBeforeDetails=False

' Showing the group footer.
Me.gridGroupingControl1.TableDescriptor.ChildGroupOptions.ShowGroupFooter=True
```

### 5.3.11.7 How to apply grouping properties for TopLevelGroups

Grouping properties for TopLevelGroups can be applied using the below code snippet.

**[C#]**

```
// Hiding the AddNewRecord field before details row.
this.gridGroupingControl1.TableDescriptor.TopLevelGroupOptions.ShowAddNewRecordBeforeDetails=false;

// Hiding the Header cells.
this.gridGroupingControl1.TableDescriptor.TopLevelGroupOptions.ShowColumnHeaders=false;
```

```
// Adding the Filter Bar to the Child level groups.  
this.gridGroupingControl1.TableDescriptor.TopLevelGroupOptions.ShowFilterBar=true;  
  
//Setting the caption.  
this.gridGroupingControl1.TableDescriptor.TopLevelGroupOptions.CaptionText="Hai";  
  
//Shows the group footer.  
this.gridGroupingControl1.TableDescriptor.TopLevelGroupOptions.ShowGroupFooter=true;
```

**[VB]**

```
' Hiding the AddNewRecord before details.  
Me.gridGroupingControl1.TableDescriptor.TopLevelGroupOptions.ShowAddNewRecordBeforeDetails=False  
  
' Hiding the column headers.  
  
Me.gridGroupingControl1.TableDescriptor.TopLevelGroupOptions.ShowColumnHeaders=False  
  
' Adding the Filter Bar to the Top level groups.  
  
Me.gridGroupingControl1.TableDescriptor.TopLevelGroupOptions.ShowFilterBar=True  
  
'Setting the caption.  
Me.gridGroupingControl1.TableDescriptor.TopLevelGroupOptions.CaptionText="Hai"  
  
'Shows the group footer.  
Me.gridGroupingControl1.TableDescriptor.TopLevelGroupOptions.ShowGroupFooter=True
```

### 5.3.11.8 How to derive user defined groups

This can be done using the below code.

**[C#]**

```
//group "Col2" using a custom categorizer and Comparer
```

```
Syncfusion.Grouping.SortColumnDescriptor cd = new
Syncfusion.Grouping.SortColumnDescriptor("Col2");
cd.Categorizer = new CustomCategorizer();
cd.Comparer = new CustomComparer();
this.gridGroupingControll.TableDescriptor.GroupedColumns.Add(cd);

// Custom comparer function.
public int Compare(object x, object y)
{
    if(x == null)
        return -1;
    else if(y == null)
        return 100;
    else
    {
        int i = int.Parse(x.ToString());
        int j = int.Parse(y.ToString());
        return i - j;
    }
}

// Custom categorizer function.
public static int GetCategory(int i)
{
    int ret = 0;
    if(i < 10)
        ret = 1;
    else if(i >= 10 && i < 20)
        ret = 2;
    else if(i >= 20 && i < 30)
        ret = 3;
    else if(i >= 30 && i < 40)
        ret = 4;
    else
        ret = 5;
    return ret;
}
```

**[VB]**

```
'group "Col2" using a custom categorizer and Comparer
Dim cd As Syncfusion.Grouping.SortColumnDescriptor = New
Syncfusion.Grouping.SortColumnDescriptor("Col2")
cd.Categorizer = New CustomCategorizer()
cd.Comparer = New CustomComparer()
```

```
Me.gridGroupingControl1.TableDescriptor.GroupedColumns.Add(cd)

'Custom Comparer function.
Public Function Compare(ByVal x As Object, ByVal y As Object) As Integer Implements IComparer.Compare
    If x Is Nothing Then
        Return -1
    ElseIf y Is Nothing Then
        Return 100
    Else
        Dim i As Integer = Integer.Parse(x.ToString())
        Dim j As Integer = Integer.Parse(y.ToString())
        Return i - j
    End If
End Function

'Custom Categorizer function.
Public Shared Function GetCategory(ByVal i As Integer) As Integer
    Dim ret As Integer = 0
    If i < 10 Then
        ret = 1
    ElseIf i >= 10 AndAlso i < 20 Then
        ret = 2
    ElseIf i >= 20 AndAlso i < 30 Then
        ret = 3
    ElseIf i >= 30 AndAlso i < 40 Then
        ret = 4
    Else
        ret = 5
    End If

    Return ret
End Function
```

### 5.3.11.9 What are the events fired when the group is collapsing / collapsed?

Following are the events triggered when the group is collapsing or collapsed.

[C#]

```
// Shows the GroupCollapsing event.  
private void gridGroupingControl1_GroupCollapsing(object sender,  
Syncfusion.Grouping.GroupEventArgs e)  
{  
// Shows all the records in the group which is being collapsed.  
foreach(Record r in e.Group.Records)  
{  
    Console.WriteLine("Collapsing event "+r.Info);  
}  
}  
  
// Shows the GroupCollapsed event.  
private void gridGroupingControl1_GroupCollapsed(object sender,  
Syncfusion.Grouping.GroupEventArgs e)  
{  
    // Shows all the records in the group which has collapsed.  
    foreach(Record r in e.Group.Records)  
    {  
        Console.WriteLine("Collapsed event "+r.Info);  
    }  
}
```

**[VB]**

```
' Shows the GroupCollapsed events.  
Private Sub gridGroupingControl1_GroupCollapsed(ByVal sender As  
Object, ByVal e As Syncfusion.Grouping.GroupEventArgs) Handles  
gridGroupingControl1.GroupCollapsed()  
    For Each r As Record In e.Group.Records  
        Console.WriteLine("Collapsed event " + r.Info)  
    Next r  
End Sub  
  
' Shows the GroupCollapsing events.  
Private Sub gridGroupingControl1_GroupCollapsing(ByVal sender As  
Object, ByVal e As Syncfusion.Grouping.GroupEventArgs) Handles  
gridGroupingControl1.GroupCollapsing()  
    For Each r As Record In e.Group.Records  
        Console.WriteLine("Collapsing event " + r.Info)  
    Next r  
End Sub
```

### 5.3.11.10 What are the events fired when the group is expanded / expanding?

Following are the events fired when the group is expanding or expanded.

#### [C#]

```
//Shows the GroupExpanding event.  
private void gridGroupingControll1_GroupExpanding(object sender,  
Syncfusion.Grouping.GroupEventArgs e)  
{  
    // Shows all the records in the group which is being Expanded.  
    foreach(Record r in e.Group.Records)  
    {  
        Console.WriteLine("Expanding event "+r.Info);  
    }  
}  
  
//Shows the GroupExpanded event.  
private void gridGroupingControll1_GroupExpanded(object sender,  
Syncfusion.Grouping.GroupEventArgs e)  
{  
    // Shows all the records in the group which has expanded.  
    foreach(Record r in e.Group.Records)  
    {  
        Console.WriteLine("Expanded event "+r.Info);  
    }  
}
```

#### [VB]

```
//Shows the GroupExpanding event.  
Private Sub gridGroupingControll1_GroupExpanding(sender As Object, e As Syncfusion.Grouping.GroupEventArgs)  
{  
    // Shows all the records in the group which is being Expanded.  
    For Each r In e.Group.Records  
    {  
        Console.WriteLine("Expanding event "+r.Info);  
    }  
}  
  
//Shows the GroupExpanded event.  
Private Sub gridGroupingControll1_GroupExpanded(sender As Object, e As Syncfusion.Grouping.GroupEventArgs)
```

```
{  
    // Shows all the records in the group which has expanded.  
    foreach(Record r in e.Group.Records)  
    {  
        Console.WriteLine("Expanded event "+r.Info);  
    }  
}
```

### 5.3.12 How to Move Groups Upward/Downward Using Custom Comparer

To enable moving the groups using custom **Comparer** in the GridGrouping control, an **IComparer** object and **QueryValue** event should be handled.

The **Comparer** object allows you to control the movement of groups in GGC. The group moving logic should be implemented in custom Comparer. After customizing the sorting logic through **IComparer**, the **QueryValue** event should be applied to make the groups move among its position.

```
[C#]  
  
public class CustomGroupSortOrderComparer : IGroupSortOrderComparer  
{  
  
    string sortColumnDescriptorName;  
    public CustomGroupSortOrderComparer(string sortColumnDescriptorName)  
    {  
        this.sortColumnDescriptorName = sortColumnDescriptorName;  
    }  
    #region IComparer Members  
    public int Compare(object x, object y)  
    {  
        Group x1 = x as Group;  
        Group y1 = y as Group;  
        return  
            String.Compare(x1.Records[0].GetValue(sortColumnDescriptorName).ToString(),  
                           y1.Records[0].GetValue(sortColumnDescriptorName).ToString());  
    }  
    #endregion  
}
```

```
//In the above codes, IComparer is used to compare the groups to move. If X1 is greater than the Y1 groups, it will return either X1 or Y1. This Comparer class is used to move all groups from their original position. sortColumnDescriptorName describes the grouped column.

//Handle the QueryValue event to make the grid use the custom Comparer.

void gridGroupingControll_QueryValue(object sender, FieldValueEventArgs e)
{
//Code...
if (g == e.Record.ParentGroup)
{
ArrayList list = new ArrayList(table.Values.Count);
list.AddRange(table.Values);
list.Sort();
if (flag1 || flag2)
{
if (flag1)
{
index =
list.IndexOf(table[e.Record.GetValue(e.TableDescriptor.GroupedColumns[0].Name)]) - 1;
}
else if (flag2)
{
index =
list.IndexOf(table[e.Record.GetValue(e.TableDescriptor.GroupedColumns[0].Name)]) + 1;
}
if (index < 0 || index >= list.Count)
{
MessageBox.Show("Cannot go further!");
}
//Code...
}
//If you click the Move Up button, the index of the current record is
decreased using the below code.
index =
list.IndexOf(table[e.Record.GetValue(e.TableDescriptor.GroupedColumns[0].Name)]) - 1;

//If you click the Move Down button then the index of the current record
is increased. Refer to the below code.
index =
list.IndexOf(table[e.Record.GetValue(e.TableDescriptor.GroupedColumns[0].Name)]) - 1;

//Code...

}
}
```

```
[VB]
public class CustomGroupSortOrderComparer : IGroupSortOrderComparer

Dim sortColumnDescriptorName As String
public CustomGroupSortOrderComparer(String sortColumnDescriptorName)
Me.sortColumnDescriptorName = sortColumnDescriptorName
'#Region "IComparer Members"
public Integer Compare(Object x, Object y)
Dim x1 As Group = TryCast(x, Group)
Dim y1 As Group = TryCast(y, Group)
Return
String.Compare(x1.Records(0).GetValue(sortColumnDescriptorName).ToString(
), y1.Records(0).GetValue(sortColumnDescriptorName).ToString())
'#End Region

'In the above codes, IComparer is used to compare the groups to move. If
X1 is greater than the Y1 groups it will return the groups, either X1 or
Y1. This Comparer class is used to move all groups from their original
position. sortColumnDescriptorName describes the grouped column.

'Handle the QueryValue event to make the grid use the custom Comparer.

void gridGroupingControll_QueryValue(Object sender, FieldValueEventArgs
e)
'Code...
If g = e.Record.ParentGroup Then
Dim list As New ArrayList(table.Values.Count)
list.AddRange(table.Values)
list.Sort()
If flag1 OrElse flag2 Then
If flag1 Then
index =
list.IndexOf(table(e.Record.GetValue(e.TableDescriptor.GroupedColumns(0).
Name))) - 1
ElseIf flag2 Then
index =
list.IndexOf(table(e.Record.GetValue(e.TableDescriptor.GroupedColumns(0).
Name))) + 1
End If
If index < 0 OrElse index >= list.Count Then
MessageBox.Show("Cannot go further!")
End If

'Code...

End If
'If you click the Move Up button the index of the current record is
decreased using the below code.
index =
list.IndexOf(table(e.Record.GetValue(e.TableDescriptor.GroupedColumns(0).
```

```
Name)) - 1  
  
'If you click the Move Down button then the index of the current record  
is increased. Refer to the below code.  
index =  
list.IndexOf(table(e.Record.GetValue(e.TableDescriptor.GroupedColumns(0).  
Name))) - 1  
'Code...  
  
End Sub
```

Refer to the following sample file for more details.

[http://www.syncfusion.com/downloads/Support/DirectTrac/98850/Demo%20\(2\)539070983.zip](http://www.syncfusion.com/downloads/Support/DirectTrac/98850/Demo%20(2)539070983.zip)

### 5.3.13 Sorting

#### 5.3.13.1 How to Do Custom Sorting for a Specified String Column

You can customize the sorting and you can set the rows with an empty string, to the last. This can be done by making use of the **Compare** method of the **IComparer** Class. You can write your custom code for the **Compare()** method where you can set the empty string rows as last. This can be achieved by finding the length of the string in the compare() method and canceling its sorting thereby pushing the empty strings at the last.

##### Example

Here is the code for implementing the string length compare method.

[C#]

```
public int Compare(object x, object y)  
{  
    if(x == null && y == null)  
        return 0;  
  
    // Don't include empty strings for comparison.  
    else if(x == System.DBNull.Value)  
    {  
        return 0;  
    }  
    else if(y == System.DBNull.Value)  
    {  
        return 0;
```

```
        }
    else
    {
        return ((IComparable) x.ToString()).CompareTo(y.ToString());
    }
}

SortColumnDescriptor cd = new
Syncfusion.Grouping.SortColumnDescriptor();
this.gridGroupingControl1.TableDescriptor.SortedColumns.Add(cd);
cd.SortDirection = System.ComponentModel.ListSortDirection.Descending;
cd.Comparer = new StrLenComparer();
```

**[VB .NET]**

```
Public Function Compare(ByVal x As Object, ByVal y As Object) As
Integer
    If x Is Nothing AndAlso y Is Nothing Then
        Return 0

        ' Don't include empty strings for comparison.
    Else If x Is System.DBNull.Value Then
        Return 0
    Else If y Is System.DBNull.Value Then
        Return 0
    Else
        Return (CType(x.ToString(),
IComparable)).CompareTo(y.ToString()))
    End If
End Function

Private cd As SortColumnDescriptor = New
Syncfusion.Grouping.SortColumnDescriptor()
Me.gridGroupingControl1.TableDescriptor.SortedColumns.Add(cd)
Private cd.SortDirection =
System.ComponentModel.ListSortDirection.Descending
Private cd.Comparer = New StrLenComparer()
```

### 5.3.13.2 How to perform Custom Sorting in GridGroupingControl

To use custom sorting in the *GridGroupingControl*, you need to disable the default sorting using the *TableControlQueryAllowSortColumn* and the *TableControlCellClick* events.

The following are the steps to disable default sorting and use custom sorting:

1. Set the *AllowSort* property of the *TableControlQueryAllowSortColumn* event to false to disable the default sorting. The following code illustrates this:

```
[c#]
void gridGroupingControll_TableControlQueryAllowSortColumn(object sender,
GridQueryAllowSortColumnEventArgs e)
{
    if (checkBox1.Checked && e.Column.Name == "Source")
        e.AllowSort = false;
    else
        e.AllowSort = true;
}
```

**[VB]**

```
Private Sub gridGroupingControll_TableControlQueryAllowSortColumn(ByVal
sender As Object, ByVal e As GridQueryAllowSortColumnEventArgs)
    If e.Column.Name = "Source" Then
        e.AllowSort = False
    Else
        e.AllowSort = True
    End If
End Sub
```

2. Ensure the the *CellType* is *ColumnHeaderCell* in *TableControlCellClick* event and then call your sorting method. The following code illustrates this:

```
[c#]
void gridGroupingControll_TableControlCellClick(object sender,
GridTableViewCellEventArgs e)
{
    GridTableCellStyleInfo style =
e.TableControl.GetTableViewStateInfo(e.Inner.RowIndex, e.Inner.ColIndex);
    if (style.TableCellIdentity.TableCellType ==
GridTableCellType.ColumnHeaderCell)
    {
        if (style.TableCellIdentity.Column != null &&
style.TableCellIdentity.Column.Name == "Source")
        {

```

```
        CustomSorting(style.TableCellIdentity.Column.Name);
    }
}
}
```

[VB]

```
Private Sub gridGroupingControll_TableControlCellClick(ByVal sender As Object, ByVal e As GridTableCellClickEventArgs)
    Dim style As GridTableCellStyleInfo =
e.TableControl.GetTableViewStyleInfo(e.Inner.RowIndex, e.Inner.ColIndex)
    If style.TableCellIdentity.TableCellType =
GridTableCellType.ColumnHeaderCell Then
        If style.TableCellIdentity.Column IsNot Nothing AndAlso
style.TableCellIdentity.Column.Name = "Source" Then

            CustomSorting(style.TableCellIdentity.Column.Name)
        End If
    End If
End Sub
```

In the preceding code example the *CustomSorting()* is the user defined method. In this method, the values from the grid are added to an *ArrayList* and sorted. And then the sorted values are stored back into the grid. The following code illustrates this:

```
[c#]
public void CustomSorting(string sortColumn)
{
    ArrayList list = new ArrayList();
    foreach (Record r in this.gridGroupingControll.Table.Records)
    {
        list.Add(r.GetValue(sortColumn));
    }
    list.Sort();

    for (int i = 0; i <
this.gridGroupingControll.Table.Records.Count; i++)
    {

this.gridGroupingControll.Table.Records[i].GetRecord().SetValue(sortColumn,
list[i].ToString());
    }
    this.gridGroupingControll.Refresh();
```

```
}
```

```
[VB]
Public Sub CustomSorting(ByVal sortColumn As String)
    Dim list As New ArrayList()
    For Each r As Record In Me.gridGroupingControl1.Table.Records
        list.Add(r.GetValue(sortColumn))
    Next
    list.Sort()

    For i As Integer = 0 To
Me.gridGroupingControl1.Table.Records.Count - 1

Me.gridGroupingControl1.Table.Records(i).GetRecord().SetValue(sortColumn,
list(i).ToString())
    Next
    Me.gridGroupingControl1.Refresh()
End Sub
```

### 5.3.13.3 How to prevent sorting if the column has the same value in each cell?

To prevent sorting column with same cell values, the choices are obtained through the 'GetFilterBarChoices()' method from the filter bar cell model. This retrieves the unique values of the respective column. The 'OptimizeFilterPerformance' property can also be enabled to get these choices.

```
[C#]
```

```
GridTableFilterBarCellModel filterModel = null; //Declare this
globally.

//In form load.

filterModel =
this.gridGroupingControl1.TableModel.CellModels["FilterBarCell"]
```

```
as GridTableFilterBarCellModel;

this.gridGroupingControl1.OptimizeFilterPerformance = true;

//In event

private void
gridGroupingControl1_TableControlQueryAllowSortColumn(object
sender,
Syncfusion.Windows.Forms.Grid.Grouping.GridQueryAllowSortColumnEv
entArgs e)

{

int rowIndex = e.TableControl.Table.Records[0].GetRowIndex();

int colIndex = e.Column.GetRelativeColumnIndex() + 1; //Check the
col index with your application.

GridTableCellStyleInfo tableStyleInfo =
e.TableControl.GetTableViewStyleInfo(rowIndex, colIndex);

object[] items =
(object[])filterModel.GetFilterBarChoices(tableStyleInfo.TableCel
lIdentity);

if (items.Length == 2 && items[0].GetType() ==
typeof(System.DBNull)) //Check if there is only one unique item.

{

e.AllowSort = false;

}

}
```

**[VB]**

```
Dim filterModel As GridTableFilterBarCellModel = Nothing 'Declare
this globally.

'In form load.
```

```
filterModel =
TryCast(Me.gridGroupingControl1.TableModel.CellModels("FilterBarCell"), GridTableFilterBarCellModel)

Me.gridGroupingControl1.OptimizeFilterPerformance = True

'In event

private void
gridGroupingControl1_TableControlQueryAllowSortColumn(Object
sender,
Syncfusion.Windows.Forms.Grid.Grouping.GridQueryAllowSortColumnEv
entArgs e)

Dim rowIndex As Integer =
e.TableControl.Table.Records(0).GetRowIndex()

Dim colIndex As Integer = e.Column.GetRelativeColumnIndex() + 1
'Check the col index with your application.

Dim tableStyleInfo As GridTableCellStyleInfo =
e.TableControl.GetTableViewStyleInfo(rowIndex, colIndex)

Dim items() As Object =
 CType(filterModel.GetFilterBarChoices(tableStyleInfo.TableCellIde
ntity), Object())

If items.Length = 2 AndAlso items(0).GetType() Is
GetType(System.DBNull) Then 'Check if there is only one unique
item.

e.AllowSort = False
```

## 5.4 Common to GridControl, GridDataBoundGrid and GridGrouping

The tasks and solutions discussed in this section apply to either a GridControl or a GridDataBoundGrid.

## 5.4.1 How to Add Conditional Formatting to Rows

### Introduction

Say you want to color a row if the value in column 2 is larger than 10 (or any logical condition that you can evaluate). You can do something like this, by using the **PrepareViewStyleInfo** event. This event is raised immediately prior to the cell being drawn and gives you a chance to modify the **GridStyleInfo** object that determines the appearance of the cell. Since you want to affect a visual change on the whole row when a single cell value is modified, you will need to tell the grid to redraw the whole row when the current cell moves instead of just redrawing the affected cells.

You can use the same technique to conditionally format columns or cells.

### Example

#### [C#]

```
// Tell the grid to redraw the whole row as current cell moves (say in
// Form.Load).
this.grid.Model.Options.RefreshCurrentCellBehavior =
GridRefreshCurrentCellBehavior.RefreshRow;

// The event handler.
private void grid_PreparesViewStyleInfo(object sender,
GridPrepareViewStyleInfoEventArgs e)
{
    if(e.RowIndex > 0 && e.ColumnIndex > 0)
    {
        double d;
        string val = (e.ColumnIndex == 2) ? e.Style.Text :
this.gridDataBoundGrid1[e.RowIndex, 2].Text;
        if(double.TryParse(val, System.Globalization.NumberStyles.Any,
null, out d)
            && d > 10)
        {
            e.Style.BackColor = Color.LightGoldenrodYellow;
        }
    }
}
```

#### [VB .NET]

```
' Tell the grid to redraw the whole row as current cell moves (say in
// Form.Load).
Me.grid.Model.Options.RefreshCurrentCellBehavior =
GridRefreshCurrentCellBehavior.RefreshRow
```

```
' The event handler.  
Private Sub grid_PreparesViewStyleInfo(ByVal sender As Object, ByVal e  
As GridPrepareViewStyleEventArgs)  
    If e.RowIndex > 0 And e.ColumnIndex > 0 Then  
        Dim d As Double  
        Dim val As String  
        If e.ColumnIndex = 2 Then  
            val = e.Style.Text  
        Else  
            val = Me.gridDataBoundGrid1(e.RowIndex, 2).Text  
        End If  
        If Double.TryParse(val, System.Globalization.NumberStyles.Any,  
Nothing, d) And d > 10 Then  
            e.Style.BackColor = Color.LightGoldenrodYellow  
        End If  
    End If  
  
' Grid_PreparesViewStyleInfo  
End Sub
```

## 5.4.2 How to Capture Function Keys When the Current Cell is in Active Edit Mode

### Introduction

While the grid cell is not actively being edited, the grid itself will get these keystrokes. In this case, event handlers like `grid.CurrentCellKeyDown` and `grid.KeyDown` can be used to catch the keys.

But, when the current cell is actively being edited, the grid does not automatically catch these keys. In this case, you can use the **Grid.CurrentCellControlKeyMessage** to catch the function keys. You will have to manually subscribe to it. Note that this event may be hit multiple times for each keystroke (eg., for both `KeyDown` and `KeyUp`). You can check the `e.Msg` property to see what key message is being processed by your event handler.

### Example

[C#]

```
// Subscribe to the event.  
this.grid.CurrentCellControlKeyMessage += new  
GridCurrentCellControlKeyMessageEventHandler
```

```
(grid_CurrentCellControlKeyMessage);

// The handler.
private void grid_CurrentCellControlKeyMessage(object sender,
GridCurrentCellControlKeyMessageEventArgs e)
{
    Keys keyCode = (Keys)((int)e.Msg.WParam) & Keys.KeyCode;
    Console.WriteLine(keyCode);
    Console.WriteLine(e.Msg);
}
```

**[VB.NET]**

```
' Subscribe to the event.
AddHandler Me.grid.CurrentCellControlKeyMessage, AddressOf
grid_CurrentCellControlKeyMessage

' The handler.
Private Sub grid_CurrentCellControlKeyMessage(ByVal sender As Object,
ByVal e As GridCurrentCellControlKeyMessageEventArgs)
    Dim keyCode As Keys = CType(Fix(e.Msg.WParam), Keys) And
    Keys.KeyCode
    Console.WriteLine(keyCode)
    Console.WriteLine(e.Msg)

' Grid_CurrentCellControlKeyMessage
End Sub
```

### 5.4.3 How to Capture Mouse and Key Events When the Text Box Cell is in an Active State

#### Introduction

The embedded text box control gets the mouse actions while the text box is active. You can subscribe to the embedded text box's events inside a cell by accessing it through the cell's renderer.

#### Example

**[C#]**

```
private void Form1_Load(object sender, System.EventArgs e)
```

```
{  
    // Create TextBoxCellRenderer object.  
    GridTextBoxCellRenderer tbr = (GridTextBoxCellRenderer)  
this.grid.CellRenderers["TextBox"];  
  
    // Handle Renderer.TextBox.MouseDown to capture mouse events.  
    tbr.TextBox.MouseDown += new  
MouseEventHandler(textbox_MouseDown);  
  
    // Handle Renderer.TextBox.KeyUp to capture key events.  
    tbr.TextBox.KeyUp += new KeyEventHandler(textBox_KeyUp);  
}  
  
private void textbox_MouseDown(object sender, MouseEventArgs e)  
{  
    Console.WriteLine("textbox_MouseDown");  
}  
  
private void textBox_KeyUp(object sender, KeyEventArgs e)  
{  
    Console.WriteLine("textBox_KeyUp");  
}
```

**[VB.NET]**

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As  
System.EventArgs)  
  
    ' Create TextBoxCellRenderer object.  
    Dim tbr As GridTextBoxCellRenderer =  
CType(Me.grid.CellRenderers("TextBox"), GridTextBoxCellRenderer)  
  
    ' Handle Renderer.TextBox.MouseDown to capture mouse events.  
    AddHandler tbr.TextBox.MouseDown, AddressOf textbox_MouseDown  
    AddHandler tbr.TextBox.KeyUp, AddressOf textBox_KeyUp  
  
    ' Form1_Load  
    End Sub  
  
    Private Sub textbox_MouseDown(ByVal sender As Object, ByVal e As  
MouseEventArgs)  
        Console.WriteLine("textbox_MouseDown")  
  
    ' Form1_Load  
    End Sub X
```

```
Private Sub textBox_KeyUp(ByVal sender As Object, ByVal e As  
KeyEventArgs)  
    Console.WriteLine("textBox_KeyUp")  
  
    ' TextBox_KeyUp  
End Sub
```

## 5.4.4 How to Change the Color of All Headers

### Introduction

The styles of the Header cells are controlled by base styles. The "Header" base style will affect all column headers including cell 0,0. The "Column Header" **base style** will affect the column headers excluding cell 0,0. The "RowHeader" base style will affect all row headers excluding cell 0,0.

### Example

#### [C#]

```
// Column headers including cell 0,0.  
this.grid.BaseStylesMap["Header"].StyleInfo.BackColor = Color.Blue;  
  
// Column headers excluding cell 0,0.  
this.grid.BaseStylesMap["Column Header"].StyleInfo.BackColor =  
Color.Red;  
  
// Row headers excluding cell 0,0.  
this.grid.BaseStylesMap["Row Header"].StyleInfo.BackColor =  
Color.Green;
```

#### [VB.NET]

```
' Column headers including cell 0,0.  
Me.grid.BaseStylesMap("Header").StyleInfo.BackColor = Color.Blue  
  
' Column headers excluding cell 0,0.  
Me.grid.BaseStylesMap("Column Header").StyleInfo.BackColor = Color.Red  
  
' Row headers excluding cell 0,0.  
Me.grid.BaseStylesMap("Row Header").StyleInfo.BackColor = Color.Green
```

## 5.4.5 How to Change the Size of the Combo Box Button

### Introduction

Change the size of the combobox button by changing the **ButtonBarSize** in the **CellModel** for the control.

### Example

#### [C#]

```
// Create Combobox Cell model object.  
GridComboBoxCellModel model = this.grid.Model.CellModels["ComboBox"] as  
GridComboBoxCellModel;  
  
// Assign a new value to its ButtonBarSize property.  
model.ButtonBarSize = new Size(8, 8);
```

#### [VB .NET]

```
' Create Combobox Cell model object.  
Dim model As GridComboBoxCellModel =  
 CType(Me.Grid.Model.CellModels("ComboBox"), GridComboBoxCellModel)  
  
' Assign a new value to its ButtonBarSize property.  
model.ButtonBarSize = New Size(8, 8)
```

## 5.4.6 How to Change the Mouse Cursor for a GridControl

### Introduction

The simplest way is to derive the grid and override the **OnSetCursor**. You can add additional checks to narrow down where to set the cursor.

### Example

#### [C#]

```
// GridDataBoundGrid
```

```
public class MyGridControl : GridControl
{
    protected override void OnSetCursor(ref Message m)
    {
        base.OnSetCursor(ref m);

        // Always set the cursor to a cross.
        Cursor.Current = Cursors.Cross;

        // or
        // Put special cursor over cell 2,2.
        // Point pt =
        this.PointToClient(Control.MousePosition);
        // int row, col;
        // if(this.PointToRowCol(pt, out row, out col, -1)
        //     && row == 2 && col == 2)
        // {
        //     Cursor.Current = Cursors.Cross;
        // }
    }
}
```

**[VB.NET]**

```
Public Class MyGridControl
    Inherits GridControl 'GridDataBoundGrid

    Protected Overrides Sub OnSetCursor(ByRef m As Message)
        MyBase.OnSetCursor(m)

        ' Always set the cursor to a cross.
        Cursor.Current = Cursors.Cross

        ' OR
        ' Put special cursor over cell 2,2.
        ' Point pt = this.PointToClient(Control.MousePosition);
        ' int row, col;
        ' If Me.PointToRowCol(pt, row, col, -1) AndAlso row = 2
        AndAlso col = 2 Then
            ' Cursor.Current = Cursors.Cross;
            ' End If

        ' OnSetCursor
    End Sub

    ' MyGridControl
```

End Class

## 5.4.7 How to Control the Number of Visible Items in a ComboBox Cell

### Introduction

There is a **GridComboBoxListBoxPart.DropDownRows** property which, you can set to control this. The **GridComboBoxListBoxPart** is the actual control type of the list that is dropped to display the items. But, it is buried a little deep and generally needs an event handler to set it. The reason of using an event handler is that normally a single combobox control is shared among all combobox cells. Each cell can potentially have a different list and may need different numbers of visible rows. So to handle this, you must catch the **Grid.CurrentCellShowingDropDown** event and set the property there depending upon the exact row and column.

### Example

[C#]

```
private void grid_CurrentCellShowingDropDown(object sender,
GridCurrentCellShowingDropDownEventArgs e)
{
    GridControlBase grid = sender as GridControlBase;
    if(grid != null)
    {
        GridCurrentCell cc = grid.CurrentCell;
        GridComboBoxCellRenderer cr = cc.Renderer as
GridComboBoxCellRenderer;

        // Set number of visible items for comboboxes in Row 6 as
4, Row 4 as 7, Row 2 as 10 , and so on.
        if(cc != null)
        {
            if(cc.RowIndex == 6)

((GridComboBoxListBoxPart)cr.ListBoxPart).DropDownRows = 4;
            else if(cc.RowIndex == 4)

((GridComboBoxListBoxPart)cr.ListBoxPart).DropDownRows = 7;
            else if(cc.RowIndex == 2)

((GridComboBoxListBoxPart)cr.ListBoxPart).DropDownRows = 10;
            else
        }
    }
}
```

```
((GridComboBoxListBoxPart)cr.ListBoxPart).DropDownRows = 6;  
    }  
}  
}
```

**[VB.NET]**

```
Private Sub Grid_CurrentCellShowingDropDown(sender As Object, e As  
GridCurrentCellShowingDropDownEventArgs)  
    Try  
        Dim grid As GridControlBase = sender  
        Dim cc As GridCurrentCell = grid.CurrentCell  
        If cc.Renderer Is GetType(GridComboBoxCellRenderer) Then  
            Dim cr As GridComboBoxCellRenderer = cc.Renderer  
  
                // Set number of visible items for comboboxes in Row 6  
                // as 4, Row 4 as 7, Row 2 as 10 , and so on.  
                If cc.RowIndex = 6 Then  
                    CType(cr.ListBoxPart,  
GridComboBoxListBoxPart).DropDownRows = 4  
                ElseIf cc.RowIndex = 4 Then  
                    CType(cr.ListBoxPart,  
GridComboBoxListBoxPart).DropDownRows = 7  
                ElseIf cc.RowIndex = 2 Then  
                    CType(cr.ListBoxPart,  
GridComboBoxListBoxPart).DropDownRows = 10  
                Else  
                    CType(cr.ListBoxPart,  
GridComboBoxListBoxPart).DropDownRows = 6  
                End If  
            End If  
        Catch  
    End Try  
End Sub
```

## 5.4.8 How to Control the Way the Grid Handles Exceptions

### Introduction

Syncfusion.Windows.Forms.ExceptionManager has static members which, you can use to control how the grid handles exceptions.

### Example

To suspend the grid's error handling, try.

**[C#]**

```
// Suspend Grid's Error Handling.  
Syncfusion.Windows.Forms.ExceptionManager.SuspendCatchExceptions()
```

**[VB .NET]**

```
' Suspend Grid's Error Handling.  
Syncfusion.Windows.Forms.ExceptionManager.SuspendCatchExceptions()
```



**Note:** You can also subscribe to an event (`Syncfusion.Windows.Forms.ExceptionManager.ExceptionCatched`) to get any exception thrown and handle them by yourself or re-throw them.

## 5.4.9 How to Control Whether OLE Drag-and-Drop is Supported in the Grid

### Introduction

Whether a grid is an OLE drop target which, is controlled by the **DragDropTargetFlags** in the grids **Model.Options** class. These flags control things like the clipboard format, the type of the data, whether columns or rows can be appended to accommodate the dropped data and whether auto scrolling is supported. Check the enums for **GridDragDropFlags** to see the full set of options.

### Example

**[C#]**

```
// Turn off being a drop target.  
gridControl1.Model.Options.DragDropDropTargetFlags =  
GridDragDropFlags.Disabled;  
  
// Turn on accepting text.  
gridControl1.Model.Options.DragDropDropTargetFlags =  
GridDragDropFlags.Text;  
  
// Accept both text and styles.
```

```
gridControl1.Model.Options.DragDropDropTargetFlags =  
GridDragDropFlags.Text | GridDragDropFlags.Text;
```

**[VB.NET]**

```
' Turn off being a drop target.  
gridControl1.Model.Options.DragDropDropTargetFlags =  
GridDragDropFlags.Disabled  
  
' Turn on accepting text.  
gridControl1.Model.Options.DragDropDropTargetFlags =  
GridDragDropFlags.Text  
  
' Accept both text and styles.  
gridControl1.Model.Options.DragDropDropTargetFlags =  
GridDragDropFlags.Text Or GridDragDropFlags.Text
```

#### 5.4.10 How to convert the contents of a GridControl or GridDataBoundGridControl to Excel

The Contents of the GridControl and GridDataBoundGrid can be transferred to Excel by using the GridToExcel method of the GridExcelConverter class. Here is the code snippet.

**[C#]**

```
Syncfusion.GridExcelConverter.GridExcelConverterControl gecc = new  
Syncfusion.GridExcelConverter.GridExcelConverterControl();  
gecc.GridToExcel(this.gridControl1.Model, @"C:\MyGC.xls");
```

**[VB.NET]**

```
Dim gecc As New Syncfusion.GridExcelConverter.GridExcelConverterControl  
gecc.GridToExcel(Me.gridControl1.Model, "C:\MyGC.xls")
```

The following dll files should be added, along with the default dll files in the reference folder:  
Syncfusion.GridConverter.Base and Syncfusion.XlsIO.Base.

## 5.4.11 How to Disable Clipboard Cut / Copy / Paste in a Grid

### Introduction

If you want to conditionally disable support, handle the **ClipboardCanCut / ClipboardCanCopy / ClipboardCanPaste** events and cancel the events by setting the e.Handled to True and e.Result to False under the desired conditions.

### Example

To completely turn off support, set this property.

#### [C#]

```
// Completely turn off Clipboard cut/copy/paste.  
this.grid.CutPaste.ClipboardFlags = GridDragDropFlags.Disabled;
```

#### [VB .NET]

```
' Completely turn off Clipboard cut/copy/paste.  
Me.grid.CutPaste.ClipboardFlags = GridDragDropFlags.Disabled
```

Here are some code samples.

#### [C#]

```
// Subscribe to the event.  
this.gridControl1.ClipboardCanPaste += new  
GridCutPasteEventHandler(this.gridControl1_ClipboardCanPaste);  
  
// Handle ClipBoardCanPaste event to disable Paste operation.  
private void gridControl1_ClipboardCanPaste(object sender,  
GridCutPasteEventArgs e)  
{  
    if(someCondition)  
    {  
        e.Handled = true;  
  
        // Set Result Property to False to disable the process.  
        e.Result = false;  
    }  
}
```

#### [VB .NET]

```
' Handle ClipBoardCanPaste event to disable Paste operation.  
Private Sub gridControl1_ClipboardCanPaste(sender As Object, _e As  
GridCutPasteEventArgs) _Handles GridControl1.ClipboardCanPaste  
    If someCondition Then  
        e.Handled = True  
  
        ' Set Result Property to False to disable the process.  
        e.Result = False  
    End If  
  
' GridControl1_ClipboardCanPaste  
End Sub
```

#### 5.4.12 How to Display Placeholder Characters when Cell Content Exceeds Cell Width

If the content of a grid cell exceeds the cell width, then characters can be displayed to act as placeholders instead of the actual letters or numbers, indicating that the cell contains more content than can be shown within the cell.

By default, the number sign (#) is used as the placeholder character, but you can specify custom characters.

You can enable the following types of cell content to be converted to placeholder characters when content exceeds cell size:

- Alphabetic
- Numeric
- Both alphabetic and numeric
- None

**Note:** When set to None, the original content will be displayed.

	A	B	C	D
1	100300300	#####	row0 col2	row0 c
2	103950919	#####	row1 col2	row1 c
3	107689062	#####	row2 col2	row2 c
4	111515765	#####	row3 col2	row3 c
5	115432064	#####	row4 col2	row4 c
6	119438998	row5 c	row5 col2	row5 c
7	123537601	row6 c	row6 col2	row6 c
8	127728912	row7 c	row7 col2	row7 c
9	132013967	row8 c	row8 col2	row8 c
10	136393802	row9 c	row9 col2	row9 c
11	140869456	row10	row10 col2	row10
12	145441963	row11	row11 col2	row11

*Figure 493: Cells with Placeholder Characters*

### Converting Alphabetic Content

Use the following code to specify that only alphabetic content will be converted to placeholder characters when content exceeds a cell.

#### [C#]

```
this.gridcontrol1[1,1].AutoFit=AutoFitOptions.Alphabet;
```

#### [VB]

```
Me.gridcontrol1(1,1).AutoFit=AutoFitOptions.Alphabet
```

### Converting Numeric Content

Use the following code to specify that only numeric content will be converted to placeholder characters when content exceeds a cell.

#### [C#]

```
this.gridcontrol1[1,1].AutoFit=AutoFitOptions.Numeric
```

#### [VB]

```
Me.gridcontrol1(1,1).AutoFit=AutoFitOptions.Numeric
```

### Converting Alphabetic and Numeric Content

The following code specifies that both alphabetic and numeric content will be converted to placeholder characters. To do so, set **AutoFitOptions** to **Both**.

#### [C#]

```
this.gridcontrol1[1,1].AutoFit=AutoFitOptions.Both
```

#### [VB]

```
Me.gridcontrol1(1,1).AutoFit=AutoFitOptions.Both
```

### Not Converting Content

Use the following code to display the original cell content without converting anything to placeholder characters. This is performed by setting **AutoFitOptions** to **None**.

#### [C#]

```
this.gridcontrol1[1,1].AutoFit=AutoFitOptions.None
```

#### [VB]

```
Me.gridcontrol1(1,1).AutoFit=AutoFitOptions.None
```

**Note:** By default, **AutoFitOptions** is set to **None**.

### Setting Custom Characters

Characters other than the number sign (#), the default, can be used to replace alphanumeric content that exceeds the size of a cell. To do so, set **AutoFitChar** to the desired character. For example, in the code samples below, the placeholder character has been changed to the *at* (@) character.

**[C#]**

```
this.gridcontrol1[1,1]. AutoFitChar='@'
```

**[VB]**

```
Me.gridcontrol1[1,1]. AutoFitChar='@'
```

	A	B	C	D
1	#####	!!!!!!!	row0 col2	row0 col3
2	!!!!!!!	#####	row1 col2	row1 col3
3	^^^^^	#####	row2 col2	row2 col3
4	\$\$\$\$\$	#####	row3 col2	row3 col3
5	@@@@	#####	row4 col2	row4 col3
6	1194389	row5 col	row5 col2	row5 col3
7	1235376	row6 col	row6 col2	row6 col3
8	1277289	row7 col	row7 col2	row7 col3
9	1320139	row8 col	row8 col2	row8 col3
10	1363938	row9 col	row9 col2	row9 col3
11	1408694	row10 c	row10 col2	row10 col3
12	1454419	row11 c	row11 col2	row11 col3
13	1501123	row12 c	row12 col2	row12 col3

*Figure 494: Custom Placeholder Characters*

### Sample Link

The following sample demonstrates how to use placeholder characters for content exceeding a cell's dimensions.

<Installed location of Essential Studio>\<Essential Studio Version Number>\Windows\Grid.Windows\Samples\2.0\Grid Layout\Resize To Fit Demo

### 5.4.13 How Does the Helper class Support Percentage Sizing in GridControl / GridDataBoundGrid

## Introduction

The example given below has a helper class which, supports the automatic sizing in both a **GridControl** or a **GridDataBoundGrid** in two ways. One way which, is referred to as proportional sizing, is where all the columns in the grid are equally sized. The other technique which, is used is referred to as the percentage sizing allows you to specify that certain columns should occupy certain percentages of the available space for these columns.

The **WireGrid** and **UnwireGrid** are methods that are used to enable the sizing of the helper class. Proportional sizing is achieved by using the **QueryColWidth** event. Percentage sizing will explicitly set the **ColWidth[index]** property for each column.

## Example

[C#]

```
private void Form1_Load(object sender, System.EventArgs e)
{
    DataTable dt = new DataTable();
    for(int i=0; i<10; i++)
    {
        dt.Columns.Add();
        dt.Rows.Add(new object[] { });
    }
    this.gridDataBoundGrid1.DataSource = dt;

    // There is a condition check in accepting
hashtable values.
    // ie.. the total of values passed must not be
greater than (100 - Unmodified Column count)
    ht = new Hashtable();
    ht.Add(1,20);
    ht.Add(9,30);
    ht.Add(4,13);

    // Attempts like using unwanted keys are not
encouraged because count is being used.
    // ht.Add(12,100);
    Helper1 = new GridColSizingHelper();
    Helper2 = new GridColSizingHelper();
    Helper2.WireGrid(this.gridControl1,ht);
    this.gridDataBoundGrid1.AllowResizeToFit =
false;
    Helper1.WireGrid(this.gridDataBoundGrid1,ht);
}
```

```
        private void buttonAdv1_Click(object sender,
System.EventArgs e)
    {
        this.Helper1.UnwireGrid();
        this.Helper2.UnwireGrid();
    }

        private void Form1_Resize(object sender,
System.EventArgs e)
    {
        this.Helper1.TurnOffProportion();
        this.Helper1.TurnOnPercentages();
        this.Helper2.TurnOffProportion();
        this.Helper2.TurnOnPercentages();
    }

        private void buttonAdv2_Click(object sender,
System.EventArgs e)
    {
        Helper2.WireGrid(this.gridControl1);
        Helper1.WireGrid(this.gridDataBoundGrid1);
    }

        private void buttonAdv3_Click(object sender,
System.EventArgs e)
    {
        Helper2.WireGrid(this.gridControl1, ht);
        Helper1.WireGrid(this.gridDataBoundGrid1, ht);
    }
}
```

**[VB.NET]**

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    Dim dt As DataTable = New DataTable()
    For i As Integer = 0 To 9
        dt.Columns.Add()
        dt.Rows.Add(New Object() {})
    Next i
    Me.gridDataBoundGrid1.DataSource = dt
```

```
' There is a condition check in accepting hashtable values.  
' ie.. the total of values passed must not be greater than (100 -  
Unmodified Column count)  
ht = New Hashtable()  
ht.Add(1,20)  
ht.Add(9,30)  
ht.Add(4,13)  
  
' Attempts like using unwanted keys are not encouraged because  
count is being used.  
' ht.Add(12,100);  
Helper1 = New GridColSizingHelper()  
Helper2 = New GridColSizingHelper()  
Helper2.WireGrid(Me.gridControl1,ht)  
Me.gridDataBoundGrid1.AllowResizeToFit = False  
Helper1.WireGrid(Me.gridDataBoundGrid1,ht)  
End Sub  
  
Private Sub buttonAdv1_Click(ByVal sender As Object, ByVal e As  
System.EventArgs) Handles buttonAdv1.Click  
    Me.Helper1.UnwireGrid()  
    Me.Helper2.UnwireGrid()  
End Sub  
  
Private Sub Form1_Resize(ByVal sender As Object, ByVal e As  
System.EventArgs) Handles MyBase.Resize  
    If Not Helper1 Is Nothing Then  
        Me.Helper1.TurnOffProportion()  
        Me.Helper1.TurnOnPercentages()  
        Me.Helper2.TurnOffProportion()  
        Me.Helper2.TurnOnPercentages()  
    End If  
End Sub  
  
Private Sub buttonAdv2_Click(ByVal sender As Object, ByVal e As  
System.EventArgs) Handles buttonAdv2.Click  
    Helper2.WireGrid(Me.gridControl1)  
    Helper1.WireGrid(Me.gridDataBoundGrid1)  
End Sub  
  
Private Sub buttonAdv3_Click(ByVal sender As Object, ByVal e As  
System.EventArgs) Handles buttonAdv3.Click  
    Helper2.WireGrid(Me.gridControl1, ht)  
    Helper1.WireGrid(Me.gridDataBoundGrid1, ht)  
End Sub  
End Class
```

```
End Namespace
```

#### 5.4.14 How to export CellCommentTips to Excel using GridExcelConverterControl

You can achieve this by handling the **QueryImportExportCellInfo** event handler. In the event, check the GridExcelTipStyleProperties for ExcelTip properties, and accordingly add the comment to the IRange.

**[C#]**

```
Syncfusion.GridExcelConverter.GridExcelConverterControl gecc = new Syncfusion.GridExcelConverter.GridExcelConverterControl();
gecc.QueryImportExportCellInfo += new Syncfusion.GridExcelConverter.GridImportExportCellInfoEventHandler(gecc_QueryImportExportCellInfo);

void gecc_QueryImportExportCellInfo(object sender,
Syncfusion.GridExcelConverter.GridImportExportCellInfoEventArgs e)
{
    ExcelTip.GridExcelTipStyleProperties style = new ExcelTip.GridExcelTipStyleProperties(e.GridCell);
    if (style.HasExcelTipText)
        e.ExcelCell.AddComment().Text = style.ExcelTipText;
}
```

**[VB.NET]**

```
Dim gecc As New Syncfusion.GridExcelConverter.GridExcelConverterControl()
AddHandler gecc.QueryImportExportCellInfo, AddressOf gecc_QueryImportExportCellInfo

Private Sub gecc_QueryImportExportCellInfo(ByVal sender As Object,
ByVal e As Syncfusion.GridExcelConverter.GridImportExportCellInfoEventArgs)
    Dim style As New ExcelTipDLL.GridExcelTipStyleProperties(e.GridCell)
    If style.HasExcelTipText Then
        e.ExcelCell.AddComment().Text = style.ExcelTipText
    End If
End Sub 'gecc_QueryImportExportCellInfo
```

## 5.4.15 How to Get the Selected Ranges in a Grid

### Introduction

The `grid.Selections.Ranges` is a **GridRangeInfoList** object which, holds the currently selected ranges.

### Example

You can iterate through this list retrieving `GridRangeInfo` objects with code such as.

#### [C#]

```
// Iterate through the SelectionRanges, to display every range in the
list.
foreach(GridRangeInfo range in this.grid.Selections.Ranges)
{
    MessageBox.Show(range.ToString());
}
```

#### [VB .NET]

```
Dim range As GridRangeInfo

' Iterate through the SelectionRanges, to display every range in the
list.
For Each range In Me.grid.Selections.Ranges
    MessageBox.Show(range.ToString())
Next
```

In the previous code, note that the `GridRangeInfo` object is not really being used for anything. When you actually try to use it, you will need to take into account the fact that you cannot make any assumptions regarding whether this range object is a row range or a cell range or a column range or something else. For example, you may want to make use of `Range.Top` and `Range.Bottom` to get the top and bottom rows that have been selected. But, if the range happens to be a column range, then these properties will not be valid as column ranges.

So, before using such range properties, you must check whether these properties have been properly set. The `GridRangeInfo` class has a method that can manage this task:

**GridRangeInfo.ExpandRange**. If you need to access properties like left, right, top and bottom in a `GridRangeInfo` where it is unknown whether or not the range is a cell range, then you should first call **ExpandRange** to make sure that this is the case.

**[C#]**

```
int rowLimit = grid.Model.RowCount;
    int colLimit = grid.Model.ColCount;

    // Call ExpandRange method and display the top and bottom rows in
    each range.
    foreach(GridRangeInfo range in this.grid.Selections.Ranges)
    {
        GridRangeInfo range1 = range.ExpandRange(1, 1, rowLimit,
    colLimit);
        MessageBox.Show("top={0} bot={1}", range1.Top, range1.Bottom);
    }
```

**[VB.NET]**

```
Dim rowLimit As Integer = grid.Model.RowCount
    Dim colLimit As Integer = grid.Model.ColCount
    Dim range As GridRangeInfo

    ' Call ExpandRange method and display the top and bottom rows
    in each range.
    For Each range In Me.grid.Selections.Ranges
        Dim range1 As GridRangeInfo = range.ExpandRange(1, 1, rowLimit,
    colLimit)
        MessageBox.Show("top={0} bot={1}", range1.Top,
range1.Bottom)
    Next range
```

To retrieve the active range, use the below given code snippet.

**[C#]**

```
// Retrieve the active range.
GridRangeInfo activeRange = grid.Selections.Ranges.ActiveRange;
```

**[VB.NET]**

```
' Retrieve the active range.
Dim activeRange As GridRangeInfo = grid.Selections.Ranges.ActiveRange
```

## 5.4.16 How to Get the Cell Coordinates Under a Given Point

### Introduction

If the point is given as part of one of the grids mouse event arguments, the e.X and e.Y members of the event args should give the point in the grids coordinates. If the point is obtained in some other manner, you will have to first change it to the grid's coordinates. Once you have the point in grid coordinates, you can call the **Grid.PointToRowCol** method to get the row and column under the point.

### Example

#### [C#]

```
// In a mouse event you might have code such as this to get the point.
Point pt = new Point(e.X, e.Y);

// In other situations, you could use the static Cursor.Position method
// to get the current mouse point in screen coordinates.
Point pt = this.gridControl1.PointToClient(Cursor.Position);

// Then get the row and col.
int row, col;
this.grid.PointToRowCol(pt, out row, out col);

// From the row, col, you can get the cell rectangle.
Rectangle cellRect =
this.grid.RangeInfoToRectangle(GridRangeInfo.Cell(row, col));
```

#### [VB.NET]

```
' In a mouse event you might have code such as this to get the point.
Dim pt As New Point(e.X, e.Y)

' In other situations, you could use the static Cursor.Position method
' to get the current mouse point in screen coordinates.
Dim pt As Point = Me.GridControl1.PointToClient(Cursor.Position)

' Then get the row and col.
Dim row, col As Integer
Me.grid.PointToRowCol(pt, row, col)

' From the row, col, you can get the cell rectangle.
Dim cellRect As Rectangle =
Me.grid.RangeInfoToRectangle(GridRangeInfo.Cell(row, col))
```

## 5.4.17 How to Get the Top / Bottom / Left / Right Viewable Row and Column Indexes

### Introduction

Use the following variables to get the viewable row and column indexes.

### Example

#### [C#]

```
// Top Row Index  
this.grid.TopRowIndex  
  
// Left Column Index  
this.grid.LeftColIndex  
  
// Bottom Row Index  
this.grid.ViewLayout.LastVisibleRow  
  
// Right Column Index  
this.grid.ViewLayout.LastVisibleCol
```

#### [VB .NET]

```
' Top Row Index  
Me.grid.TopRowIndex  
  
' Left Column Index  
Me.grid.LeftColIndex  
  
' Bottom Row Index  
Me.grid.ViewLayout.LastVisibleRow  
  
' Right Column Index  
Me.grid.ViewLayout.LastVisibleCol
```

## 5.4.18 How to Get the Screen Point for the Given Cell Coordinates

## Introduction

You can get the cell's rectangle in grid coordinates from the **RangeInfoToRectangle** method. Then with the rectangle's coordinates, you can get the screen point using the **PointToScreen** method.

## Example

### [C#]

```
// For a given row and col.  
Rectangle rect = this.grid.RangeInfoToRectangle(GridRangeInfo.Cell(row,  
col));  
Point screenPoint = this.grid.PointToScreen(new Point(rect.Left,  
rect.Top));
```

### [VB .NET]

```
' For a given row and col.  
Dim rect As Rectangle =  
Me.grid.RangeInfoToRectangle(GridRangeInfo.Cell(row, col))  
Dim screenPoint As Point = Me.grid.PointToScreen(New Point(rect.Left,  
rect.Top))
```

## 5.4.19 How to Get the New Value and the Old Value when an Item is Selected in a Combobox Cell

### Introduction

The **CurrentCellCloseDropDown** event gets triggered when a dropdown is closed in a grid cell. The new value of the **ComboBox** can be obtained from the CurrentCell's **Renderer** property and the old value can be obtained from the grid.

## Example

### [C#]

```
GridCurrentCell cc;  
  
// Handle CurrentCellCloseDropDown.  
private void gridDataBoundGrid1_CurrentCellCloseDropDown(object sender,  
Syncfusion.Windows.Forms.PopupClosedEventArgs e)
```

```
{  
    cc= this.gridDataBoundGrid1.CurrentCell;  
    Console.WriteLine(e.PopupCloseType.ToString());  
  
    // Use Renderer.GetCellValue() to retrieve the new cell  
    value.  
    Console.WriteLine("New Value  
{0}",cc.Renderer.GetCellValue());  
  
    // Retrieve the oldvalue.  
    Console.WriteLine("Old Value  
{0}",this.gridDataBoundGrid1[cc.RowIndex,cc.ColIndex].CellValue.ToString());  
}  
  
// Handle CurrentCellCloseDropDown.  
private void gridControl1_CurrentCellCloseDropDown(object sender,  
Syncfusion.Windows.Forms.PopupClosedEventArgs e)  
{  
    cc= this.gridControl1.CurrentCell;  
    Console.WriteLine(e.PopupCloseType.ToString());  
  
    // Use Renderer.GetCellValue() to retrieve the new cell value.  
    Console.WriteLine("New Value {0}",cc.Renderer.GetCellValue());  
  
    // Retrieve the oldvalue.  
    Console.WriteLine("Old Value  
{0}",this.gridControl1[cc.RowIndex,cc.ColIndex].CellValue.ToString());  
}
```

**[VB.NET]**

```
Private cc As GridCurrentCell  
  
' Handle CurrentCellCloseDropDown.  
Private Sub gridDataBoundGrid1_CurrentCellCloseDropDown(ByVal sender As  
Object, ByVal e As Syncfusion.Windows.Forms.PopupClosedEventArgs)  
    cc= Me.gridDataBoundGrid1.CurrentCell  
    Console.WriteLine(e.PopupCloseType.ToString())  
  
    ' Use Renderer.GetCellValue() to retrieve the new cell value.  
    Console.WriteLine("New Value {0}",cc.Renderer.GetCellValue())  
  
    ' Retrieve the oldvalue.  
    Console.WriteLine("Old  
Value{0}",Me.gridDataBoundGrid1(cc.RowIndex,cc.ColIndex).CellValue.  
ToString())
```

```
End Sub

' Handle CurrentCellCloseDropDown.
Private Sub gridControl1_CurrentCellCloseDropDown(ByVal sender As Object, ByVal e As Syncfusion.Windows.Forms.PopupClosedEventArgs)
    cc= Me.gridControl1.CurrentCell
    Console.WriteLine(e.PopupCloseType.ToString())

    ' Use Renderer.GetCellValue() to retrieve the new cell value.
    Console.WriteLine("New Value {0}",cc.Renderer.GetCellValue())

    ' Retrieve the oldvalue.
    Console.WriteLine("Old Value
{0}",Me.gridControl1(cc.RowIndex,cc.ColIndex).CellValue.ToString())
End Sub
```

## 5.4.20 How to Have Character Casing Settings for a Cell

### Introduction

CharacterCasing works only with the CellType = "OriginalTextBox" which, uses a control that is derived from System.Windows.Forms.TextBox. The **celltype** text box is derived from the **RichTextBox** which, does not have a **CharacterCasing** property. To enable UpperCasing for the whole grid, set the properties in the **TableStyle**. To enable CharacterCasing on a column, row or cell basis, set the style properties using the techniques that are appropriate for the grid that you are using as discussed in the topics on changing **backcolor**.

### Example

#### [C#]

```
// Enable UpperCasing for the whole grid.
this.grid.TableStyle.CellType = "OriginalTextBox";
this.grid.TableStyle.CharacterCasing = CharacterCasing.Upper;
```

#### [VB .NET]

```
' Enable UpperCasing for the whole grid.
Me.grid.TableStyle.CellType = "OriginalTextBox"
Me.grid.TableStyle.CharacterCasing = CharacterCasing.Upper
```

## 5.4.21 How to Hide a Row / Column

### Introduction

You can hide rows and columns using the **grid.Model.Rows.Hidden** collection and the **grid.Model.Cols.Hidden** collection. You can hide or show a range of columns / rows using the **grid.Model.HideCols.SetRange** / **grid.Model.HideRows.SetRange**.

### Example

#### [C#]

```
// Hide row 2.  
this.grid.Model.Rows.Hidden[2] = true;  
  
// Hide column 3.  
this.grid.Model.Cols.Hiddent[3] = true;  
  
// Hide cols 1-3.  
this.grid.Model.HideCols.SetRange(1, 3, true);
```

#### [VB .NET]

```
' Hide row 2.  
Me.grid.Model.Rows.Hidden(2) = True  
  
' Hide column 3.  
Me.grid.Model.Cols.Hiddent(3) = True  
  
' Hide cols 1-3.  
Me.grid.Model.HideCols.SetRange(1, 3, True)
```

## 5.4.22 How to Make a Cell Display '...' if it is Not Wide Enough

### Introduction

You must set the **GridStyleInfo's Trimming** property to achieve this. To enable **trimming** for the whole grid, set this property in the **TableStyle**. To enable trimming on a column, row or cell basis, set this style property using the techniques that are appropriate for the grid that you are using as discussed in the topics on changing **backcolor**.

## Example

### [C#]

```
// Set Ellipsis Text for the whole grid.  
this.grid.TableStyle.Trimming = StringTrimming.EllipsisWord;  
this.grid.TableStyle.Trimming = StringTrimming.EllipsisWord;
```

### [VB .NET]

```
' Set Ellipsis Text for the whole grid.  
Me.grid.TableStyle.Trimming = StringTrimming.EllipsisWord  
Me.grid.TableStyle.Trimming = StringTrimming.EllipsisWord
```

## 5.4.23 How to Make the Grid Behave Like a List Box

### Introduction

You need to do two things to make the grid work like a list box. You must set the **grid.ListBoxSelectionMode** property to make the grid select the whole row when you click a cell or move the current cell with arrow keys. This will highlight the entire row except for the current cell. If your grid is to be non-editable, you must highlight the current cell as well as handle the **CurrentCellActivating** event.

## Example

### [C#]

```
// In form.Load  
this.gridControl1.ListBoxSelectionMode = SelectionMode.One;  
this.gridControl1.CurrentCellActivating += new  
GridCurrentCellActivatingEventHandler(grid_CurrentCellActivating);  
  
// The handler.  
private void grid_CurrentCellActivating(object sender,  
GridCurrentCellActivatingEventArgs e)  
{  
    e.RowIndex = 0;  
}
```

### [VB .NET]

```
' In form.Load
Me.GridControl1.ListBoxSelectionMode = SelectionMode.One
AddHandler Me.GridControl1.CurrentCellActivating, AddressOf
grid_CurrentCellActivating

' The handler.
Private Sub grid_CurrentCellActivating(ByVal sender As Object, ByVal e
As GridCurrentCellActivatingEventArgs)
    e.ColumnIndex = 0

' Grid_CurrentCellActivating
End Sub
```

#### 5.4.24 How to make resizing possible in additional row headers of a GridControl and GridDataBoundGrid

The resizing of additional row headers on the grid can be made possible by setting the ResizeColsBehaviour flag to Grid.GridResizeCellsBehavior.InsideGrid.

##### [C#]

```
grid.ResizeColsBehavior =
(((Syncfusion.Windows.Forms.Grid.GridResizeCellsBehavior.ResizeSingle
| Syncfusion.Windows.Forms.Grid.GridResizeCellsBehavior.InsideGrid)
| Syncfusion.Windows.Forms.Grid.GridResizeCellsBehavior.OutlineHeaders)
| Syncfusion.Windows.Forms.Grid.GridResizeCellsBehavior.OutlineBounds);
```

##### [VB .NET]

```
grid.ResizeColsBehavior =
(((Syncfusion.Windows.Forms.Grid.GridResizeCellsBehavior.ResizeSingle
| Syncfusion.Windows.Forms.Grid.GridResizeCellsBehavior.InsideGrid)
| Syncfusion.Windows.Forms.Grid.GridResizeCellsBehavior.OutlineHeaders)
Dim
Syncfusion.Windows.Forms.Grid.GridResizeCellsBehavior.OutlineBounds) As
|
```

## 5.4.25 How to Move to the Next Row from the Last Cell of a Row

### Introduction

Set the **WrapCellBehavior** property to wrap a row when the Tab or Enter key is pressed.

### Example

#### [C#]

```
// Set WrapCellBehaviour property to Wrap Row to move to the next row.  
this.grid.Model.Options.WrapCellBehavior =  
GridWrapCellBehavior.WrapRow;
```

#### [VB .NET]

```
' Set WrapCellBehaviour property to Wrap Row to move to the next row.  
Me.grid.Model.Options.WrapCellBehavior = GridWrapCellBehavior.WrapRow
```

## 5.4.26 How to Prevent Resizing a Specific Column in a GridControl

### Introduction

Handle the **ResizingColumns** event and cancel the resizing for specific columns.

### Example

#### [C#]

```
// Handle ResizingColumns event.  
private void grid_ResizingColumns(object sender,  
GridResizingColumnsEventArgs e)  
{  
    // Disable Column Resizing for the third column from the Right.  
    if(e.Columns.Right == 2)  
    {  
        e.Cancel = true;  
    }  
}
```

**[VB.NET]**

```
' Handle ResizingColumns event.  
Private Sub grid_ResizingColumns(ByVal sender As Object, ByVal e As  
GridResizingColumnsEventArgs)  
  
    // Disable Column Resizing for the third column from the Right.  
    If e.Columns.Right = 2 Then  
        e.Cancel = True  
    End If  
End Sub
```

## 5.4.27 How to Print a Grid

### Introduction

The **GridPrintDocument** class will allow you to [print](#) your grids. Here is a button click event handler that will show you how to use this class.

### Example

**[C#]**

```
private void menuItem13_Click(object sender, System.EventArgs e)  
{  
    if (this.grid != null)  
    {  
        try  
        {  
            // Create a print document for the grid.  
            GridPrintDocument pd = new  
GridPrintDocument(this.grid);  
  
            // Create Print Dialog.  
            PrintDialog dlg = new PrintDialog();  
  
            // Assign the print document to the dialog object.  
            dlg.Document = pd;  
  
            // Apply Dialog Settings.  
            dlg.AllowSelection = true;  
            dlg.AllowSomePages = true;
```

```
        DialogResult result = dlg.ShowDialog();
        if (result == DialogResult.OK)
        {
            // Print the Grid Document.
            pd.Print();
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("An error occurred - " + ex.Message);
    }
}
}
```

**[VB.NET]**

```
Private Sub menuItem13_Click(ByVal sender As Object, ByVal e As
System.EventArgs)
    If Not (Me.grid Is Nothing) Then
        Try

            ' Create a print document for the grid.
            Dim pd As New GridPrintDocument(Me.grid)

            ' Create Print Dialog.
            Dim dlg As New PrintDialog

            ' Assign the print document to the dialog object.
            dlg.Document = pd

            ' Apply Dialog Settings.
            dlg.AllowSelection = True
            dlg.AllowSomePages = True
            Dim result As DialogResult = dlg.ShowDialog()
            If result = DialogResult.OK Then

                ' Print the Grid Document.
                pd.Print()
            End If
        Catch ex As Exception
            MessageBox.Show(("An error occurred - " +
ex.Message))
        End Try
    End If
' MenuItem13_Click
End Sub
```

## 5.4.28 How to Print Preview a Grid

### Introduction

The **GridPrintDocument** class will allow you to [print preview](#) your grids. Here is a button click event handler that will show you how to use this class.

### Example

[C#]

```
private void menuItem13_Click(object sender, System.EventArgs e)
{
    if (this.grid != null)
    {
        try
        {
            GridPrintDocument pd = new
GridPrintDocument(this.grid, true);

            // Create a PrintPreviewDialog object.
            PrintPreviewDialog dlg = new PrintPreviewDialog();

            // Initialize it with the print document to be
previewed.
            dlg.Document = pd;

            // Display the preview dialog.
            dlg.ShowDialog();
        }
        catch (Exception ex)
        {
            MessageBox.Show("An error occurred - " + ex.Message);
        }
    }
}
```

[VB.NET]

```
Private Sub menuItem13_Click(ByVal sender As Object, ByVal e As
System.EventArgs)
    If Not (Me.grid Is Nothing) Then
```

```
Try
    Dim pd As New GridPrintDocument(Me.grid True)

        ' Create a PrintPreviewDialog object.
    Dim dlg As New PrintPreviewDialog

        ' Initialize it with the print document to be
previewed.
    dlg.Document = pd

        ' Display the preview dialog.
    dlg.ShowDialog()
Catch ex As Exception
    MessageBox.Show(("An error occurred - " + ex.Message))
End Try
End If

' MenuItem13_Click
End Sub
```

## 5.4.29 How to Put a CheckBox in a Header Cell in a GridControl or GridDataBoundGrid

### Introduction

To place a CheckBox in a **GridControl** you must set the **CellType** to CheckBox and assign string values for the **CheckedValue** and the **UncheckedValue** in the **CheckBoxOptions** property. The value of the CheckBox will be stored to a particular cell in the GridControl. When the CheckBox is clicked the **CheckBoxClick** event gets triggered.

To place a CheckBox in the GridDataBoundGrid, 2 events have to be implemented: the **QueryCellInfo** event which, is used to set the style properties and the **SaveCellInfo** event which, is used to save the cell's value. The value of the CheckBox cannot be stored in the GridDataBoundGrid so, any datatype / collection can be used to store the value. The **CheckBoxClick** event gets triggered when the CheckBox is clicked.

### Example

```
[C#]

private string CheckBoxValue;
```

```
private void Model_QueryCellInfo(object sender,
GridQueryCellInfoEventArgs e)
{
    // Check for a row header.
    if(e.ColumnIndex > 0 && e.RowIndex == 0)
    {
        int colIndex1 =
this.gridDataBoundGrid1.Binder.NameToColIndex("Column2");
        if(colIndex1 == e.ColumnIndex)
        {
            e.Style.Description = "Check";

            // Display the CellValue.
            e.Style.CellValue = CheckBoxValue;
            e.Style.CellValueType = typeof(string);

            // Determine CheckBoxOptions Values.
            e.Style.CheckBoxOptions = new
GridCheckBoxCellInfo(true.ToString(), false.ToString(), "", true);

            // Set up a CheckBox control in the header.
            e.Style.CellType = "CheckBox";
            e.Style.CellAppearance = GridCellAppearance.Raised;
            e.Style.Enabled = true;
        }
    }
}

private void Model_SaveCellInfo(object sender,
GridSaveCellInfoEventArgs e)
{
    if(e.ColumnIndex > 0 && e.RowIndex == 0)
    {
        int colIndex1 =
this.gridDataBoundGrid1.Binder.NameToColIndex("Column2");
        if(colIndex1 == e.ColumnIndex)
        {
            // Save the changes that are made in the Cell Value.
            if(e.StyleCellValue != null)
                CheckBoxValue = (string)e.StyleCellValue;
        }
    }
}
```

**[VB.NET]**

```
Private CheckBoxValue As Boolean = False
```

```
Private Sub Model_QueryCellInfo(ByVal sender As Object, ByVal e As GridQueryCellInfoEventArgs)

    ' Check for a row header.
    If e.ColumnIndex > 0 AndAlso e.RowIndex = 0 Then
        Dim colIndex1 As Integer =
Me.gridDataBoundGrid1.Binder.NameToColIndex("Column2")
        If colIndex1 = e.ColumnIndex Then
            e.Style.Description = "Check"

            ' Display the CellValue.
            e.Style.CellValue = CheckBoxValue
            e.Style.CellValueType = GetType(Boolean)

            ' Determine CheckBoxOptions Values.
            e.Style.CheckBoxOptions = New
GridCheckBoxCellInfo(True.ToString(), False.ToString(), "", True)

            ' Set up a CheckBox control in the header.
            e.Style.CellType = "CheckBox"
            e.Style.CellAppearance = GridCellAppearance.Raised
            e.Style.Enabled = True
        End If
    End If
End Sub

Private Sub Model_SaveCellInfo(ByVal sender As Object, ByVal e As GridSaveCellInfoEventArgs)
    If e.ColumnIndex > 0 AndAlso e.RowIndex = 0 Then
        Dim colIndex1 As Integer =
Me.gridDataBoundGrid1.Binder.NameToColIndex("Column2")
        If colIndex1 = e.ColumnIndex Then

            ' Save the changes that are made in the Cell Value.
            If Not e.StyleCellValue Is Nothing Then
                CheckBoxValue = CBool(e.StyleCellValue)
            End If
        End If
    End If
End Sub
```

#### 5.4.30 How to Put a Cell in Overstrike Mode so Characters get Replaced instead of Inserted as you type

## Introduction

This can be achieved by using the **TextBox.SelectionLength** property of the **GridTextBoxCellRenderer** in the **CurrentCellKeyPress** event. The idea to achieve this is by selecting one character when a key is pressed, and also neglecting the Backspace key.

## Example

### [C#]

```
private void gridControl1_CurrentCellKeyPress(object sender,
    KeyPressEventArgs e)
{
    GridTextBoxCellRenderer cr =
this.gridControl1.CurrentCell.Renderer as GridTextBoxCellRenderer;

    if(e.KeyChar != Convert.ToChar(Keys.Back)
        && cr.TextBox.SelectionLength == 0)
    {

        // Programmatically selecting One char.
        cr.TextBox.SelectionLength = 1;
    }
}

private void gridControl1_CurrentCellKeyDown(object sender,
    System.Windows.Forms.KeyEventArgs e)
{
    if(e.KeyCode == Keys.Back)
    {

        // Translating the Backspace key to a left arrow key.
        SendKeys.Send("{LEFT}");
        e.Handled = true;
    }
}
```

### [VB .NET]

```
Private Sub gridControl1_CurrentCellKeyPress(ByVal sender As Object,
    ByVal e As KeyPressEventArgs)

    Dim cr As GridTextBoxCellRenderer = CType(IIf(TypeOf
Me.gridControl1.CurrentCell.Renderer Is GridTextBoxCellRenderer,
Me.gridControl1.CurrentCell.Renderer, Nothing),
GridTextBoxCellRenderer)
```

```
' Programmatically selecting One char.  
If e.KeyChar <> Convert.ToChar(Keys.Back) AndAlso  
cr.TextBox.SelectionLength = 0 Then  
cr.TextBox.SelectionLength = 1  
  
End If  
  
End Sub  
  
Private Sub gridControl1_CurrentCellKeyDown(ByVal sender As Object,  
 ByVal e As System.Windows.Forms.KeyEventArgs)  
 If e.KeyCode = Keys.Back Then  
  
     ' Translating the Backspace key to a left arrow key.  
     SendKeys.Send("{LEFT}")  
     e.Handled = True  
 End If  
End Sub
```

### 5.4.31 How to Put a ComboBox in a Header Cell in a GridControl or GridDataBoundGrid

#### Introduction

#### GridControl

To place a ComboBox in a header cell of a **GridControl**, you must set the **CellType** of the header to **ComboBox** and assign a valid data source (Choicelist or DataSource).

#### Example

[C#]

```
// Set up a ComboBox control.  
this.gridControl1[0,4].CellType = "ComboBox";  
this.gridControl1[0,4].CellAppearance = GridCellAppearance.Raised;  
this.gridControl1[0,4].ShowButtons =  
GridShowButtons.ShowCurrentCellEditing;  
this.gridControl1[0,4].ChoiceList = items;  
this.gridControl1[0,4].CellValue = "Combo";
```

**[VB.NET]**

```
' Set up a ComboBox control.  
Me.gridControl1(0,4).CellType = "ComboBox"  
Me.gridControl1(0,4).CellAppearance = GridCellAppearance.Raised  
Me.gridControl1(0,4).ShowButtons =  
GridShowButtons.ShowCurrentCellEditing  
Me.gridControl1(0,4).DataSource = items  
Me.gridControl1(0,4).CellValue = "Combo"
```

### **GridDataBoundGrid**

To place a ComboBox in a header cell of a GridDataBoundGrid, you need to handle two events (QueryCellInfo and SaveCellInfo). Set the data source (Choicelist or DataSource) for the header cell in the **QueryCellInfo** event handler and save the edited value back to datasource in the **SaveCellInfo** event handler.

#### **Example**

**[C#]**

```
this.gridDataBoundGrid1.Model.QueryCellInfo += new  
GridQueryCellInfoEventHandler(Model_QueryCellInfo);  
this.gridDataBoundGrid1.Model.SaveCellInfo += new  
GridSaveCellInfoEventHandler(Model_SaveCellInfo);  
String Collection items;  
string choice;  
  
// Handle QueryCellInfo to set datasource for combobox.  
private void Model_QueryCellInfo(object sender,  
GridQueryCellInfoEventArgs e)  
{  
    if(e.ColIndex > 0 && e.RowIndex == 0)  
    {  
        int colIndex2 =  
this.gridDataBoundGrid1.Binder.NameToColIndex("Column4");  
        if(colIndex2 == e.ColIndex)  
        {  
            e.Style.Borders.All = gb;  
            e.Style.BackColor = Color.White;  
            e.Style.CellType = "ComboBox";  
  
            // Set Datasource for combobox.  
            e.Style.ChoiceList = items;  
            e.Style.CellValue = choice;  
            e.Style.CellAppearance = GridCellAppearance.Raised;
```

```
        e.Style.Enabled = true;
    }
}

// Handle SaveCellInfo to save the edited values back to the
datasource.
private void Model_SaveCellInfo(object sender,
GridSaveCellInfoEventArgs e)
{
    if(e.ColIndex > 0 && e.RowIndex == 0)
    {
        int colIndex2 =
this.gridDataBoundGrid1.Binder.NameToColIndex("Column4");
        if(colIndex2 == e.ColIndex)
        {
            if(e.Style.CellValue != null )
                choice = (string)e.Style.CellValue;
        }
    }
}
```

**[VB.NET]**

```
AddHandler gridDataBoundGrid1.Model.QueryCellInfo, AddressOf
Model_QueryCellInfo
AddHandler gridDataBoundGrid1.Model.SaveCellInfo, AddressOf
Model_SaveCellInfo
Private items As StringCollection
Private choice As String

' Handle QueryCellInfo to set datasource for combobox.
Private Sub Model_QueryCellInfo(ByVal sender As Object, ByVal e As
GridQueryCellInfoEventArgs)
    If e.ColIndex > 0 AndAlso e.RowIndex = 0 Then
        Dim colIndex2 As Integer =
Me.gridDataBoundGrid1.Binder.NameToColIndex("Column4")
        If colIndex2 = e.ColIndex Then
            e.Style.Borders.All = gb
            e.Style.BackColor = Color.White
            e.Style.CellType = "ComboBox"

            ' Set Datasource for combobox.
            e.Style.ChoiceList = items
            e.Style.CellValue = choice
            e.Style.CellAppearance = GridCellAppearance.Raised
    End If
End Sub
```

```
        e.Style.Enabled = True
    End If
End If
End Sub

' Handle SaveCellInfo to save the edited values back to the datasource.
Private Sub Model_SaveCellInfo(ByVal sender As Object, ByVal e As
GridSaveCellInfoEventArgs)
    If e.ColumnIndex > 0 AndAlso e.RowIndex = 0 Then
        Dim colIndex2 As Integer =
Me.gridDataBoundGrid1.Binder.NameToColIndex("Column4")
        If colIndex2 = e.ColumnIndex Then
            If Not e.Style.CellValue Is Nothing Then
                choice = CStr(e.Style.CellValue)
            End If
        End If
    End If
End Sub
```

### 5.4.32 How to Set the Background Color for a Grid

#### Introduction

To set the backcolor for the area of the grid populated by cells, you must set the **grid.BackColor** property to the color. The grid display may also have regions where there are no cells. These regions will be the grid's client area where there are no cells or scrollbars.

#### Example

##### [C#]

```
// Blue
grid.BackColor = Color.Blue;

// Orange
grid.Properties.BackgroundColor = Color.Orange;
```

##### [VB .NET]

```
' Blue
grid.BackColor = Color.Blue
```

```
' Orange  
grid.Properties.BackgroundColor = Color.Orange
```



Figure 495: BackColor set to "Blue" and Properties.BackgroundColor set to "Orange"

### 5.4.33 How to Select All Text in a Grid After Clicking a Cell

#### Introduction

The **ActivateCurrentCellBehavior** property controls the activation behavior as a cell becomes current by being clicked or through the cursor keys. If you want the cell text to be fully selected when a cell becomes the current cell, then use the following property.

The following code illustrates how to set the Cell Activation behavior to SelectAll in GridControl:

#### [C#]

```
// Set Cell Activation behavior to 'SelectAll'.  
this.gridControl1ActivateCurrentCellBehavior =  
GridCellActivateAction.SelectAll;
```

#### [VB.NET]

```
' Set Cell Activation behavior to 'SelectAll'.  
Me.GridActivateCurrentCellBehavior = GridCellActivateAction.SelectAll
```

The following code illustrates how to set the Cell Activation behavior to SelectAll in GridGrouping control:

[C#]

```
this.gridGroupingControl1ActivateCurrentCellBehavior =  
GridCellActivateAction.SelectAll;
```



**Note:** Other options range from None (no activation at all) to ClickOnCell, DblClickOnCell or SetCurrent.

### 5.4.34 How to Size Column Widths or Row Heights to Fit the Text

#### Introduction

To size columns so that all the text is visible, use the **grid.Model.ColWidths.ResizeToFit** method. This method will take two arguments, a **GridRangeInfo** object that will specify the cells that are to be resized and a **GridResizeToFitOptions** setting that will specify certain behaviors. The second setting controls whether you'll allow the cell to shrink when it is resized and whether you want to include any header cells in the resizing.

There is also a **grid.Model.RowHeights.ResizeToFit** method to size row heights.

#### Example

[C#]

```
// AutoFit RowHeights.  
grid.Model.RowHeights.ResizeToFit(GridRangeInfo.Table,  
GridResizeToFitOptions.NoShrinkSize);  
  
// AutoFit ColumnWidths.  
grid.Model.ColWidths.ResizeToFit(GridRangeInfo.Col(2),  
GridResizeToFitOptions.NoShrinkSize);
```

[VB .NET]

```
' AutoFit RowHeights.  
grid.Model.RowHeights.ResizeToFit(GridRangeInfo.Table,
```

```
GridResizeToFitOptions.NoShrinkSize)

' AutoFit ColumnWidths.
grid.Model.ColWidths.ResizeToFit(GridRangeInfo.Col(2),
GridResizeToFitOptions.NoShrinkSize)
```



**Note:** Resizing the entire grid for very large grids can be time consuming. To only resize the visible area of the grid, you can set the range argument to `grid.ViewLayout.VisibleCellsRange`.

## 5.4.35 How to Validate Changes Made to a Grid Cell

### Introduction

There are two events that can be used to validate the changes that are made to a grid cell: **CurrentCellValidateString** and **CurrentCellValidating**. The CurrentCellValidateString is fired every time a key is pressed and the current text is passed as part of the **EventArgs**. The CurrentCellValidating is only fired once when the user tries to leave the current cell.

### Example

Here are some code samples that will show how you can get the new value of the cell as well as the old value of the cell in the CurrentCellValidating event.

#### [C#]

```
// Retrieve the old and new values of a cell.
GridCurrentCell cc = this.grid.CurrentCell;
string newValue = cc.Renderer.ControlText;
string oldValue = this.grid[cc.RowIndex, cc.ColIndex].Text;
```

#### [VB .NET]

```
' Retrieve the old and new values of a cell.
Dim cc As GridCurrentCell = Me.GridControl1.CurrentCell
Dim newValue As String = cc.Renderer.ControlText
Dim oldValue As String = Me.GridControl1(cc.RowIndex, cc.ColIndex).Text
```

In either event, if you want the validation to fail, set `e.Cancel = True`. Here is the code that will not allow your user to blank out cells in column 2.

#### [C#]

```
private void grid_CurrentCellValidating(object sender, CancelEventArgs e)
{
    GridCurrentCell cc = this.grid.CurrentCell;

    // Prevent user from blanking out cells in column 2.
    if (cc.ColumnIndex == 2)
    {
        string val = cc.Renderer.ControlText;
        if (val.Length == 0)
        {

            // No empty string in col 2.
            e.Cancel = true;
        }
    }

    // Grid_CurrentCellValidating
}
```

**[VB.NET]**

```
Private Sub grid_CurrentCellValidating(ByVal sender As Object, ByVal e As CancelEventArgs)
    Dim cc As GridCurrentCell = Me.grid.CurrentCell

    ' Prevent user from blanking out cells in column 2.
    If cc.ColumnIndex = 2 Then
        Dim val As String = cc.Renderer.ControlText
        If val.Length = 0 Then

            ' No empty string in col 2.
            e.Cancel = True
        End If
    End If

    ' Grid_CurrentCellValidating
End Sub
```

### 5.4.36 How to Write Syntax for Grid Model Properties

From Essential Studio 9.4.6.20 it is not mandatory to specify the complete network to utilize the GridModel properties.

Table 23: GridModel Properties

Property	Description	Type	Data Type	Reference links
ShowColumnHeaders	Specifies whether column headers have to be shown.	GridControl	Boolean	<a href="#">Show or Hider Header</a>
ShowRowHeaders	Specifies whether row headers have to be shown.	GridControl	Boolean	<a href="#">Show or Hider Header</a>
DisplayHorizontalLines	Specifies whether horizontal lines have to be displayed.	GridControl	Boolean	<a href="#">Customize the Appearance</a>
DisplayVerticalLines	Specifies whether vertical lines have to be displayed.	GridControl	Boolean	<a href="#">Customize the Appearance</a>
GridLineColor	Specifies the grid line color.	GridControl	Boolean	<a href="#">Customize the Appearance</a>
PrintColumnHeader	Specifies whether column header has to be printed.	GridControl	Boolean	<a href="#">Printing Options</a>
PrintHorizontalLines	Specifies whether horizontal lines have to be printed.	GridControl	Boolean	<a href="#">Printing Options</a>
PrintRowHeader	Specifies whether row header has to be printed.	GridControl	Boolean	<a href="#">Printing Options</a>

PrintVerticalLines	Specifies whether vertical lines have to be printed.	GridControl	Boolean	<a href="#">Printing Options</a>
--------------------	--	-------------	---------	----------------------------------

Table 24: Grid Model Option

Property	Description	Type	Data Type	Reference links
ActivateCurrentCellBehavior	Used to set currentcell behaviour	GridControl	Enum	<a href="#">Grid Model Options</a>
AllowScrollCurrentCellInView	To set Allow Scroll in current cell view	GridControl	Enum	<a href="#">Grid Model Options</a>
AlphaBlendSelectionColor	To select alpha blend selection color	GridControl	Enum	<a href="#">Grid Model Options</a>
ClickedOnDisabledCellBehavior	Disable to click behaviour	GridControl	Enum	<a href="#">Grid Model Options</a>
ShowCurrentCellBorderBehavior	To show currentcell border behaviour	GridControl	Enum	<a href="#">Grid Model Options</a>
DefaultGridBorderStyle	To set Default Grid Border style.	GridControl	Enum	<a href="#">Grid Model Options</a>

#### 5.4.36.1 Show or Hider Header

You can show or hide row and column headers using the *ShowColumnHeaders* and *ShowRowHeaders* properties.

##### Column Header

To show the column header set the *ShowColumnHeaders* property to true.

The following code illustrates how show the column header in GridControl:

```
gridControl1.ShowColumnHeaders = true;
```

The following code illustrates how show the column header in GridDataBoundGrid:

```
gridDataBoundGrid.ShowColumnHeaders = true;
```

The following code illustrates how show the column header in GridGrouping control:

```
gridGroupingControl1.ShowColumnHeaders = true;
```

#### **Row Header**

To show the row header set the *ShowRowHeaders* property to true.

The following code illustrates how show the row header in GridControl:

```
gridControl1.ShowRowHeaders = true;
```

The following code illustrates how show the row header in GridDataBoundGrid:

```
gridDataBoundGrid.ShowRowHeaders = true;
```

The following code illustrates how show the row header in GridGrouping control:

```
gridGroupingControl1.ShowRowHeaders = true;
```

### **5.4.36.2 Customize the Appearance**

Essential Grid provides support to display horizontal and vertical lines and customize the grid line color.

#### **Displaying Horizontal Lines**

You can display horizontal lined using the *DisplayHorizontalLines*property.

The following code illustrates how to display horizontal lines in GridControl:

```
gridControl1.DisplayHorizontalLines = true;
```

The following code illustrates how to display horizontal lines in GridDataBoundGrid:

```
gridDataBoundGrid.DisplayHorizontalLines = true;
```

The following code illustrates how to display horizontal lines in GridGrouping control:

```
gridGroupingControl1.DisplayHorizontalLines = true;
```

### **Displaying Vertical Lines**

You can display vertical lined using the *DisplayVerticalLines* property.

The following code illustrates how to display vertical lines in GridControl:

```
gridControl1.DisplayVerticalLines = true;
```

The following code illustrates how to display vertical lines in GridDataBoundGrid:

```
gridDataBoundGrid.DisplayVerticalLines = true;
```

The following code illustrates how to display vertical lines in GridGrouping control:

```
gridGroupingControl1.DisplayVerticalLines = true;
```

### **Customizing Grid Line Color**

You can customize grid line color using the *GridLineColor* property.

The following code illustrates how to display vertical lines in GridControl:

```
gridControl1.GridLineColor = Color.Red;
```

The following code illustrates how to customize grid line color in GridDataBoundGrid:

```
gridDataBoundGrid.GridLineColor = Color.Red;
```

The following code illustrates how to customize grid line color in GridGrouping control:

```
gridGroupingControl1.GridLineColor = Color.Red;
```

### 5.4.36.3 Printing Options

Essential Grid provides support to print the column and row header and horizontal and vertical lines.

#### Printing Column Header

You can print the column header using the *PrintColumnHeader* property.

The following code illustrates how to print the column header in GridControl:

```
gridControl1.PrintColumnHeader = true;
```

The following code illustrates how to print the column header in GridDataBoundGrid:

```
gridDataBoundGrid.PrintColumnHeader = true;
```

The following code illustrates how to print the column header GridGrouping control:

```
gridGroupingControl1.PrintColumnHeader = true;
```

#### Printing Row Header

You can print the row header using the *PrintRowHeader* property.

The following code illustrates how to print the row header in GridControl:

```
gridControl1.PrintRowHeader = true;
```

The following code illustrates how to print the row header in GridDataBoundGrid:

```
gridDataBoundGrid.PrintRowHeader = true;
```

The following code illustrates how to print the row header GridGrouping control:

```
gridGroupingControl1.PrintRowHeader = true;
```

### **Printing Horizontal Lines**

You can print the horizontal lines using the *PrintHorizontalLines* property.

The following code illustrates how to print the horizontal lines in GridControl:

```
gridControl1.PrintHorizontalLines = true;
```

The following code illustrates how to print the horizontal lines in GridDataBoundGrid:

```
gridDataBoundGrid.PrintHorizontalLines = true;
```

The following code illustrates how to print the horizontal lines GridGrouping control:

```
gridGroupingControl1.PrintHorizontalLines = true;
```

### **Printing Vertical Lines**

You can print the vertical lines using the *PrintVerticalLines* property.

The following code illustrates how to print the vertical lines in GridControl:

```
gridControl1.PrintVerticalLines = true;
```

The following code illustrates how to print the vertical lines in GridDataBoundGrid:

```
gridDataBoundGrid.PrintVerticalLines = true;
```

The following code illustrates how to print the vertical lines GridGrouping control:

```
gridGroupingControl1.PrintVerticalLines = true;
```

## **5.4.36.4 Grid Model Options**

This section provides information on how to write syntax for the following GridModel Options:

- ActivateCurrentCellBehavior

- AllowScrollCurrentCellInView
- AlphaBlendSelectionColor
- ClickedOnDisabledCellBehavior
- ShowCurrentCellBorderBehavior
- DefaultGridBorderStyle

### **Activating Current Cell Behavior**

The following code illustrates how to activate current cell behavior:

```
this.gridGroupingControl1ActivateCurrentCellBehavior =  
GridCellActivateAction.SelectAll;
```

### **Allow scroll for Current cell view**

The following code illustrates how to allow scroll for current cell view:

```
this.gridGroupingControl1AllowScrollCurrentCellInView =  
GridScrollCurrentCellReason.Activate;
```

### **Alpha Blend Selection Color**

The following code illustrates how to select alpha blend selection color:

```
this.gridGroupingControl1AlphaBlendSelectionColor = Color.Red;
```

### **ClickedOnDisabledCellBehavior**

The following code illustrates how to define the current cell behavior when disabled cell is clicked:

```
this.gridGroupingControl1ClickedOnDisabledCellBehavior =  
GridClickedOnDisabledCellBehavior.Default;
```

### **Show Current Cell Border Behavior**

The following code illustrates how to show current cell border behavior:

```
this.gridGroupingControl1ShowCurrentCellBorderBehavior =  
GridShowCurrentCellBorder.HideAlways;
```

### **Default Grid Border Style**

The following code illustrates how to set default grid border style:

```
this.gridGroupingControl1DefaultGridBorderStyle =  
GridBorderStyle.None;
```

### 5.4.37 How to Overcome SendKey Exception in Currency Cell

The *CurrentCellKeyDown* event cannot be handled for *CurrencyTextbox*, when the Windows Forms application is hosted into Internet Explorer. It will throw an error message as, “SendKeys cannot run inside this application.” To overcome this exemption, set the *ActivateSendKey* property to false.

The following code illustrates this:

```
[C#]  
  
//GridGroupingControl:  
  
this.Grid.TableModel.Option.ActivateSendKey = false;  
  
//GridControl/GridDataBound:  
  
this.Grid.Model.Option. ActivateSendKey = false;  
  
//GridListControl:  
  
this.GridList.Grid.Model.Option.ActivateSendKey = false;
```

```
[VB]  
  
'GridGroupingControl:  
  
Me.Grid.TableModel.Option.ActivateSendKey = False  
  
'GridControl/GridDataBound:  
  
Me.Grid.Model.Option.ActivateSendKey = False  
  
'GridListControl:  
  
Me.GridList.Grid.Model.Option.ActivateSendKey = False
```

### 5.4.38 How To Release the Tab Focus from Grid

When you set the *ActiveControl* variable to *Grid control*, the focus will be on the *Grid control*. This may not allow you to navigate to other controls in the form. To overcome this difficulty, set the *ActiveControl* to the *Grid control*. Then set the *WantTabKey* property to false. This helps you navigate to the other controls in the form.

The following code illustrates how to release the tab focus from GridGroupingControl:

[C#]

```
private void FormMain_Load(object sender, EventArgs e)
{
    this.ActiveControl =
this.gridGroupingControl1.TableControl;
    gridGroupingControl1.WantTabKey = false;
    return;
}
```

[VB]

```
Private Sub FormMain_Load(ByVal sender As Object, ByVal e As EventArgs)
    Me.ActiveControl =
Me.gridGroupingControl1.TableControl
    gridGroupingControl1.WantTabKey = False
    Return
End Sub
```

#### 5.4.39 How To Resize Row Height Based On the Font of the Grid Cell Content

You can resize the row height based on the cell content height using the *ResizingRows* event. Calculate the content height using the *MeasuringString()* method. This considers the text, font style and font size to calculate the content size. Assign the calculated value to the *RowHeights* property.

You can resize the rows individually using the *AllowResizingIndividualRows()* method of *GridHelperclasses*. This method will be effective only when used before *InitializeComponent()*.

The following code illustrates resizing rows based on the grid cell content:

[C#]

```
public Form1()
{
    //To customize the row height for individual row
    GridEngineFactory.Factory = new
Syncfusion.GridHelperClasses.AllowResizingIndividualRows();
    InitializeComponent();
}

Size stringSize;
    void grid_ResizingRows(object sender,
Syncfusion.Windows.Forms.Grid.GridResizingRowsEventArgs e)
{

    if (e.Reason == GridResizeCellsReason.DoubleClick)
    {
        Graphics graphics = CreateGraphics();
        long maxHeight = 0;
        GridStyleInfo style =
this.grid.GetViewStyleInfo(e.Rows.Bottom, 1);
        stringSize = graphics.MeasureString(style.Text,
style.GdipFont, this.grid.Model.ColWidths[1]).ToSize();
        if (maxHeight < stringSize.Height)
        {
            maxHeight = (long)stringSize.Height;
        }
        this.grid.Model.RowHeights[e.Rows.Bottom] =
(int)maxHeight;
        e.Cancel = true;
    }
}
```

[VB]

```
Public Sub New()
    GridEngineFactory.Factory = New
Syncfusion.GridHelperClasses.AllowResizingIndividualRows()
    InitializeComponent()
End Sub

Private Sub grid_ResizingRows(ByVal sender As Object, ByVal e As
Syncfusion.Windows.Forms.Grid.GridResizingRowsEventArgs)
    If e.Reason = GridResizeCellsReason.DoubleClick Then
        Dim graphics As Graphics = CreateGraphics()
        Dim maxHeight As Long = 0
        Dim style As GridStyleInfo =
Me.grid.GetViewStyleInfo(e.Rows.Bottom, 1)
        stringSize = graphics.MeasureString(style.Text,
style.GdipFont, Me.grid.Model.ColWidths(1)).ToSize()
        If maxHeight < stringSize.Height Then
            maxHeight = CLng(stringSize.Height)
        End If
        Me.grid.Model.RowHeights(e.Rows.Bottom) =
CInt(maxHeight)
        e.Cancel = True
    End If
End Sub
```

#### 5.4.40 How to Align Image Button at the Centre of a Cell

To align the button at center of the cell, customize the model and the renderer of the cell. Specify the bounds of the button using the *OnLayout()* method.

The following code illustrates how to align the button at the center of the grid cell:

**[C#]**

```

protected override Rectangle OnLayout(int rowIndex, int colIndex,
GridStyleInfo style, Rectangle innerBounds, Rectangle[] buttonsBounds)
{
    TraceUtil.TraceCurrentMethodInfo(rowIndex, colIndex,
style, innerBounds, buttonsBounds);
    Rectangle rect = GridUtil.CenterInRect(innerBounds,
this.Model.ButtonBarSize);
    int width = rect.Width / 2;
    buttonsBounds[0] = new Rectangle(rect.X, rect.Y, width,
rect.Height);
    return innerBounds;
}

```

**[VB]**

```

Protected Overrides Function OnLayout(ByVal rowIndex As Integer, ByVal
colIndex As Integer, ByVal style As GridStyleInfo, ByVal innerBounds As
Rectangle, ByVal buttonsBounds As Rectangle()) As Rectangle
    TraceUtil.TraceCurrentMethodInfo(rowIndex, colIndex, style,
innerBounds, buttonsBounds)
    Dim rect As Rectangle = GridUtil.CenterInRect(innerBounds,
Me.Model.ButtonBarSize)
    Dim width As Integer = rect.Width / 2
    buttonsBounds(0) = New Rectangle(rect.X, rect.Y, width,
rect.Height)
    Return innerBounds
End Function

```

#### 5.4.41 How to Hold the Row Selection After the Cell is Deactivated

Essential Grid enables you to hold the selection after the cell is deactivated. To hold the selection, cancel the *SelectionChanging* event, when the *e.Reason* is *MouseDown* or *Clear*.

The following code holds the selection even after the cell is deactivated:

[C#]

```
void Model_SelectionChanging(object sender, GridSelectionChangingEventArgs e)
{
    if (e.Reason == GridSelectionReason.MouseDown || e.Reason ==
GridSelectionReason.Clear)
        e.Cancel = true;
}
```

[VB]

```
Private Sub Model_SelectionChanging(ByVal sender As Object, ByVal e As
GridSelectionChangingEventArgs)
    If e.Reason = GridSelectionReason.MouseDown OrElse e.Reason =
GridSelectionReason.Clear Then
        e.Cancel = True
    End If
End Sub
```

#### 5.4.42 How to Suppress KeyDown Event

To suppress the *TablecontrolCurrentCellKeyDown* event, enable the *SuppressKeyPress* property in this event. To get the Keys data, override the *ProcessCmdKey()* method.,,

1. Suppress the *KeyDown* event as given in the following code:

[c#]

```
void gridGroupingControll_TableControlCurrentCellKeyDown(object sender,
Syncfusion.Windows.Forms.Grid.Grouping.GridTableControlKeyEventArgs e)
{
    e.Inner.SuppressKeyPress = true;
}
```

. [VB]

```
Private Sub gridGroupingControl1_TableControlCurrentCellKeyDown(ByVal sender As Object, ByVal e As Syncfusion.Windows.Forms.Grid.Grouping.GridTableControlKeyEventArgs)
    e.Inner.SuppressKeyPress = True
End Sub
```

2. Handle the *ProcessCmdKey()* method to get the keys data as given in the following code:

[C#]

```
protected override bool ProcessCmdKey(ref Message msg, Keys keyData)
{
    Keys keyCode = keyData & Keys.KeyCode;
    switch (keyCode)
    {
        case Keys.Down: Console.WriteLine("down");
            break;
        case Keys.Up: Console.WriteLine("Up");
            break;
        case Keys.Right: Console.WriteLine("Right");
            break;
        case Keys.Left: Console.WriteLine("Left");
            break;
        case Keys.PageUp: Console.WriteLine("PageUp");
            break;
        case Keys.PageDown: Console.WriteLine("PageDown");
            break;
    }
    return base.ProcessCmdKey(ref msg, keyData);
}
```

[VB]

```
Protected Overrides Function ProcessCmdKey(ByRef msg As Message, ByVal keyData As Keys) As Boolean
    Dim keyCode As Keys = keyData And Keys.KeyCode
    Select Case keyCode
        Case Keys.Down
            Console.WriteLine("down")
            Exit Select
        Case Keys.Up
            Console.WriteLine("Up")
```

```
        Exit Select
    Case Keys.Right
        Console.WriteLine("Right")
        Exit Select
    Case Keys.Left
        Console.WriteLine("Left")
        Exit Select
    Case Keys.PageUp
        Console.WriteLine("PageUp")
        Exit Select
    Case Keys.PageDown
        Console.WriteLine("PageDown")
        Exit Select
    End Select
    Return MyBase.ProcessCmdKey(msg, keyData)
End Function
```

#### 5.4.43 How to Restrict Alphabetical Characters in NumericUpDown Cell

The NumericUpDown cell can be customized to allow only the numeric characters by setting the AcceptAlphaKeys property to *false*. This property can be derived from the GridNumericUpDownCellModel class.

##### [C#]

```
GridNumericUpDownCellModel model =
this.gridControl1.CellModels[GridCellTypeName.NumericUpDown] as
GridNumericUpDownCellModel;
model.AcceptAlphaKeys = false;
```

##### [VB .NET]

```
Dim model As GridNumericUpDownCellModel =
TryCast(Me.gridControl1.CellModels(GridCellTypeName.NumericUpDown),
GridNumericUpDownCellModel)
model.AcceptAlphaKeys = False
```

#### 5.4.44 How to Enable ColorStyle Settings in Windows Grids

This feature enables you to apply enhanced visual styles to the following Windows Forms Grid controls: Grid, GridGrouping, GridDataBoundGrid, and GridList.

You can apply one of the following styles:

- Office2007Blue
- Office2007Black
- Office2007Silver
- Office2010Blue
- Office2010Black
- Office2010Silver
- Metro

### Properties

Property	Description	Type	Data Type
EnableLegacyStyle	Get or set the value.		Boolean

### Sample Link

{Installed

Path}\Syncfusion\EssentialStudio\{Version}\Windows\Grid.Grouping.Windows\Samples\2.0\Styling and Formatting\Skin Customization Demo\

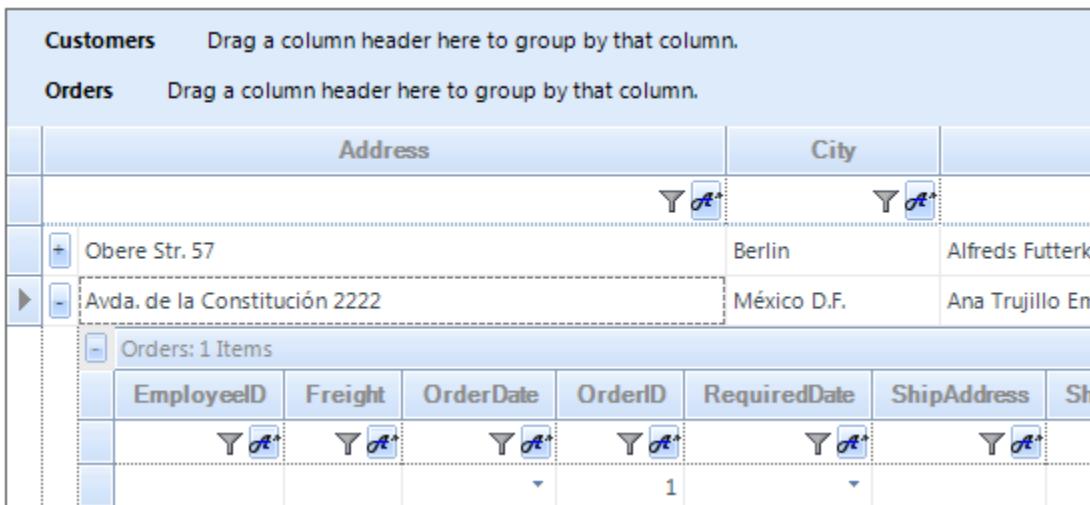


Figure 496: Office2010Blue Theme

To enable this feature, use the following code:

```
[C#]
```

```
this.gridGroupingControl1.TableModel.EnableLegacyStyle = false;
```

[VB]

```
Me.gridGroupingControll.TableModel.EnableLegacyStyle = False
```

#### 5.4.45 How to trigger an event when the ComboBox DropDownList has null values or has no datasource bound to it?

If a ComboBox is neither bound to any datasource nor has list items, it can be notified to the user by clicking on it. The notification message that the user wants to display can be given as shown in the following TableControlCurrentCellShowingDropDown event.

[C#]

```
//Code..

void gridGroupingControll_TableControlCurrentCellShowingDropDown(object
sender, GridTableControlCurrentCellShowingDropDownEventArgs e)
{
GridComboBoxCellRenderer rend = e.TableControl.CurrentCell.Renderer as
GridComboBoxCellRenderer;
ListBox list = rend.ListBoxPart;
if (list.Items.Count == 0)
{
    // write your code here..
}
//Code..
```

[VB]

```
'Code..
void gridGroupingControll_TableControlCurrentCellShowingDropDown(object
sender, GridTableControlCurrentCellShowingDropDownEventArgs e)

Dim rend As GridComboBoxCellRenderer =
TryCast(e.TableControl.CurrentCell.Renderer, GridComboBoxCellRenderer)
Dim list As ListBox = rend.ListBoxPart
If list.Items.Count = 0 Then

    ' write your code here..

End If

'Code..
```

### 5.4.46 How to get the selected text from a TextBox cell?

To get the selected text from a TextBox cell, the TextBox renderer should be acquired inside the **CurrentCellStartEditing** event. The **MouseUp** event should be hooked to get the selected text from the TextBox cell.

**[C#]**

```
this.gridDataBoundGrid1.CurrentCellStartEditing += new
CancelEventHandler(gridDataBoundGrid1_CurrentCellStartEditing);
}

void gridDataBoundGrid1_CurrentCellStartEditing(object sender,
CancelEventArgs e)
{
    GridCurrentCell cc = this.gridDataBoundGrid1.CurrentCell;
    if (cc.Renderer is GridTextBoxCellRenderer)
    {
        GridTextBoxCellRenderer rend = cc.Renderer as
GridTextBoxCellRenderer;
        rend.TextBox.MouseUp += new
MouseEventHandler(TextBox_MouseUp);
    }
}

void TextBox_MouseUp(object sender, MouseEventArgs e)
{
    TextBox text = sender as TextBox;
    if (text.SelectionLength > 0)
    {
        Console.WriteLine(text.SelectedText);
    }
}
```

**[VB]**

```
AddHandler gridDataBoundGrid1.CurrentCellStartEditing, AddressOf
gridDataBoundGrid1_CurrentCellStartEditing
End Sub
Private Sub gridDataBoundGrid1_CurrentCellStartEditing(ByVal
sender As Object, ByVal e As CancelEventArgs)
    Dim cc As GridCurrentCell =
Me.gridDataBoundGrid1.CurrentCell
    If TypeOf cc.Renderer Is GridTextBoxCellRenderer Then
        Dim rend As GridTextBoxCellRenderer = TryCast(cc.Renderer,
GridTextBoxCellRenderer)
        AddHandler rend.TextBox.MouseUp, AddressOf TextBox_MouseUp

```

```
End If  
End Sub  
  
Private Sub TextBox_MouseUp(ByVal sender As Object, ByVal e As  
MouseEventArgs)  
    Dim text As TextBox = TryCast(sender, TextBox)  
    If text.SelectionLength > 0 Then  
        Console.WriteLine(text.SelectedText)  
    End If  
End Sub
```

Refer to the following sample file for more details:

[http://www.syncfusion.com/downloads/Support/DirectTrac/99890/99890\\_SelectedText653111008.zip](http://www.syncfusion.com/downloads/Support/DirectTrac/99890/99890_SelectedText653111008.zip)

Similarly, the text from the Combobox cell, RichTextbox cell etc can be acquired by taking their respective renderers. You can also achieve the same behavior for the Grid control and the GridGrouping control.

#### 5.4.47 How to enable double-click on the formula cell?

To enable double-click on the formula cell, set the **ForceEditWhenActivated** property to *False*. The **ActivateCurrentCellBehavior** property cannot be achieved in the GridFormulaCellRenderer unless the property is set to *False*.

```
[C#]  
this.gridControl1ActivateCurrentCellBehavior =  
GridCellActivateAction.DblClickOnCell;  
GridFormulaCellRenderer.ForceEditWhenActivated = false ;
```

```
[VB.NET]  
Me.gridControl1ActivateCurrentCellBehavior =  
GridCellActivateAction.DblClickOnCell  
GridFormulaCellRenderer.ForceEditWhenActivated = False
```

#### 5.4.48 Why does the call to the Application.DoEvents never return while updating the Grid using the BeginUpdate() method?

If the Grid is performing a time-consuming task, the BeginUpdate() and EndUpdate() methods will be used to ensure the updates correctly reflected in the grid.

The problem arises when users try to dispatch windows messages to the controls (triggering the Application.DoEvents) inside the BeginUpdate() and EndUpdate() methods.

**Root Cause:**

The root cause of the problem is that the usage of Application.DoEvents will ask the WM\_PAINT messages of Grid to execute, but the WM\_PAINT messages will be ignored since the grid is in an updating state (the WM\_PAINT message will be processed by the Grid only when the EndUpdate is called). Hence the above makes an **Infinite loop execution**, meaning the loop will never be returned from WM\_PAINT.

The above problem can be resolved in the following two ways:

**Solution 1:**

If the usage of the Application.DoEvents is mandatory, then the simplest way to get rid of this problem is by calling the grid.CancelUpdate before triggering Appplication.DoEvents. After that, the grid can be updated with the call to Grid.BeginUpdate.

**Solution 2:**

Add a public static member to the class where Application.DoEvents is to be used. Then override the Grid's WndProc method and process the WM\_PAINT if the DoEvents loop is triggered.

```
[C#]
public static bool InDoEvents = false;
private void button1_Click(object sender, System.EventArgs e)
{
    this.gridControl1.BeginUpdate();
    this.gridControl1.Refresh();

    Console.WriteLine("before Application.DoEvents");
    Form1.InDoEvents = true;
    Application.DoEvents();
    Form1.InDoEvents = false;
    Console.WriteLine("after Application.DoEvents");

    this.gridControl1.EndUpdate();
}

public class MyGridControl : GridControl
{
    public const int WM_PAINT = 15;
    protected override void WndProc(ref
System.Windows.Forms.Message msg)
    {
        if (Form1.InDoEvents && msg.Msg == WM_PAINT)
            DefWndProc(ref msg);
        else
            base.WndProc(ref msg);
    }
}
```

**[VB]**

```
Public Shared InDoEvents As Boolean = False
Private Sub button1_Click(sender As Object, e As
System.EventArgs)
    Me.gridControl1.BeginUpdate()
    Me.gridControl1.Refresh()

    Console.WriteLine("before Application.DoEvents")
    Form1.InDoEvents = True
    Application.DoEvents()
    Form1.InDoEvents = False
    Console.WriteLine("after Application.DoEvents")

    Me.gridControl1.EndUpdate()
End Sub
Public Class MyGridControl
    Inherits GridControl
    Public Const WM_PAINT As Integer = 15
    Protected Overrides Sub WndProc(ByRef msg As
System.Windows.Forms.Message)
        If Form1.InDoEvents AndAlso msg.Msg = WM_PAINT Then
            DefWndProc(msg)
        Else
            MyBase.WndProc(msg)
        End If
    End Sub
End Class
```

#### 5.4.49 How can I implement an Excel accounting format in the Syncfusion WinForms GridControl?

We can achieve this behavior by setting the cell type to Currency. The currency symbol is removed from the cell data with the following code:

**[C#]**

```
this.gridControl1[i, j].CurrencyEdit.CurrencySymbol = "";
```

**[VB]**

```
Me.gridControl1[i, j].CurrencyEdit.CurrencySymbol = "";
```

The CurrencyDecimalDigits property is used and the digits after the decimal point are set using the code below:

[C#]

```
this.gridControl1[i, j].CurrencyEdit.CurrencyDecimalDigits = 2;
```

[VB]

```
Me.gridControl1[i, j].CurrencyEdit.CurrencyDecimalDigits = 2;
```

By default the contents are left-aligned, so the horizontal alignment is set to Right.

[C#]

```
this.gridControl1.ColStyles[j].HorizontalAlignment =  
Syncfusion.Windows.Forms.Grid.GridHorizontalAlignment.Right;
```

[VB]

```
Me.gridControl1.ColStyles[j].HorizontalAlignment =  
Syncfusion.Windows.Forms.Grid.GridHorizontalAlignment.Right;
```

### 5.4.50 How to achieve Excel-like Text Alignment in Grid Control?

Text alignment in Grid cells is similar to text alignment in Excel cells. When no data type is explicitly specified for a cell, by default, text containing numbers is right-aligned, and the remaining text is left-aligned.

If the data type of the cell is set to *Integer*, *Double* or *Decimal*, the text is right-aligned; else the text is left-aligned. In some cases, when the text contains numbers, but the data type of the cell is set to *String*, the text is left-aligned. This default text alignment also applies to the respective Header cells of the Grid control.

Excel-like Text Alignment can be achieved in Grid cells by using the **ExcelLikeAlignment** property. It enables you to apply Excel-like alignment to text in grid cells. This property is of Boolean data type and is set to *false*, by default.

[C#]

```
this.gridControl1.ExcelLikeAlignment = true;
```

[VB]

```
me.gridControl1.ExcelLikeAlignment = True;
```



**Note:** The preceding settings can be applied to all Grid Controls.

### 5.4.51 How to export the user-defined function from Grid to Excel workbook?

To export the user-defined formula library function from Grid to Excel:

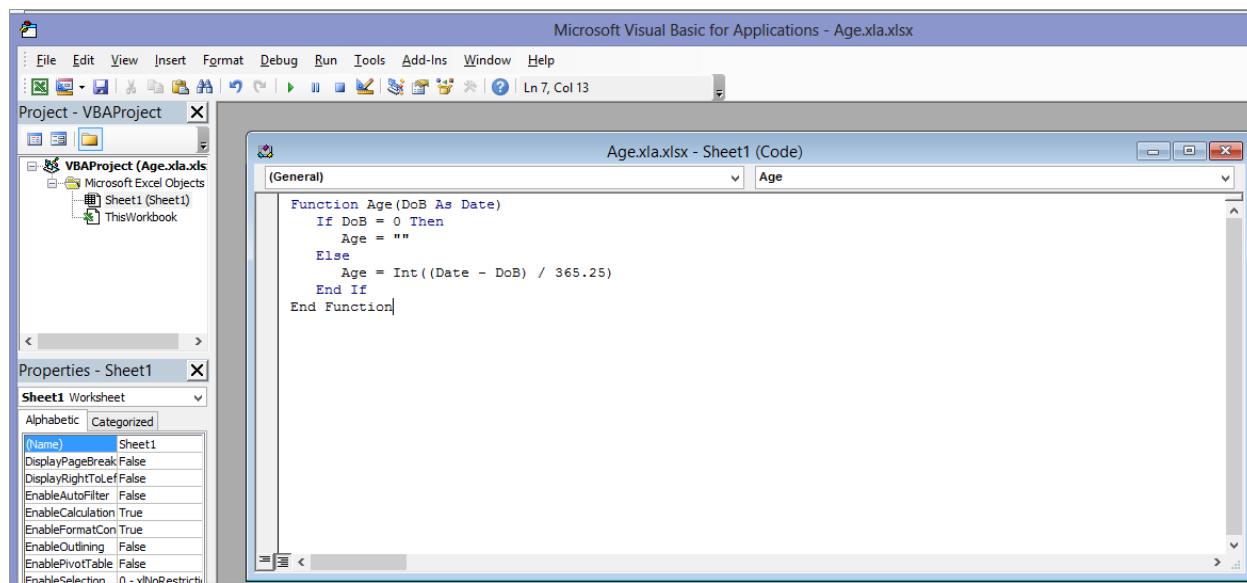
1. Create an Excel Add-In file (\*.xla) including the user-defined function as illustrated in the following link: [http://msdn.microsoft.com/en-us/library/office/aa140990\(v=office.10\).aspx](http://msdn.microsoft.com/en-us/library/office/aa140990(v=office.10).aspx).
2. Now, bind the add-in file path and custom method name with **GridExcelConverterControl** through **AddCustomFunction** method.

#### Creating the XLA File

To create an XLA file in Excel:

1. Create a new, empty workbook and click **Save As** from the **File** menu.
2. In the **Save As** dialog box, select the location to save the workbook. Enter the file name for the Add-In workbook, and then select the type as **Excel Add-In** (\*.xla).
3. Click **Save**, and then press ALT+ F11 to open the VBA Editor. Now, you can create the custom function through the VBA Editor.

The following screen shot illustrates how to create the user-defined function through the VBA editor.



The XLA file will not be visible in Excel because Add-Ins are never visible. All changes must be done to the VBA Editor only.

The following code illustrates how to export the user-defined function to Excel.

[C#]

```
// Exporting the user-defined function.
string xla = @"..\..\AddIns\Add.xla";
Syncfusion.GridExcelConverter.GridExcelConverterControl gecc = new
Syncfusion.GridExcelConverter.GridExcelConverterControl();
gecc.AddCustomFunction(xla, "add", saveFileDialog1.FileName);
gecc.GridToExcel(this.gridControl1.Model, saveFileDialog1.FileName);
```

## **Index**

### **A**

ABS 250

Accessing Values in the Grid Data Bound Grid and in the Data Source 551

ACCRINT 250

ACOS 251

ACOSH 251

ACOT 251

ACOTH 252

AcscH 252

Activating Current Cell Behavior 185

Adding Formulas to the Formula Library 245

Adding GroupDropArea 710

Adding Hierarchical GroupDropArea to an Application 717

Adding Multi-Grid Printing to an Application 501

Adding Skin Manager to an Application 1183

Adding Sort Icon Placement to an Application 29

Adding Special Controls to Grid Cells 147

Adding Virtual Grid 101

ADDRESS 252

Advanced Features 992

AND 253

Appearance 426, 844

Appearance Options 848

Appearance Properties 436

Applying Enhanced GridVisualStyle to the Application 1152

Applying Metro Theme to a Control 1155

Applying Special Column Formats 92

Applying User-Defined Colors as Metro Themes 1156

ARABIC 253

ArrayList Class with IBindingList Support 548

AsecH 253

ASIN 254

ASINH 254

ATAN 254

ATAN2 255

ATANH 255

AutoComplete Support for Combo Box in Edit Mode 153

Autosizing Custom Cell 481

AVEDEV 255

AVERAGE 256

AVERAGEA 256

AVERAGEIF 257

AVERAGEIFS 258

AVG 259

### **B**

BackColor 178

Banner Cells 508

BASE 259

BaseStyles 181, 873

Basic Grid Control 78

Basic Grid Data Bound Grid 82

Before You Begin 44

BigMul 259

Bind a Grid Grouping Control to a Manual Data Source 596

Binding a Grid Grouping Control to an MDB File 594

Binding to a DataTable 549

Binding to an ArrayList 546

Binding to an MDB File By Using VS 2003 45

Binding to an MDB File By Using VS 2005 64

Binding to Custom Collections 676

Binding to XML Data 674

Binom.Inv 261

BINOMDIST 260	Collection Base 686
Browse-Only Grid 200	Color Edit 150
Built-in Error Provider Support 1178	ColumnStyles 858
Built-in Navigation Support for RecordNavigationControl in GridGroupingControl 1087	COMBIN 264
Button Edit 206	COMBINA 265
C	Combo Box 151, 1153
Calculator Text Box 209	ComboBox and RichText Cell Conversion 116
Calendar 211	ComboBoxBase Feature 1077
Card View Layout 1143	Common to GridControl, GridDataBoundGrid and GridGrouping 1380
CEILING 261	CONCATENATE 265
CEILING.MATH 262	Concepts and Features 146, 545, 612, 1075
Cell Attributes 140	Conditional Formatting 866
Cell Comment Tips 198	CONFIDENCE 266
Cell Events 1040	CONFIDENCE.T 266
Cell Style Architecture 175	Context Menu 1344
Cell Tips 197	Control 155
Cell Types 142	Control Events 1050
CellButtonClicked Event 1042	Controlling the Resize Behavior 475
CellClick Event 1043	CORREL 267
CellDoubleClick Event 1046	COS 267
CellDrawn Event 1045	COSH 268
Cell-level and grid-level 531	COT 268
Changing Column Order in a Grid Data Bound Grid 561	COTH 269
Chart Cell 222	COUNT 269
Check Box 148	COUNTA 269
CHI- TEST 263	COUNTBLANK 270
CHIDIST 262	COUNTIF 270
CHIINV 262	COVAR 271
Choosing the Best Grid 24	COVARIANCE.P 271
CLEAN 264	COVARIANCE.S 272
Clipboard Events 525	Covered Cells 486
Clipboard Support 519	Creating a Grid Object 393
Coded UI Support in Windows Grids 1158	Creating Grid Control 133
	Creating Grid Data Bound Grid 538

- Creating Grid Grouping Control 593
- Creating Grid List Control 1068
- Creating Grid Record Navigation Control 1083
- Creating the Project and Data Source 98
- Creating Unit Tests with VS2010 1160
- CRITBINOM 272
- Cross Sheet Reference 378
- CSC 273
- CSCH 273
- CUMIPMT 273
- CUMPRINC 274
- Currency Cell Conversion 115
- Currency Edit 156
- Current Cell 231
- Current Cell Events 1047
- CurrentRecordContextChange Event 1026
- Custom Borders 193
- Custom Cell Types 206, 1102
- Custom Drawing 461
- Custom Filter Dialog 1009
- Custom Grouping 993
- Custom Sorting 1000
- Custom Summary 1005
- Customize the Appearance 1429
- Customizing GroupDropArea 711
- Customizing List control 1078
- D**
- Data Binding 143, 672
- Data binding and Selection Modes 1075
- Data Binding using ADO.NET 673
- Data Handling 527
- Data Relations 571
- Data Representation 702
- Datasource 1358
- DATE 274
- Date Time Picker 212
- DateTime Cell Conversion 116
- DATEVALUE 275
- DAVERAGE 276
- DAY 276
- DAY360 276
- DB 277
- DCOUNT 278
- DCOUNTA 278
- DDB 278
- DECIMAL 279
- Defining a FormulaCell 243
- DEGREES 279
- Delete Collection of Records in GridGroupingControl 1061
- Deploying Extension Assembly 1159
- Deployment Requirements 40
- Derived Commands 399
- Deriving a Cell Control 201
- Deriving GridPrintDocument 487
- DEVSQ 280
- DGET 280
- Disc 280
- Display GroupDropArea in Hierarchy 717
- DisplayElements 1335
- DivRem 281
- DMAX 281
- DMIN 281
- Documentation 34
- Dollar 281
- DollarDe 282
- DollarFr 282
- Double Text Box 227
- Drag Column Header 503
- Drop-Down Form and User Control Cell 225

- Drop-Down Grid Cell 223  
DSTDEV 282  
DSTDEVP 283  
DSUM 283  
Duration 283  
DVAR 283  
DVARP 284  
Dynamic Filter 783  
Dynamic Formatting 869  
**E**  
Elaborate Structure of the Control 138  
Embeding OLE Objects in the Grid Cell 208  
Enable or Disable Sorting 742  
Enable the Card View Layout 1146  
Enables Migration of .Net Grid to Essential Grid 587  
Enabling Error Alert 1179  
Enhanced Numeric Up Down 213  
Enhanced Visual Styles for the Syncfusion Windows Grids 1151  
Enter Key Behavior 483  
ERROR.TYPE 284  
EVEN 284  
Events 585, 732, 1021  
Excel Converter Options 118  
Excel Export 30, 402, 946  
Excel-Like Filters 1138  
EXP 285  
ExpandAll and CollapseAll Methods 576  
Exploring Summaries 749  
EXPONDIST 285  
Export as Image Support 409  
Export Summary as Caption 1065  
Exporting Grid Data to Excel 2010 121  
Exporting Grid Grouping Data To Excel 118  
Exporting Multiple Grids 119  
Exporting the Grid Control or Grid Data Bound Grid To Excel 114  
Expression Fields 762  
Expressions 1346  
**F**  
F.Inv.Rt 289  
FACT 286  
FACTDOUBLE 286  
False 287  
FDIST 287  
Feature Summary 139, 600  
Field Chooser Events 906  
Field Chooser for Grid Data Bound Grid 590  
Field Chooser for Stacked Header 909  
Fill Series 1134  
Filter Bar 777  
Filter By DisplayMember 791  
Filter for Specific Columns 32  
Filter Optimization 1020  
Filtering 31  
Filtering a Grid Data Bound Grid 553  
Filtering By Display Member 1118  
Filtering Null Values from the Grid 1142  
Filters and Expressions 761  
Find Replace 238  
Finv 288  
FISHER 289  
FISHERINV 290  
Fixed 290  
Floating Cells 489  
FLOOR 290  
Focused Selection 923  
FORECAST 291

- Foreign Key Columns: Showing One Value but Saving Another 563
- Foreign Key KeyWords Relation 814
- Foreign Key Reference For Child Table 1323
- Foreign Key Reference Relation 806
- Format Cells Dialog Support 864
- Formatting Drop-down List 464
- Formula Cell 158
- Formula Cell Conversion 117
- Formula Support 243
- Freeze Pane 235
- Frequently Asked Questions 1197
- Frozen Rows and Columns 470
- Function Reference Section 249
- Functionalities 144
- FV 292
- Fvschedule 292
- G**
- Gaming Applications 513
- GAMMADIST 293
- GAMMAINV 295
- GAMMALN 294
- GCD 295
- General 146, 1293
- Generic Collection 690
- GEOMEAN 296
- Get Cell Styles 878
- Getting Started 43
- Grid Aware Text Box 1090
- Grid Control 132
- Grid Control Designer 432
- Grid Controls 132
- Grid Data Bound Grid 537
- Grid Data Bound Grid Performance 583
- Grid Designer 971
- Grid Dynamic Filter 1097
- Grid Field Chooser 905, 1117
- Grid Folder Browser 515
- Grid Format Cell Dialog 1121
- Grid Grouping Control 593
- Grid Helper Classes 1091
- Grid Layout 892
- Grid List Control 159, 1068
- Grid Model Options 1432
- Grid New Feature 199
- Grid Properties 435
- Grid Record Navigation Control 1083
- Grid to PDF Conversion 1091
- Grid Visual Styles 430
- GridBoundColumns and Controlling the Column Format 557
- GridCellModelBase 203
- GridCellRendererBase 204
- GridControl 1197
- GridControl to Excel Conversion Process 115
- GridDataBoundGrid 1241
- GridFontInfo 179
- GridFormatCellDialog 426
- GridGrouping Control 1293
- GridInCell 215
- GridRangeInfo 184
- GridStyleInfo Class Overview 175
- GridStyleInfo Properties 189
- GroupAdded Event 1027
- GroupByOptions 719
- GroupCollapsed Event 1029
- GroupCollapsing Event 1028
- GroupDropArea 708
- GroupExpanded Event 1031
- GroupExpanding Event 1030

- Grouping 26, 702  
Grouping Performance 667  
GroupRemoving Event 1032  
Groups 1360  
GroupSummaryInvalidated Event 1032  
Growth 296
- H**
- Handling Events to Retrieve Data for Virtual Grid 105  
HARMEAN 297  
Header 161  
Header Rows and Columns 469  
Hiding Rows and Columns 467  
Hierarchical Grid with Tree Lines 575  
High Frequency Real Time Updates 525  
High Frequency Updates 652  
Highlighting Row and Column Headers 241  
HLOOKUP 298  
HOUR 298  
How can I implement an Excel accounting format in the Syncfusion WinForms GridControl? 1447  
How can we make the Current Row Bold? 1238  
How does one add currency symbol for parent table and child table 1322  
How Does the Helper class Support Percentage Sizing in GridControl / GridDataBoundGrid 1396  
How to access a particular DisplayElement if RowIndex is provided 1336  
How to access a particular group 1361  
How to access a particular record from a table 1351  
How to access a record given the row index 1352  
How to access all the DisplayElements or a particular DisplayElement in the GridGrouping control 1336  
How to access all the groups 1360
- How to access child table's display elements 1314  
How to access selected records 1348  
How to access sorted or filtered records 1352  
How to Access the Corresponding Parent Table's Row from the Child 1293  
How to access the current record 1295  
How to access the group from the DisplayElements 1362  
How to access the group from the Record 1363  
How to Access the Nested Tables in a GridGroupingControl 1315  
How to access unfiltered records 1353  
How to achieve Excel-like Text Alignment in Grid Control? 1448  
How to achieve Numeric Column Sorting in GridDataBoundGrid 1292  
How to Add a Sort Icon (Up / Down Arrow) in a GridControl's Column Header 1198  
How to add and format unbound columns in GridGrouping control 1296  
How to Add Conditional Formatting to Rows 1381  
How to add Expression columns 1346  
How to Align Image Button at the Centre of a Cell 1437  
How to Align Radio Buttons 1197  
How to align the summary cells in the GridGroupingControl 1326  
How to Allocate Equal Size for Each of the Columns in all the Tables 1337  
How to Apply Filtering on the GridDataBound Grid based on a Node Selected in the Tree View? 1287  
How to apply grouping properties for a particular column 1363  
How to apply grouping properties for ChildLevelGroups 1364  
How to apply grouping properties for TopLevelGroups 1365  
How to apply text color for a RichText celltype in GridGroupingControl 1342

- How to attach a context menu to a cell in the GridGrouping control 1344
- How to attach a context menu to the GridGrouping control 1345
- How to Avoid RangeStyles being Written out to Code in a Derived GridControl 1200
- How to Avoid the A, B, C and / or 1, 2, 3 in the Headers 1199
- How to Capture Function Keys When the Current Cell is in Active Edit Mode 1382
- How to Capture Mouse and Key Events When the Text Box Cell is in an Active State 1383
- How to Change a Cell's Font to a Particular Font Family 1204
- How to Change the Appearance of a Single Header Cell 1201, 1241
- How to Change the Backcolor of a Column 1202, 1242
- How to Change the Backcolor of a Single Cell 1202, 1243
- How to Change the Backcolor of a Single Row 1203, 1244
- How to change the caption text 1296
- How to Change the Color of All Headers 1385
- How to Change the Column Header Text 1245
- How to Change the Format of the Summary Cell in Group Caption 1334
- How to Change the Look of a Cell's Border 1204
- How to Change the Look of a Column's Border 1246
- How to Change the Mouse Cursor for a GridControl 1386
- How to change the row header to display line numbers instead of the black triangle in GridDataBoundGrid 1247
- How to Change the Size of the Combo Box Button 1386
- How to Control the Number of Visible Items in a Combo Box Cell 1388
- How to Control the Way the Grid Handles Exceptions 1389
- How to Control Whether OLE Drag-and-Drop is Supported in the Grid 1390
- How to convert the contents of a GridControl or GridDataBoundGridControl to Excel 1391
- How to Copy a Range of Cells to the Clipboard 1249
- How to Create a Cell that contains Hypertext Link along with Different Formatted Texts 1207
- How to Create a Multi-Select DropDownList in a Cell 1206
- How to create multiple rows per record in a GridGroupingControl 1338
- How to create weighted summaries in the GridGroupingControl 1327
- How to define a summary row in the grid 1328
- How to derive user defined groups 1366
- How to Determine that No Cell is Selected 1285
- How to Disable Clipboard Cut / Copy / Paste in a Grid 1392
- How to Disable Sorting While Record Added 1286
- How to disable the resizing of rows and columns 1297
- How to Display Placeholder Characters when Cell Content Exceeds Cell Width 1393
- How to Do Custom Sorting for a Specified String Column 1374
- How to draw a check box cell in a GridDataBoundGrid header 1250
- How to Efficiently Customize the Child Table / Group using a Custom Engine 1298
- How to Enable ColorStyle Settings in Windows Grids 1441
- How to enable double-click on the formula cell? 1445
- How to Exclude Hidden Rows and Column for Scrolling? 1209
- How to export CellCommentTips to Excel using GridExcelConverterControl 1400
- How to export the user-defined function from Grid to Excel workbook? 1449
- How to find a DisplayElement Type 1337

- How to format summary rows 1329  
How to freeze specified columns 1339  
How to get a record index given the rowindex 1353  
How to get a rowindex given the record index 1354  
How to Get a Value from the Parent Record of the Nested Table when the Dropdown List is Clicked 1316  
How to Get all the Data in a GridControl as an Array 1210  
How to Get New Form Window in Front of an Active Window while Double Clicking a Cell 1313  
How to Get Selected Ranges in GridDataBoundGrid 1283  
How to Get Selected Rows in GridDataBoundGrid 1282  
How to Get Summary Cell Values in sorting? 1333  
How to Get the Cell Coordinates Under a Given Point 1403  
How to Get the New Value and the Old Value when an Item is Selected in a Combobox Cell 1405  
How to Get the Position of a Row in the DataSource from the Current Record 1358  
How to get the row position of the AddNewRecord in the GridGroupingControl 1354  
How to Get the Screen Point for the Given Cell Coordinates 1404  
How to Get the Selected Ranges in a Grid 1401  
How to get the selected text from a TextBox cell? 1444  
How to Get the Top / Bottom / Left / Right Viewable Row and Column Indexes 1404  
How to group a column programmatically 1298  
How to Have Character Casing Settings for a Cell 1407  
How to Hide / Unhide the Columns in a Grouping Grid 1303  
How to Hide a Row / Column 1408  
How to Hide the Custom Option in the FilterBar DropDown 1298  
How to hide the numbered row and column headers while printing or print previewing in GridControl 1211  
How to hide the row headers of a child table in the GridGroupingControl 1318  
How to Hold the Row Selection After the Cell is Deactivated 1438  
How to Implement a Fill Paste in the Grid 1212  
How to Include an Icon in the Column Header 1214, 1257  
How to Incorporate Summary Cells while Sorting? 1333  
How to Insert an Unbound CheckBox Column 1252  
How to Make a Cell Display '...' if it is Not Wide Enough 1408  
How to make resizing possible in additional row headers of a GridControl and GridDataBoundGrid 1410  
How to Make the Grid Behave Like a List Box 1409  
How to Make the GridFilterBar Use an Autocomplete Combo Box 1259  
How to Make the Particular Cells ReadOnly 1258  
How to Make Use of the Journal Control using GridGrouping Controls 1310  
How to merge two columns in a GridDataBoundGrid 1261  
How to Move Groups Upward/Downward Using Custom Comparer 1371  
How to Move to the Next Row from the Last Cell of a Row 1411  
How to Optimize Pixel Scrolling in a GridControl 1215  
How to Overcome SendKey Exception in Currency Cell 1434  
How to Paste Clipboard Contents Bigger than GridDataBoundGrid Row and Column Count 1262

- How to perform Custom Sorting in GridGroupingControl 1375
- How to Place a Checkbox in a Header Cell 1303
- How to Place a UserControl in a Header Cell 1305
- How to pre-select a checkbox inside a grid cell 1218
- How to Prevent Columns Resizing for Child Tables 1307
- How to Prevent Resizing a Specific Column in a GridControl 1411
- How to Prevent Showing the '+' Sign Next to the Parent Rows with no Children 1265
- How to prevent sorting if the column has the same value in each cell? 1378
- How to Print a Grid 1412
- How to Print Preview a Grid 1414
- How to Put a Cell in Overstrike Mode so Characters get Replaced instead of Inserted as you type 1417
- How to Put a CheckBox in a Header Cell in a GridControl or GridDataBoundGrid 1415
- How to Put a ComboBox in a Header Cell in a GridControl or GridDataBoundGrid 1419
- How to QueryCellFormattedText and SaveCellFormattedText 1269
- How to reject the changes made to the GridGroupingControl 1308
- How To Release the Tab Focus from Grid 1434
- How To Resize Row Height Based On the Font of the Grid Cell Content 1435
- How to resize the columns to fit in a page while exporting to PDF 1309
- How to resolve the "Object reference not set to an instance of an object" error generation in GridDataBoundGrid Designer 1291
- How to Restrict Alphabetical Characters in NumericUpDown Cell 1441
- How to retrieve a summary item 1330
- How to Retrieve the DataRow from the GridDataBoundGrid with the RowIndex 1270
- How to Retrieve the Text From a Cell 1219
- How to save the ComboBox cell value instantly after the dropdown is closed? 1240
- How to save the contents of a Grid in memory rather than a file system to export to another grid 1220
- How to select a record programmatically 1349
- How to Select All Text in a Grid After Clicking a Cell 1423
- How to select all the contents in a cell after double-clicking on the contents 1290
- How to Set a Value into a Cell 1223
- How to Set Column Style Properties for the Nested Tables in a Grouping Grid 1319
- How to set conditional formatting in the GroupingGrid 1340
- How to set ListBoxSelectionMode 1349
- How to set style properties of a nested table 1321
- How to set texts in the preview record 1340
- How to Set the Background Color for a Grid 1422
- How to Set the Cell Properties for a Range of Cells 1220
- How to set the cursor to GridGroupingControl instead of the default one? 1341
- How to Set the Height of a Row 1222, 1272
- How to Set the Number of Rows / Columns 1221
- How to Set the Text Color that Appears in a Cell 1225
- How to Set the Text in a Header Cell 1224
- How to Set the Width of a Column 1225, 1271
- How to Set Transparent Backcolor for a GridControl 1226
- How to set up a datasource to the grouping grid 1359
- How to Show a ContextMenu in the Center of CurrentCell or Selected Cells Irrespective of the Mouse Click 1227
- How to Show Multiple Images in a Cell 1228
- How to show Summary Results in the Inner Most Table on every Parent Level in Nested Related Tables 1330

- How to Size Column Widths or Row Heights to Fit the Text 1424
- How to Sort the Grid based on the Summary Cell Values? 1333
- How to Stop the Errors Thrown when Pasting Larger Clipboard Contents in a GridDataBoundGrid 1273
- How to Support Tri-State Sorting in a GridDataBoundGridControl 1274
- How to Suppress KeyDown Event 1439
- How to Swap Rows and Columns 1229
- How to Synchronize the Selection of Multiple Grids 1230
- How to trigger an event when the ComboBox DropDownList has null values or has no datasource bound to it? 1443
- How to Update Summaries Immediately Upon Changing a Field 1332
- How to Use a ColorEdit Control in a Cell and Retrieve its Value 1235
- How to Use a ColorEdit Control in a Column and Retrieve its Value 1280
- How to Use a Combo Box in a Cell 1232
- How to Use a Combo Box in a Column 1277
- How to Use a PushButton in a Cell and Catch the User Clicking It 1236
- How to Use a PushButton in a Column and Catch the User Clicking It 1281
- How to Validate Changes Made to a Grid Cell 1425
- How to Write Syntax for Grid Model Properties 1426
- HYPGEOMDIST 299
- I**
- IBindingList 681
- IEEERemainder 300
- IF 300
- If CellType not specified 116
- If CellType specified 116
- If format not specified 115, 116, 117
- If format specified 115, 116, 117
- IFERROR 300
- IFNA 301
- IList 677
- IList Grouping Performance 669
- Image Cell Conversion 116
- ImageList 179
- Import an Excel Sheet into Essential Grid 117
- In-built Coded UI Support for 3.5 and 4.0 Frameworks 1177
- Index 301
- Indirect 302
- Initializing the Virtual Grid 102
- Inside Essential Grid's Formula Support 245
- Installation 36
- Installation and Deployment 36
- INT 302
- Integer Text Box 226
- INTERCEPT 303
- INTRATE 302
- Introduction to Essential Grid 21
- IPMT 303
- IRR 304
- IsBlank 311
- IsErr 305
- ISERROR 305
- ISEVEN 311
- IsLogical 313
- IsNA 305
- IsNonText 307
- ISNUMBER 305
- ISODD 312
- ISPMT 306
- ISREF 306
- IsText 307

ITypedList 682

**K**

Key Events 1057

KURT 307

**L**

LARGE 308

Layout and Appearance 1337

LCM 308

LEFT 309

LEN 310

Lesson 1: Grid Grouping Control Designer 44

Lesson 2: Grid Control Designer 77

Lesson 3: Grid Data Bound Grid Designer 81

Lesson 4: Virtual Grid Tutorial 97

Lesson 5: Excel Export 113

Lesson 6: Accessibility Information for Grid  
Windows Forms 122

Link Label Cell 216

List of Controls 130

List of Expressions 767

List of Filter Expressions 793

ListChanged Performance 633

ListItem Reference Relation 819

LN 309

Localization Support 1184

Localization Support for ComboBox Cell 1188

Localization Support for CompareOperatorListBox  
785

LOG10 311

Logest 312

LOGINV 313

LOGNORMDIST 314

Look and Feel 881

LOOKUP 315

Lower 316

**M**

Major Control Classes Overview 610

Making a Change to a ReadOnly Cell 396

Making a Grid Browse-Only 201

Managing Clipboard Operations with  
GridModelCutPaste 520

Managing Current Cell Operations 187

Masked Edit 162

MAX 316

MAXA 316

MDTERM 317

MEDIAN 317

Memory Performance - Engine Optimizations 616

Merge Cells Dynamically 872

Merge Cells Feature 510

Merging Cells Dynamically in a Grid 873

Metro Theme for Essential Grid Controls 1154

MID 318

MIN 318

MINA 318

MINUTE 319

MINVERSE 319

MIRR 320

MMULT 320

MOD 321

MODE 321

Model Based Selection 913

MONTH 322

Month Calendar 164

Mouse Events 1052

MouseDown Event 1054

MouseLeave Event 1055

MouseUp Event 1056

Moving Rows and Columns 472

MROUND 322

MS Excel 2007-like UI	514	<b>O</b>	
MS Excel-like Features	230	ODD	331
MS Office Simulation	143	Office2007Filter	1138
Multi Row Record	578, 901	Office2010 Theme in Windows Grids	1146
MultiLevel Undo and Redo	236	OG	310
Multinomial	323	OLE Drag-and-Drop	504
Multiple Document Interface (MDI) Support	518	Optimized GridExcelFilter	1140
Multiple Grid Printing	500	Optional Events	417
Multiple Headers	582	OR	331, 1333
Multiple Nested Relations	574	Outlook 2007 Demo	969
Multiple Record Selection	931	Overview	21
MUNIT	323	<b>P</b>	
<b>N</b>		Paging Support in GridGrouping Control	842
N	324	Paging support with a different data source	843
NA	324	PDF Converter	962
Named Ranges	381	PDF Export	405
Navigation Bar	985	PEARSON	331
NEGBINOMDIST	325	Percent Text Box	228
Nested Drop-down Grids	571	PERCENTILE	332
Nested Expression Fields	765	PERCENTILE.EXC	333
Nested Tables	1314	PERCENTILE.INC	333
NETWORKDAYS	325	PERCENTRANK	333
NORM.S.DIST	328	PERCENTRANK.EXC	334
NORM.S.INV	328	Performance	25, 525, 612
NORMDIST	326	PERMUT	334
NORMINV	327	PERMUTATIONA	335
NormsDist	327	PI	335
NormsInv	328	Picture Box	217
NOT	329	Pivot Grid	420
NOW	329	PMT	335
NPER	329	POISSON	336
NPV	330	Populating a Grid Control	386
Number Cell Conversion	115	Portfolio Grid	965
Number Formats	196	Pow	337
Numeric Up Down	165	POWER	337

PPMT 338	Record Navigation Bar 579
PrepareViewStyleInfo Event 492	RecordExpanding Event 1034
Preparing the Grid Application 1160	RecordFilters 771
Prerequisites and Compatibility 23	Records 1351
Print and Print Preview 987	RecordValueChanged Event 1035
Print Preview and Printing 493	RecordValueChanging Event 1034
Print Properties 447	Refreshing Behavior of Current Cell 188
Printing 1123	Related Master Details Relation 800
Printing Options 1431	Relations and Hierarchy 795
PROB 338	Render Images in Header Cells 861
PRODUCT 339	Required Events 414
Progress Bar 166	Resize To Fit 476
ProgressBar Cell Conversion 116	ResizeToFit Behavior in AutoSize 480
Properties 177, 1166	ResizeToFitOptimized 478
Push Button 168	Resizing Heights of Individual Rows in Grid 1096
PV 339	Rich Text 170
<b>Q</b>	RIGHT 345
QUARTILE 340	ROMAN 345
QueryCellInfo Event 415	ROUND 346
QueryColCount Event 415	ROUNDDOWN 346
QueryColWidth 418	ROUNDUP 347
QueryCoveredRange 419	RSQ 347
QueryRowCount Event 414	<b>S</b>
QueryRowHeight 417	Sample and Location 36
QUOTIENT 341	Sample Link 29, 200, 717
<b>R</b>	SaveCellInfo Event 417
RADIANS 342	Saving Edited Values 107
RAND 342	Scroll Bar Properties 451
RANDBETWEEN 342	Scroll Cell into View 186
RANK 343	SEARCH 348
RATE 343	Searching for text in all columns 1067
Real Time Applications 965	Searching for text in multiple columns 1066
Real-time Applications 513	Searching for text in particular columns 1067
RECEIVED 344	SEC 348
Record Based Selection 916	SECH 349

- SECOND 349
- Select Collection of Records In the GridGroupingControl 1063
- Selected Ranges Collection 936
- Selection Events for Filter Bar 1059
- Selection Frame 230
- Selection Modes 506
- Selections 913, 1348
- Serialization 400, 938
- Setting Column Styles and Row Styles 473
- Setting Column Widths and Row Heights 472
- Setting GridStyleInfo Properties 389
- Setting Properties in a Virtual Grid 110
- Setting ReadOnly Cell by Cell 396
- Setting ReadOnly Grid Wide 395
- Setting up Foreign Key Relations 1100
- Show or Hider Header 1428
- Show/Hide Current Cell Border 187
- SIGN 349
- SIN 350
- SINH 350
- SKEW 350
- SKEW.P. 351
- Slider 171, 218
- SLN 351
- SLOPE 352
- SMALL 352
- SOAP, Binary and XML Serialization 409
- Sort by DisplayMember 568
- Sort By Summary In Caption 758
- Sort Icon Placement 28
- SortedItemsInGroup Event 1037
- Sorting 27, 567, 736, 1374
- SortingItemsInGroup Event 1037
- SourceListChanged Event 1038
- Splitter 233
- SQRT 353
- SQRTPI 353
- Stacked MultiHeaders 892
- STANDARDIZE 354
- Static 172
- STDEV 354
- STDEV.S 357
- STDEVA 355
- STDEVP 355
- STDEVPA 356
- STEYX 357
- Strongly Typed Collections 686
- Style Properties 104, 189
- Styles At Group Level 856
- Styles At Table Level 853
- SUBSTITUTE 357
- SUBTOTAL 358
- Sum 359
- SUMIF 360
- SUMIFS 360
- Summaries 744, 1326
- Summary 27
- Summary In Caption 754
- SUMPRODUCT 361
- SUMSQ 361
- SUMX2MY2 362
- SUMX2PY2 362
- SUMXMY2 363
- Support for Skin Manager in Windows Forms Grid 1182
- Supported Arithmetic Operators and Calculation Precedence 244
- SYD 363

**T**

T 364

TabBarSplitterControl 490

Table Events 1022

Table Options 885

Tables for Properties and Events 1154

Tables for Properties, Methods, and Events 1143

TAN 365

TANH 365

Testing the Application with Generated Coded UI Tests 1162

TEXT 365

Text and CellValue 180

Text Box 174

The Basics 398

The Grid Control Indexer Technique 386

The GridControl.ChangeCells Method 185

The GridControl.PopulateValues Method 387

Through Code 136, 389, 394, 544, 594, 1074, 1086

Through Designer 133, 393, 394, 538, 594, 1069, 1084

Tile Image In Grid Cell 536

TIME 366

TIMEVALUE 366

TODAY 367

Transactions 399

Trim 367

TRIMMEAN 367

True 368

TRUNC 368

Tutorials 43

Type Conversions 113

**U**

Unbound Mode 696

Unhide Column by Double-Clicking Disabled 240

Uniform Child List Relation 827

Upper 368

Using a Master-Details Relation 561

Using the CurrencyManager 552

Using the Formula Library 244

Using the GridDataBoundGrid.Binder Class 560

Using the GridStyleInfo Class 389

Using the ReadOnly Attribute 395

Using Undo/Redo 397

**V**

VALUE 369

VAR 369

VARA 370

VARP 370

VARPA 371

VDB 372

Virtual Grids 413

VLOOKUP 373

**W**

WEEKDAY 373

WEIBULL 374

WEIBULL.DIST 375

What are the events fired when records are collapsing or collapsed? 1355

What are the events fired when the group is collapsing / collapsed? 1368

What are the events fired when the group is expanded / expanding? 1370

What are the events fired when the record values are changed? 1357

What are the events fired when the records are expanded / expanding? 1356

What are the events fired when the records are selected? 1350

What are the options in the summary columns? 1333

What are the various ExpressionColumn options?

1347

What is the Difference between TextAlign,  
HorizontalAlignment and VerticalAlignment?

1237

Why does the call to the Application.DoEvents  
never return while updating the Grid using the  
BeginUpdate() method? 1445

Word Converter 400, 960, 1127

Workbook 232

Working with Filters 789

Working with Group Elements 721

Working with Groups 728

Working with Relations 837

Working with Rows and Columns 467

**X**

XHTML Cell 220

XIRR 375

Xml Serialization 938

XOR 376

**Y**

YEAR 376

**Z**

Zoom Support 1190

Zooming Grid Controls 1192

Zooming Individual Cells of the Grid 1195

Zooming options 531

ZTEST 377