Developer's Guide

This guide provides information about using MultiRow for Windows Forms.

- System Requirements
- Notational Conventions
- Installation and Redistribution
- Licensing the Product
- Technical Support
- End-User License Agreement
- About MultiRow
- Using MultiRow
- Limitations
# Table of Contents

1. Table of Contents

1. System Requirements
2. Notational Conventions
3. Installation and Redistribution
4. Technical Support
5. Licensing the Product
6. End-User License Agreement
7. About MultiRow
8. Product Overview
9. Product Structure
10. Main Features
11. Feature Comparison
12. What's New
13. What's New in the Designer
14. What's New in the Grid Control
15. New Cell Types
16. Breaking Changes
17. Compatibility with the Previous Version
18. Breaking Changes from 6.0
19. Co-existence and Migration from the Previous Version
20. Coexistence with the Previous Version
21. Migration from the Previous Version
22. Using MultiRow
23. MultiRow Architecture
24. Basic Concepts
25. Template Structure
26. Template File
## MultiRow Windows Forms Developer's Guide

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell Structure</td>
<td>51-56</td>
</tr>
<tr>
<td>Format String</td>
<td>56</td>
</tr>
<tr>
<td>Quick Start</td>
<td>56-57</td>
</tr>
<tr>
<td>Adding MultiRow Component and Template to the Project</td>
<td>57-61</td>
</tr>
<tr>
<td>Adding the MultiRow Component to a Visual Studio 2013 Project</td>
<td>61-65</td>
</tr>
<tr>
<td>Creating an Application using GcMultiRow</td>
<td>65-66</td>
</tr>
<tr>
<td>Connecting to SQL Server Database</td>
<td>66-73</td>
</tr>
<tr>
<td>Creating an Application Using the Designer (Grid View)</td>
<td>73-81</td>
</tr>
<tr>
<td>Creating an Application Using the Designer (Single Template View)</td>
<td>81-89</td>
</tr>
<tr>
<td>Designer Window</td>
<td>89</td>
</tr>
<tr>
<td>Integration with IDE</td>
<td>89-90</td>
</tr>
<tr>
<td>Document Window</td>
<td>90-91</td>
</tr>
<tr>
<td>Toolbox</td>
<td>91-92</td>
</tr>
<tr>
<td>Menu Bar</td>
<td>92-93</td>
</tr>
<tr>
<td>Toolbar</td>
<td>93</td>
</tr>
<tr>
<td>Status Bar</td>
<td>93-94</td>
</tr>
<tr>
<td>Properties Window</td>
<td>94</td>
</tr>
<tr>
<td>Template Property List 7.0</td>
<td>94-97</td>
</tr>
<tr>
<td>Template Explorer 7.0</td>
<td>97-98</td>
</tr>
<tr>
<td>Template - Named Cell Styles 7.0</td>
<td>98-99</td>
</tr>
<tr>
<td>Designer Options</td>
<td>99-102</td>
</tr>
<tr>
<td>Using the Designer</td>
<td>102-103</td>
</tr>
<tr>
<td>Changing Height and Width of a Row in Designer</td>
<td>103</td>
</tr>
<tr>
<td>Working with Column Headers in the Designer</td>
<td>103-104</td>
</tr>
<tr>
<td>Working with Column Footers in the Designer</td>
<td>104</td>
</tr>
<tr>
<td>Adding Cells in the Designer</td>
<td>104-105</td>
</tr>
<tr>
<td>Replacing Cells in the Designer</td>
<td>105-106</td>
</tr>
<tr>
<td>Moving Cells in the Designer</td>
<td>106</td>
</tr>
<tr>
<td>Copying Cells in the Designer</td>
<td>106-107</td>
</tr>
<tr>
<td>Selecting Cells in the Designer</td>
<td>107-108</td>
</tr>
</tbody>
</table>
Resizing Cells in the Designer
108
Get or Set Cell Properties in the Designer
108-109
Changing Cell Index in the Designer
109-110
Undo Changes in Designer
110-111
Locking Cells in the Designer
111
Changing Tab Order of Cells in the Designer
111-112
Display Modes of the Designer
112-113
Cell Display Modes in the Designer
113-114
Layout Modes of the Designer
114-115
Setting the Column Mode Template in the Designer
115
Advanced Designer Features
115-116
Create Template using Wizard
116-118
Create Template from Data Sources Window
118-120
Create Template from Table
120-125
Create Templates Using Tracing Mode
125-126
Adjust Character Size with Placeholders
126-127
Template Localization
127-130
Hints for using the Designer
131-132
Conversion from DataGridView
132-136
Using Annotations
136-138
Template
138
Adding Cells
138-139
Adding Column Headers
139-140
Adding Column Footers
140-142
Template Width and Height
142-144
Column Mode Template
144-148
Grid
148-149
Assigning and Allocating a Template
149-150
Edit Modes
150-153
Selection Mode
153-156
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>View Modes</td>
<td>156-159</td>
</tr>
<tr>
<td>Scrolling</td>
<td>159-162</td>
</tr>
<tr>
<td>Visual Style</td>
<td>162-163</td>
</tr>
<tr>
<td>Zooming</td>
<td>163-166</td>
</tr>
<tr>
<td>Splitter Line and ViewPort</td>
<td>166-169</td>
</tr>
<tr>
<td>Context Menu</td>
<td>169-171</td>
</tr>
<tr>
<td>Shortcut Keys</td>
<td>171-176</td>
</tr>
<tr>
<td>Resizing</td>
<td>176-177</td>
</tr>
<tr>
<td>Retrieving the Method Used to Move Cells</td>
<td>177-178</td>
</tr>
<tr>
<td>Automatic Merging of Cells</td>
<td>178-180</td>
</tr>
<tr>
<td>Displaying a New Row at the Top</td>
<td>180-181</td>
</tr>
<tr>
<td>Expanding Or Collapsing Columns</td>
<td>181-182</td>
</tr>
<tr>
<td>Automatic Adjustment of the Cell Width</td>
<td>182-183</td>
</tr>
<tr>
<td>Points to Note with Default Settings</td>
<td>183-185</td>
</tr>
<tr>
<td>Tips on Improving the Performance</td>
<td>185-189</td>
</tr>
<tr>
<td>Headers</td>
<td>189</td>
</tr>
<tr>
<td>Selecting Cells using Headers</td>
<td>189-190</td>
</tr>
<tr>
<td>Filtering Rows using the Column Headers</td>
<td>190-194</td>
</tr>
<tr>
<td>Sorting Rows using Column Headers</td>
<td>194-197</td>
</tr>
<tr>
<td>Changing the Header Appearance</td>
<td>197-198</td>
</tr>
<tr>
<td>Using Row Headers</td>
<td>198-199</td>
</tr>
<tr>
<td>Cell Types</td>
<td>199-200</td>
</tr>
<tr>
<td>Base Cell</td>
<td>200-201</td>
</tr>
<tr>
<td>HeaderCell</td>
<td>201-203</td>
</tr>
<tr>
<td>Change BackColor (HeaderCell)</td>
<td>203-204</td>
</tr>
<tr>
<td>ColumnHeaderCell</td>
<td>204-206</td>
</tr>
<tr>
<td>RowHeaderCell</td>
<td>206-208</td>
</tr>
<tr>
<td>CornerHeaderCell</td>
<td>208</td>
</tr>
<tr>
<td>TextBoxCell</td>
<td>208-212</td>
</tr>
</tbody>
</table>
Input Line Break (TextBoxCell) 212-213
How to Use KeyDown Event (TextBoxCell) 213-214
Display Vertical Scroll Bar (TextBoxCell) 214-215

ButtonCell 215-218
Change BackColor (ButtonCell) 218-219
Handling User Click (ButtonCell) 219-220
Display String (ButtonCell) 220-221
Using Button Commands (ButtonCell) 221-222

CheckBoxCell 222-224
Refer to Editing Value when Modifying TrueValue and FalseValue (CheckBoxCell) 224-225
Detect Value Change (CheckBoxCell) 225

ComboBoxCell 225-229
Get Index Of Selected Value (ComboBoxCell) 229-230
How to Use SelectedIndexChanged Event (ComboBoxCell) 230-231

MaskedTextBoxCell 231-235
DateTimePickerCell 235-239
Display Alternate Text for Null Values (DateTimePickerCell) 239-240
Hide Drop-Down Button (DateTimePickerCell) 240-241
Get Formatted Values (DateTimePickerCell) 241-242
Automatically Confirm Initial Value as Input Value (DateTimePickerCell) 242
Usage of ValueChanged Event (DateTimePickerCell) 242-243

NumericUpDownCell 243-246
Displaying Negative Values in Red (NumericUpDownCell) 246-248
Hiding Spin Button (NumericUpDownCell) 248-249
Displaying Numeric Value Separator (NumericUpDownCell) 249-250

DomainUpDownCell 250-253

LabelCell 253-255
Handling User Click (LabelCell) 255-256
Display String (LabelCell) 256-257
LinkLabelCell 257-259
  Open Link in Associated Browser (LinkLabelCell) 259
ImageCell 259-261
  Display Alternate Image for Empty Value (ImageCell) 261-262
  Handle User Click (ImageCell) 262-263
  Display Image (ImageCell) 263
ProgressBarCell 263-266
RichTextBoxCell 266-269
RadioGroupCell 269-272
  Detect Value Changed (RadioGroupCell) 272-273
ListBoxCell 273-275
ListLabelCell 275-277
  Setting Bullets (ListLabelCell) 277-278
ShapeCell 278-280
  Setting Shapes (ShapeCell) 280-281
Pop-upCell 282-284
  Select Color (Pop-upCell) 284-285
SummaryCell 285-288
  Show Column Sum in Footer (SummaryCell) 288-290
  Displaying an Error Icon for Data Errors (SummaryCell) 290-291
  Create Statement of Account (SummaryCell) 291-294
PrintInfoCell 294-296
TrackBarCell 296-299
FilteringTextBoxCell 299
User-Defined Cell 299-302
  Register User Defined Cell in ToolBox 302
Rows and Cells 302-303
  Current Row 303-304
  Current Cell 304-305
  Adding Rows 305-307
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deleting Rows</td>
<td>307-309</td>
</tr>
<tr>
<td>Getting and Setting the Row Count</td>
<td>309-310</td>
</tr>
<tr>
<td>Freezing Rows</td>
<td>310-312</td>
</tr>
<tr>
<td>Selected Rows and Cells</td>
<td>312-314</td>
</tr>
<tr>
<td>Restricting Cell Selection</td>
<td>314-315</td>
</tr>
<tr>
<td>Disabled Cells</td>
<td>315</td>
</tr>
<tr>
<td>Hiding Cells and Rows</td>
<td>315-317</td>
</tr>
<tr>
<td>Editing Cells</td>
<td>317-319</td>
</tr>
<tr>
<td>Displaying Error Messages</td>
<td>319-320</td>
</tr>
<tr>
<td>Read-Only Cells</td>
<td>320-321</td>
</tr>
<tr>
<td>Cell Note</td>
<td>321-325</td>
</tr>
<tr>
<td>Column Mode</td>
<td>325</td>
</tr>
<tr>
<td>Current Column</td>
<td>325-326</td>
</tr>
<tr>
<td>Adding Columns</td>
<td>326-329</td>
</tr>
<tr>
<td>Deleting Columns</td>
<td>329-332</td>
</tr>
<tr>
<td>Column Operation</td>
<td>332-334</td>
</tr>
<tr>
<td>Managing Data</td>
<td>334-335</td>
</tr>
<tr>
<td>Get and Set Data</td>
<td>335-338</td>
</tr>
<tr>
<td>Data Binding Modes</td>
<td>338-340</td>
</tr>
<tr>
<td>Sorting Data</td>
<td>340-343</td>
</tr>
<tr>
<td>Working with Clipboard</td>
<td>343-344</td>
</tr>
<tr>
<td>Set Default Value in New Row</td>
<td>344-345</td>
</tr>
<tr>
<td>Cell Styles</td>
<td>345</td>
</tr>
<tr>
<td>Rules for Applying Styles</td>
<td>345-347</td>
</tr>
<tr>
<td>Borders</td>
<td>347-350</td>
</tr>
<tr>
<td>BackColor and ForeColor</td>
<td>350-352</td>
</tr>
<tr>
<td>Background Pattern</td>
<td>352-354</td>
</tr>
<tr>
<td>Background Gradation</td>
<td>354-356</td>
</tr>
<tr>
<td>Tag</td>
<td>356-358</td>
</tr>
<tr>
<td>Image</td>
<td>358-359</td>
</tr>
<tr>
<td>Section</td>
<td>Pages</td>
</tr>
<tr>
<td>--------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>Margins and Padding</td>
<td>359-360</td>
</tr>
<tr>
<td>Multiline and Word Wrap</td>
<td>360-361</td>
</tr>
<tr>
<td>Text Formatting</td>
<td>361-362</td>
</tr>
<tr>
<td>Text Indent</td>
<td>362-363</td>
</tr>
<tr>
<td>GDI+ Compatibility Mode</td>
<td>363-366</td>
</tr>
<tr>
<td>Mouse Hover Styles</td>
<td>366-367</td>
</tr>
<tr>
<td>User-defined Word Wrap</td>
<td>367-368</td>
</tr>
<tr>
<td>Customizing the Justification of Single-byte Characters</td>
<td>368-369</td>
</tr>
<tr>
<td>Conditional Cell Styles</td>
<td>369</td>
</tr>
<tr>
<td>Types of Cell Styles</td>
<td>369-371</td>
</tr>
<tr>
<td>Manage Cell Styles with Names</td>
<td>371-374</td>
</tr>
<tr>
<td>Changing Cell Styles based on Conditions</td>
<td>374-376</td>
</tr>
<tr>
<td>Combining Multiple Styles</td>
<td>376-377</td>
</tr>
<tr>
<td>User Defined Conditional Styles</td>
<td>377-378</td>
</tr>
<tr>
<td>Manage Cell Styles with Names</td>
<td>378-382</td>
</tr>
<tr>
<td>User Input Validation</td>
<td>382</td>
</tr>
<tr>
<td>Validation using Control Events</td>
<td>382-383</td>
</tr>
<tr>
<td>Value Validation by Cell</td>
<td>383</td>
</tr>
<tr>
<td>Validation</td>
<td>383-384</td>
</tr>
<tr>
<td>Built-in Cell Validators</td>
<td>384</td>
</tr>
<tr>
<td>RangeValidator</td>
<td>384-387</td>
</tr>
<tr>
<td>IncludeListValidator</td>
<td>387-388</td>
</tr>
<tr>
<td>RequiredTypeValidator</td>
<td>388-390</td>
</tr>
<tr>
<td>RequiredFieldValidator</td>
<td>390-391</td>
</tr>
<tr>
<td>ExcludeListValidator</td>
<td>391-392</td>
</tr>
<tr>
<td>CompareCellValidator</td>
<td>392-393</td>
</tr>
<tr>
<td>CompareValueValidator</td>
<td>393-394</td>
</tr>
<tr>
<td>PairCharValidator</td>
<td>394-396</td>
</tr>
<tr>
<td>SurrogateCharValidator</td>
<td>396-397</td>
</tr>
<tr>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>RegularExpressionValidator</td>
<td>397-398</td>
</tr>
<tr>
<td>CompareStringValidator</td>
<td>398-399</td>
</tr>
<tr>
<td>TextLengthValidator</td>
<td>399-400</td>
</tr>
<tr>
<td>EncodingValidator</td>
<td>400-401</td>
</tr>
<tr>
<td><strong>Built-in Validation Actions</strong></td>
<td>401</td>
</tr>
<tr>
<td>Notification using Underline (LineNotify)</td>
<td>401-402</td>
</tr>
<tr>
<td>Notification using CellStyle (CellStyleNotify)</td>
<td>402-403</td>
</tr>
<tr>
<td>Notification using Icon (IconNotify)</td>
<td>403-404</td>
</tr>
<tr>
<td>Notification using Sound (SoundNotify)</td>
<td>404-405</td>
</tr>
<tr>
<td>TipNotify</td>
<td>406-407</td>
</tr>
<tr>
<td>Notification using Focus Change (FocusProcess)</td>
<td>407-408</td>
</tr>
<tr>
<td>ValueProcess</td>
<td>408-409</td>
</tr>
<tr>
<td>Notification using Message Box (MessageBoxNotify)</td>
<td>409-410</td>
</tr>
<tr>
<td>Notification using Three-State Icon</td>
<td>410-411</td>
</tr>
<tr>
<td><strong>Customizing the Validation Information</strong></td>
<td>411-412</td>
</tr>
<tr>
<td>User-Defined Cell Validator</td>
<td>412-413</td>
</tr>
<tr>
<td>User Defined Validation Actions</td>
<td>413-416</td>
</tr>
<tr>
<td><strong>Connecting to Database</strong></td>
<td>416-419</td>
</tr>
<tr>
<td><strong>Print and Print Preview</strong></td>
<td>419-420</td>
</tr>
<tr>
<td>Simple Printing</td>
<td>420-421</td>
</tr>
<tr>
<td>Custom Printing</td>
<td>422</td>
</tr>
<tr>
<td>Print Style</td>
<td>422-424</td>
</tr>
<tr>
<td>Paging Mode</td>
<td>424-425</td>
</tr>
<tr>
<td>Specify Print Range</td>
<td>425-426</td>
</tr>
<tr>
<td>Specifying the Page Break Position</td>
<td>426-427</td>
</tr>
<tr>
<td>Specifying Empty Rows for Print</td>
<td>427</td>
</tr>
<tr>
<td>Printing Watermarks</td>
<td>427-430</td>
</tr>
<tr>
<td>Print Settings for each Column Header Section</td>
<td>430-431</td>
</tr>
<tr>
<td><strong>File Input and Output</strong></td>
<td>431</td>
</tr>
<tr>
<td>Input and Output of Template XML</td>
<td>431-432</td>
</tr>
</tbody>
</table>
Multi-touch Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zooming</td>
<td>432-436</td>
</tr>
<tr>
<td>Scrolling</td>
<td>436-437</td>
</tr>
<tr>
<td>Cell Selection</td>
<td>437-439</td>
</tr>
<tr>
<td>Resizing the Column Width</td>
<td>439-440</td>
</tr>
<tr>
<td>Touch Toolbar</td>
<td>440-447</td>
</tr>
<tr>
<td>Touch Keyboard</td>
<td>447-448</td>
</tr>
<tr>
<td>Gripper during Cell Editing</td>
<td>448-449</td>
</tr>
<tr>
<td>Zoomed Display of the Drop-down List</td>
<td>449-450</td>
</tr>
<tr>
<td>Retrieving the Input Device</td>
<td>450-451</td>
</tr>
</tbody>
</table>

Troubleshooting

<table>
<thead>
<tr>
<th>Issue</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unexpected License Dialog Gets Displayed</td>
<td>451-452</td>
</tr>
<tr>
<td>Template Added in Project Does Not Display in Smart Tag</td>
<td>452</td>
</tr>
<tr>
<td>Unable to Select MathStatistics Object in SummaryCell</td>
<td>452-453</td>
</tr>
<tr>
<td>Unable to Modify BackColor of Button and Header</td>
<td>453</td>
</tr>
<tr>
<td>Unable to Edit Cell Placed in Column Header</td>
<td>453</td>
</tr>
<tr>
<td>Value in EditedFormattedValue Property and the Value Displayed Are Different</td>
<td>453</td>
</tr>
<tr>
<td>User-Defined Cell Added in Project Does Not Appear in Toolbox</td>
<td>453-454</td>
</tr>
<tr>
<td>Vertical Alignment and Equal Distribution Do Not Get Set for the Cell Text</td>
<td>454</td>
</tr>
<tr>
<td>Cannot Find Table Information in Template Source Code</td>
<td>454</td>
</tr>
<tr>
<td>Cell Validation Result Is Not Visible</td>
<td>454</td>
</tr>
<tr>
<td>Cell.DesignTimeValue Property Gives a Compile Error</td>
<td>454</td>
</tr>
<tr>
<td>Error &quot;LC.exe&quot; Exited with Code -1 Occurs While Building</td>
<td>454</td>
</tr>
<tr>
<td>Flickering Display of Help</td>
<td>454-455</td>
</tr>
<tr>
<td>Unable to Sort (Filter) Last Row</td>
<td>455</td>
</tr>
<tr>
<td>Displaying Bound Data in a Cell Placed in the Column Header Section</td>
<td>455</td>
</tr>
</tbody>
</table>

Limitations

2. Index

456-458

459-465
System Requirements

The following hardware and software are required to use MultiRow Windows Forms (MultiRow):

**Hardware**

Hard disk drive with more than 90MB of free space

**Software**

MultiRow can be used in the following development environments:

- **Development Language**: Visual Basic, C#
- **OS**

<table>
<thead>
<tr>
<th></th>
<th>32-bit (x86)</th>
<th>64-bit (x64)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 8, Windows 8.1</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Windows 7</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Windows XP</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Windows Vista</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Windows Server 2003</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Windows Server 2008</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Windows Server 2008 R2</td>
<td>-</td>
<td>●</td>
</tr>
</tbody>
</table>

The Designer does not work in the Visual Studio Express Edition. MultiRow runtime can be used by redistributing it on the following deployment environments:

- **Framework**: .NET Framework 2.0 SP1/3.0 SP1/3.5/3.5 Client Profile/4/4 Client Profile
- **OS**

<table>
<thead>
<tr>
<th></th>
<th>32-bit (x86)</th>
<th>64-bit (x64)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 8, Windows 8.1</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Windows 7</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Windows XP</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Windows Vista</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Windows Server 2003</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Windows Server 2008</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Windows Server 2008 R2</td>
<td>-</td>
<td>●</td>
</tr>
</tbody>
</table>

**64-Bit Environment**

Note the following limitations:

- MultiRow has common assemblies for 32-bit and 64-bit versions.
- IA-64 is not supported.

**.NET Framework 4/4 Client Profile**

Multirow runtime for .NET Framework 4/4 Client Profile is the same as the one for earlier .NET Frameworks.

**.NET Framework 3.5 Client Profile**

It is not supported in .NET Framework 3.5 Client Profile.

**Visual Studio Express Editions**

Only the templates and the GeMultiRow control can be used in Visual Studio 2008 Express Edition or Visual Studio 2010 Express. The template needs to be created through code, since the designer for GUI designing cannot be used for the templates. This limitation is due to license related constraints as well as technical constraints (extension of IDE is not supported in Express Edition). Please refer to the End User License Agreement (EULA) of Express Edition for more details on these constraints. Please use Visual Studio Standard Edition and above, in order to be able to use all the features of MultiRow.

**Windows 2000**

MultiRow has not been thoroughly tested on Windows 2000.

**Windows Server 2003 R2**

Windows Server 2003 R2 is supported in the same way as Windows Server 2003.

**Service Pack Of OS**

Unless specifically mentioned, there is no dependency on the service pack of OS.
Notational Conventions

This topic describes the notational conventions of MultiRow.

Notation of DataType

This help file uses the Visual Basic data types. The sample code uses types for that language. The following chart lists type differences between different languages.

<table>
<thead>
<tr>
<th>CLR</th>
<th>Visual Basic</th>
<th>C#</th>
</tr>
</thead>
<tbody>
<tr>
<td>System.Byte</td>
<td>Byte</td>
<td>byte</td>
</tr>
<tr>
<td>System.SByte</td>
<td>Sbyte</td>
<td>sbyte</td>
</tr>
<tr>
<td>System.Int16</td>
<td>Short</td>
<td>short</td>
</tr>
<tr>
<td>System.Int32</td>
<td>Integer</td>
<td>int</td>
</tr>
<tr>
<td>System.Int64</td>
<td>Long</td>
<td>long</td>
</tr>
<tr>
<td>System.UInt16</td>
<td>UInt16</td>
<td>ushort</td>
</tr>
<tr>
<td>System.UInt32</td>
<td>UInt32</td>
<td>uint</td>
</tr>
<tr>
<td>System.UInt64</td>
<td>UInt64</td>
<td>ulong</td>
</tr>
<tr>
<td>System.Single</td>
<td>Single</td>
<td>float</td>
</tr>
<tr>
<td>System.Double</td>
<td>Double</td>
<td>double</td>
</tr>
<tr>
<td>System.Decimal</td>
<td>Decimal</td>
<td>decimal</td>
</tr>
<tr>
<td>System.Boolean</td>
<td>Boolean</td>
<td>bool</td>
</tr>
<tr>
<td>System.Char</td>
<td>Char</td>
<td>char</td>
</tr>
<tr>
<td>System.Decimal</td>
<td>Decimal</td>
<td>decimal</td>
</tr>
<tr>
<td>System.IntPtr</td>
<td>IntPtr</td>
<td>IntPtr</td>
</tr>
<tr>
<td>System.UIntPtr</td>
<td>UIntPtr</td>
<td>UIntPtr</td>
</tr>
<tr>
<td>System.Object</td>
<td>Object</td>
<td>object</td>
</tr>
<tr>
<td>System.String</td>
<td>String</td>
<td>string</td>
</tr>
</tbody>
</table>

Sample Code

Sample code for MultiRow is provided in Visual Basic and C#. In most cases, you can paste the sample code in a standard Windows application project created in Visual Studio and run it. If a namespace is specified at the beginning of the code, add that section to the beginning of the source file. For example, the steps to create a simple project to run the sample code are as follows.

1. Select File - New - Project from Visual Studio.
2. Create a Windows Application in Visual Basic or C# from the New Project dialog.
3. Select the GcMultiRow control from the Visual Studio toolbox and place it on the form.
4. Select a Button control from the Visual Studio toolbox and place it on the form.
5. Double-click on the Button control placed on the form and paste the sample code.
6. Run the project by pressing the F5 key.

Generate an event handler in Visual Studio before using any code related to event handlers. Also, make sure you link the procedure to control events using the Properties window.

Illustrations and Screenshots
Illustrations are used to provide an overview of the control's behavior and does not necessarily indicate the functional results.

The screenshots are examples of results when working with the product on a specific OS. The working results may differ based on the features provided by the OS and user settings.

Links to Web Sites

This help includes links to Websites as reference information. In most cases the Web link icon is displayed in front of a web site link as shown below.

- 🌐 GrapeCity Inc.

It is possible that certain URL links might have changed and you might not be able to browse the respective information.

Registered Trademarks or Trademarks

- GrapeCity, PowerTools, MultiRow are registered trademarks or trademarks of GrapeCity inc.
- Microsoft Windows, Windows Vista, Visual Studio and Visual Basic are either registered trademarks or trademarks of Microsoft Corporation in United States and/or other countries.
- Other company names or products names mentioned are registered trademarks or trademarks of various companies.
Installation and Redistribution

Installation and redistribution instructions are provided in the Read Me file that accompanies this product. To view the Read Me file, do one of the following:

1. From the Start menu choose Programs->GrapeCity->Spread Studio 9->Read Me. Select the Read Me under the GrapeCity name on the Start screen with Microsoft Windows 8 or 8.1.
2. If you performed a default installation, in Windows Explorer browse to \Program Files\GrapeCity\Spread Studio 9\Docs and double-click the Readme.chm file.
Technical Support

If you have a technical question about this product, consult the Product forum at http://sphelp.grapecity.com/forums/forum/spreadsheets/.

If you are unable to find the answer using these sources, please contact Technical Support using one of these methods:

Web site http://sphelp.grapecity.com/
E-mail spread.support@grapecity.com
Phone (412) 681-4738
Fax (412) 681-4384

Technical Support is available between the hours of 9:00 a.m. and 5:00 p.m. Eastern time, Monday through Friday.
Licensing the Product

To license Windows Forms projects made with the trial version do the following:

1. Ensure that MultiRow is licensed on the machine by following the installation steps in the Read Me.
2. Open the project in Microsoft Visual Studio.
3. Open the Visual Studio Build menu and select Rebuild Solution.
4. The executable application is now licensed and no nag screens or evaluation banners appear when you run it. You can distribute the application to unlicensed machines and no nag screens or evaluation banners appear.
End-User License Agreement

The GrapeCity licensing information, including the GrapeCity end-user license agreements, frequently asked licensing questions, and the GrapeCity licensing model, is available online at http://spread.grapecity.com/Pages/Licensing-FAQs/ and http://spread.grapecity.com/Pages/EULA/.
About MultiRow

The following topics provide more information about MultiRow.

- **Product Overview**
- **Product Structure**
- **Main Features**
- **Feature Comparison**
- **What's New**
- **Breaking Changes**
- **Co-existence and Migration from the Previous Version**

## Product Overview

MultiRow provides a grid control for display and input of data in a tabular form. You can use MultiRow to design tables as needed and easily embed complex tables displaying one record in multiple rows, or a detailed display screen. Table design can be easily created with the help of a mouse using a designer that is integrated into Visual Studio. This is the same as in the case of forms.

In addition to basic features such as printing, exporting or input validation, inheritance provides you extendibility. For example, you can create your own cells or commands and use them. MultiRow supports ADO.NET data sources and hence you can connect to databases such as SQL Server using the data connection feature of Visual Studio. Also, you can use it in unbound mode, when it is not connected to a data source.

MultiRow deviates from the concept of a conventional grid control, and allows developers to implement their own ideas and provides usability to the end users. There are numerous scenarios of enhancing productivity using MultiRow. Use MultiRow to improve application usability and enhance development efficiency.

### Enhancing End User Usability

- Enhance visibility by displaying a single record in multiple rows
- Reduce learning cost by replicating the display of a paper form on an input screen
- Increase efficiency with a large number of edits by controlling IME input mode or conversion mode
- Fine control of the changing focus with respect to the user’s input results or business rules

### Enhancing Development Capabilities

- Structure that separates the grid design and logic
- Share created templates among the development team
- Maintain the grid layout separately from the application form
- Make the most of development experience since the cell type inherits from a standard control
- Use format settings of the .NET Framework as is
- Use the print feature of the .NET Framework
- Reuse common features by consolidating them in user defined cells
- Encapsulate and reuse data validation rules

### Numerous Features and Behavior Similar to the Standard Grid Control

- Reuse code created in standard control
- Access rows or cells using the same method as the standard control
- Use spin button or drop-down calendar while editing cells
- Switch IME mode separately for each cell
- Allow string rotation, vertical alignment, and equal distribution
- Change borders separately for each cell
- Precisely specify border style or color for each cell
- Create rounded borders or cross borders
- Print grid content
- Zoom-in and zoom-out of grid content
- Display sum value or average value in footer

**Compatibility With Other Controls**

- Reuse code created with standard control
- Mutually coordinate with standard control

**Product Structure**

MultiRow consists of a grid control, designer, samples, help, and so on.

**Control**

<table>
<thead>
<tr>
<th>Reference Name</th>
<th>Control Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multirow</td>
<td>GcMultiRow</td>
<td>1 record need not be confined to 1 row but can be displayed easily in multiple rows</td>
</tr>
<tr>
<td>Template</td>
<td>Template</td>
<td>Design base for the grid control</td>
</tr>
</tbody>
</table>

**Designer**

The designer is a design screen for the template that integrates with Visual Studio. The templates can be designed using the toolbox and **Properties** window of Visual Studio (which is commonly used by developers).

**Samples**

Samples are a set of programs that demonstrate using MultiRow. The same project has been created in two languages: Visual Basic and C#. Sample help that describes the purpose and usage of the samples, has also been included.

**Product Help**

This is what you are looking at right now. Information required for using MultiRow is given in this help.

**Release Notes**

The release notes are located in the installation folder of the product. They provide information such as the installation. We recommend that you read the release notes.

**Migration Tool**

The migration tool is a tool that converts a project file created in MultiRow 6.0, to a file that can be used in 7.0.

**Main Features**

MultiRow allows you to define the grid layout in a separate object, called a template. The Grid control creates an editable tabular screen at runtime, based on the defined template and data source. You can define your layout without being confined by rows and columns. Even with a complex layout with one record mapping to multiple rows, the developer can easily manage all the rows.
Make the most of the Visual Studio Experience in the Template Designer

- Since the Designer operates in Visual Studio, you can design tabular screens just like you design your forms.
- Easy maintenance due to the bifurcation, that is, design in template and implement logic in grid control on the form.
- With Runtime and Print preview, you can verify the input operations and print the output of the tabular screen, without having to build the project.
- Efficiently create layouts by making use of snap lines, zooming, tables, and wizards.
- With a window that displays cell information in the form of a tree and list, it is simple to compare and integrate the settings.

Your Own Layout

- Specify the desired size and position for the cell.
- Possible to overlap the cells.
- Define different layouts for headers, rows, and footers.
- Specify permission for cell selection and cell movements using properties.
- Allows inheriting and sharing of cell styles.
- Supports rich text alignment features such as equal distribution, vertical alignment, rotation, and so on.
- User can specify the sequence of cell movement using tab order.
Wide Range of Borders

Border roundness that can be varied for each cell, cross borders with diagonal lines, 3D borders showing textured effects, and so on, borders that are desired in tabular screens, are all available here. You can create designs that emulate paper forms and provide user friendly input screens to end users.

High Flexibility

A template created in the designer can be maintained as Visual Basic and C# source code, similar to a form, and can be easily managed and shared for debugging and team development, similar to form source code. You can create a user-defined cell by inheriting a built-in cell and then reusing it in various projects. This provides flexibility for project scalability and its long term maintenance.

Lightweight Performance

MultiRow performance has been optimized by assuming the architecture of displaying one record in multiple rows. The loading into the form or display of datasource have lightweight implementations.

Changeover from Standard Controls

MultiRow supports ADO.NET data binding and works in the same way as the standard .NET Framework controls. Since interfaces that emphasize compatibility with standard controls have been implemented for cell editing and events, you can use Multirow in the same way as you would use a DataGridView or TextBox.

Easy Deployment
When deploying an application created using MultiRow, there is no special agreement or additional cost. You just need a few megabytes of runtime files as the redistributable files.

## Feature Comparison

Below is a feature comparison list of the DataGridView control and MultiRow. Features can be used with the .NET Framework 2.0 or later.

### Feature Comparison List

<table>
<thead>
<tr>
<th>Feature</th>
<th>DataGridView</th>
<th>MultiRow Windows Forms 6.0</th>
<th>MultiRow Windows Forms 7.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Templates</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Grid</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Designer</td>
<td>Dialog</td>
<td>Document Window</td>
<td>Document Window</td>
</tr>
<tr>
<td>Column Header</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Column Footer</td>
<td>×</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Row Header</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Row Footer</td>
<td>×</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Row</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Cell</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Cell Type</td>
<td>Type</td>
<td>6</td>
<td>23</td>
</tr>
<tr>
<td>Button</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>CheckBox</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>ComboBox</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Image</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Link Label</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>String</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Numeric</td>
<td>×</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Date</td>
<td>×</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Mask</td>
<td>×</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>RichText</td>
<td>×</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Radio Group</td>
<td>×</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Progress Bar</td>
<td>×</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Print Information</td>
<td>×</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>
Aggregate  ×  ○  ○
PopUp  ×  ○  ○
Track Bar  ×  ○  ○
List Box  ×  ×  ○
List Label  ×  ×  ○
Shape  ×  ×  ○
InputManCell  ×  ○ ※2  ○
PlusPakcell  ×  ○ ※3  ○
User Defined  ○  ○  ○

Data Bound  ○  ○  ○
Unbound  ○  ○  ○
Virtual Mode  ○  ○  ○
Expand data horizontally  ×  ×  ○
Shared rows  ○  ×  ×
Sort  ○  ○  ○
Filter  ×  ○  ○
Auto Merging Cells  ×  ×  ○
Print  ×  ○  ○
Zoom  ×  ○  ○
Separator  ×  ○  ○

Frozen Row/Column  Top and Left  Top-Bottom/Left-Right  Top-Bottom/Left-Right

Column Sorting  ○  ×  ○ ※4
Display/Hide Columns  ○  ×  ○

Error Icon
Cell Style  ○  ○  ○
Named Style  ×  ○  ○
Conditional Style  ×  ○  ○

Border  Scope
Line Types  Section  1 Type  13 Types
Line Color  Each Control  Around Each Cell  Around Each Cell
3D  ○  ○  ○
**Rounded Corner**  ×  ○  ○  
**Cross**  ×  ○  ○  
**Vertical Alignment Of Text**  ×  ○  ○  
**Text Rotation**  ×  ○  ○  
**IME Input Mode**  For each Control  For each Cell  For each Cell  
**IME Conversion Mode**  ×  ○  ○  
**Value Validation**  For each Control  For each Control and each Cell  For each Control and each Cell  
**Touch Features**  ×  ○  ×  ○  
**Zoom**  ×  ○  ×  ○  
**Setting Minimum Zoom Level**  ×  ○  ×  ○  
**Snap Point**  ×  ○  ×  ○  
**Scroll**  ×  ×  ×  ○  
**Display Gripper**  ×  ×  ×  ○  ※5  
**Display Resize mark**  ×  ×  ×  ○  
**Display/hide Touch Keyboard**  ×  ×  ×  ○  
**Expanded Display of Drop Down List**  ×  ×  ×  ○  
**Touch Toolbar**  ×  ×  ×  ○  
**Retrieve Input Device**  ×  ×  ×  ○  

※1 If InputManCell (7 types) and PlusPakCell (5 types) are included.
※2 License of InputMan for Windows Forms is required.
※3 License of PlusPak for Windows Forms is required.
※4 Implementation through usage of column template mode and coding is required.
※5 For a cell that is being editing, only the following cell types can be displayed:
  - TextBoxCell
  - MaskedTextBoxCell
  - FilteringTextBoxCell
  - InputManCell

**What's New**

This section describes the features that are new in MultiRow 7.0.
- What's New in the Designer
- What's New in the Grid Control
- New Cell Types
What's New in the Designer

This section describes the new features of the MultiRow Designer.

Setting User-Defined Objects

You can call derived classes of user-defined shortcut keys and cell styles from the GUI. In 6.0, user-defined objects had to be set in code, but in 7.0, user-defined objects can easily be set using the following setting dialog.

- CellValidator Collection Editor (Cell.Validators property)
- CellValidateAction Collection Editor (CellValidator.Actions property)
- Shortcut Keys (GcMultiRow.ShortcutKeyManager property)
-CellStyle Editor (Cell.Style property)
- SummaryCell Editor (SummaryCell.Calculation property)

Smart Tags for Cells

Smart tags are supported in each cell that is placed in the template. You can use these smart tags to easily set the key properties of the cells.

Reminder Feature for GDI+ Settings

When you change the following properties to configure GDI+, you can display the GDI + Properties Confirming dialog box, and automatically perform the settings for which GDI+ is enabled.

The conditions for displaying the GDI + Properties Confirming dialog are as follows:

<table>
<thead>
<tr>
<th>Property</th>
<th>Show Dialog</th>
<th>Hide Dialog</th>
</tr>
</thead>
<tbody>
<tr>
<td>TextVertical</td>
<td>True</td>
<td>Inherit or False</td>
</tr>
<tr>
<td>TextAdjustment</td>
<td>Other than Inherit or Near</td>
<td>Inherit or Near</td>
</tr>
<tr>
<td>LineAdjustment</td>
<td>Distribute or DistributeWithSpace</td>
<td>Inherit or None</td>
</tr>
</tbody>
</table>
TextAngle

Other than NaN
Nan or 0

Enhancement of the Property Settings Dialog

You can visually set the position and font effects for the text to be placed in the cell, from the properties window.

- LineAdjustment property
- TextAdjustment property
- TextAlign property
- TextAngle property
- TextEffect property

Enhancement of the Table Feature

The following enhancements have been made in the Table.

- Setting caption for the table header
- Insertion of multiple rows or columns
- Enabling the Snap Line function for each cell in the table

Inline Editing of Cells

You can perform inline editing of the Value property, by double-clicking a cell on the template.

Properties that support inline editing are as follows:

<table>
<thead>
<tr>
<th>Cell</th>
<th>Target Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>CheckBoxCell</td>
<td>Text</td>
</tr>
<tr>
<td>RadioGroupCell</td>
<td>Items</td>
</tr>
<tr>
<td>ListBoxCell</td>
<td></td>
</tr>
<tr>
<td>ComboBoxCell</td>
<td></td>
</tr>
<tr>
<td>ListLabelCell</td>
<td></td>
</tr>
<tr>
<td>DomainUpDownCell</td>
<td></td>
</tr>
<tr>
<td>ImageCell</td>
<td>None</td>
</tr>
<tr>
<td>SummaryCell</td>
<td>Value</td>
</tr>
<tr>
<td>Other Cells</td>
<td></td>
</tr>
</tbody>
</table>

Editing Multi-line Text

You can enter text in multiple rows from the Properties window.
The properties that support multi-line text are as follows:

- TextBoxCell.DesignTimeValue property
- TextBoxCell.Style.NullValue property
- TextBoxCell.Value property
- RichTextBoxCell.DesignTimeValue property
- RichTextBoxCell.Style.NullValue property
- RichTextBoxCell.Value property
- CheckBoxCell.Text property
- FilteringTextBoxCell.DesignTimeValue property
- FilteringTextBoxCell.Style.NullValue property
- FilteringTextBoxCell.Value property
- ShapeCell.DesignTimeValue property
- ShapeCell.Style.NullValue property
- ShapeCell.Value property
- GcTextBoxCell.DesignTimeValue property
- GcTextBoxCell.Style.NullValue property
- GcTextBoxCell.Value property

**Conversion from DataGridView**

The layout set in a DataGridView control can be converted to a GcMultiRow template. You can freely change the layout of the Template after the conversion, in the designer.

For more information, refer to Conversion from DataGridView.

**Focus Flow**

The focus flow is displayed in accordance with the tab order of the cell. Since the focus flow can be changed by mouse drag, you can visually set the tab order.

For more information, refer to Changing Tab Order of Cells in the Designer.

**Changing the Index**

In the context menu that appears by right-clicking on the designer, you can automatically set the tab order and index of the cell.
Selecting the Data Type

You can select the data type in the Cell.Value property, in the Properties window.

The properties that support selection of data type are as follows:
- ButtonCell.DesignTimeValue property
- ButtonCell.Value property
- ColumnHeaderCell.DesignTimeValue property
- ColumnHeaderCell.Value property
- HeaderCell.DesignTimeValue property
- HeaderCell.Value property
- LabelCell.DesignTimeValue property
- LabelCell.Value property
- RowHeaderCell.DesignTimeValue property
- RowHeaderCell.Value property

Enhancement of the Template Wizard

You can create a basic column mode template, using the Template Wizard.
For details, refer to Create Template using Wizard.

Enhancement of the Toolbar

The following commands have been added to the toolbar:
- Designer Options
- Text Rotation
- Cell Type Icon
Annotations
You can use annotations at design time.
For more information, refer to Using Annotations.

Enhancement of Template PropertyList
You can freeze the columns of the Template PropertyList. You can also display CellIndex in the Template PropertyList.

Object Selector
In the context menu that appears by right-clicking on the designer, you can select an object which is in a non-selected state.

Copy of the Default Template
When the default template is set in the GcMultiRow control, and you select Edit Template from the smart tag of the control, you can specify whether you want to create a copy of the default template.

Cell Selection Range Mode
You can change the range selection mode by selecting the cell by dragging the mouse or ruler.
For more information, refer to **Selecting Cells in the Designer**.

**Zooming Cell Borders**

Using AllowBorderLineZoom of **Designer Options**, you can specify whether you want to zoom the border lines, when zooming the template in the designer.

**Named Cell Style Manager**

If the named cell style is used in any of the following styles, the usage status is now displayed in the named cell style manager.

- Section.DefaultCellStyle
- Template.AlternatingRowsDefaultCellStyle
- Template.ColumnFootersDefaultCellStyle
- Template.ColumnFootersDefaultHeaderCellStyle
- Template.ColumnHeadersDefaultCellStyle
- Template.ColumnHeadersDefaultHeaderCellStyle
- Template.DefaultCellStyle
- Template.RowsDefaultCellStyle
- Template.RowsDefaultHeaderCellStyle

**Recognizing the Cell Name Without Distinction Between Upper or Lower Case**

When the Template.AllowCellNameComparisonIgnoreCase (AllowCellNameComparisonIgnoreCase Property in the on-line documentation) property is set to True, the cell name (Cell.Name (Name Property in the on-line documentation) property) can be recognized without distinction between upper or lower case.

**Using Code**

This example ignores the case in the cell name.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow
Dim textBoxCell1 As New TextBoxCell() 
textBoxCell1.Name = "textBoxCell1"
Dim template1 As Template = Template.CreateGridTemplate(New Cell() {textBoxCell1})
template1.AllowCellNameComparisonIgnoreCase = True
GcMultiRow1.Template = template1
GcMultiRow1.RowCount = 3
GcMultiRow1.Rows(0).Cells("textBoxCell1").Value = "ABC"
GcMultiRow1.Rows(0).Cells("TextBoxCell1").Style.BackColor = Color.Yellow
GcMultiRow1.Rows(0).Cells("textboxcell1").Style.ForeColor = Color.Red
```

**[CS]**

```cs
using GrapeCity.Win.MultiRow;
TextBoxCell textBoxCell1 = new TextBoxCell();
textBoxCell1.Name = "textBoxCell1";
Template template1 = Template.CreateGridTemplate(new Cell[] { textBoxCell1 });
template1.AllowCellNameComparisonIgnoreCase = true;
gcMultiRow1.Template = template1;
gcMultiRow1.RowCount = 3;
gcMultiRow1.Rows[0].Cells["textBoxCell1"].Value = "ABC";
gcMultiRow1.Rows[0].Cells["TextBoxCell1"].Style.BackColor = Color.Yellow;
```
What's New in the Grid Control

This section describes the new features of the Grid control in MultiRow. Refer to New Cell Types, for the new cell types added in 7.0.

**Automatic Merging of Cells**

You can specify to merge cells automatically if the same value is set in vertically adjacent rows.

For details, refer to Automatic Merging of Cells.

<table>
<thead>
<tr>
<th>Company</th>
<th>Department</th>
<th>Employeaid</th>
<th>User Name</th>
<th>Gender</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>New York Bra</td>
<td>1</td>
<td>101</td>
<td>Ferdinand</td>
<td>F</td>
<td>NewYork</td>
</tr>
<tr>
<td>Georgia Br</td>
<td>2</td>
<td>102</td>
<td>Carr</td>
<td></td>
<td>Georgia</td>
</tr>
<tr>
<td>Maryland Br</td>
<td>3</td>
<td>103</td>
<td>Booth</td>
<td>M</td>
<td>Maryland</td>
</tr>
<tr>
<td>Alabama Br</td>
<td>4</td>
<td>104</td>
<td>Blakc Jackson</td>
<td></td>
<td>Alabama</td>
</tr>
<tr>
<td>New York Br</td>
<td>5</td>
<td>105</td>
<td>Montague</td>
<td></td>
<td>NewYork</td>
</tr>
<tr>
<td>Illinois Br</td>
<td>6</td>
<td>106</td>
<td>James Hall</td>
<td>F</td>
<td>Illinois</td>
</tr>
<tr>
<td>Ohio Branch</td>
<td>7</td>
<td>107</td>
<td>Hubert White</td>
<td></td>
<td>Ohio</td>
</tr>
<tr>
<td>Arkansas Br</td>
<td>8</td>
<td>108</td>
<td>Evan Hand</td>
<td></td>
<td>Arkansas</td>
</tr>
<tr>
<td>Colorado Br</td>
<td>9</td>
<td>109</td>
<td>Las Hard</td>
<td>M</td>
<td>Colorado</td>
</tr>
<tr>
<td>Alaska Br</td>
<td>10</td>
<td>110</td>
<td>Elton Green</td>
<td></td>
<td>Alaska</td>
</tr>
<tr>
<td>Pennsylvania</td>
<td>11</td>
<td>111</td>
<td>Upton Fox</td>
<td></td>
<td>Pennsylvania</td>
</tr>
<tr>
<td>Hawaii Br</td>
<td>12</td>
<td>112</td>
<td>Don Wood</td>
<td></td>
<td>Hawaii</td>
</tr>
<tr>
<td>Indiana Br</td>
<td>13</td>
<td>113</td>
<td>Murray Wall</td>
<td>F</td>
<td>Indiana</td>
</tr>
<tr>
<td>FLORIDA Br</td>
<td>14</td>
<td>114</td>
<td>Kelly Turner</td>
<td></td>
<td>FLORIDA</td>
</tr>
<tr>
<td>Missouri Br</td>
<td>15</td>
<td>115</td>
<td>Cedric Stock</td>
<td></td>
<td>Missouri</td>
</tr>
<tr>
<td>California Br</td>
<td>16</td>
<td>116</td>
<td>Geoffrey Sm</td>
<td>F</td>
<td>California</td>
</tr>
</tbody>
</table>

**Expanding or Collapsing Columns**

Methods for expanding, collapsing columns, and properties for getting the open and close status of the columns have been added. You can expand or collapse columns by combining these functions.

For more information, refer to Expanding Or Collapsing Columns.

**Column Mode Template**

Rather than setting the conventional expanding direction of the row, which is top to bottom, you can also set it to the horizontal direction, which is left to right. Rows can be displayed as columns, by repeating rows in the horizontal direction.

When using the column mode template, set the Template. `LayoutMode ('LayoutMode Property' in the on-line documentation) property to LeftToRight.

For details, refer to Column Mode Template.
Cell Notes

MultiRow provides cell notes for displaying additional information on the grid. The end-user can add, delete, and edit cell notes.

For details, refer to Cell Note.

Enhancement of Print Features

In the print features, the following features have been enhanced:

- Specifying the Page Break Position
- Specifying Empty Rows for Print
- Printing Watermarks
- Print Settings for each Column Header Section

Enhancement of Style Settings

Borders of the Current Row

You can set the border style for the current row. You can highlight the current row by setting the color and style of its borders.

For details, refer to Current Row.

Default Header for Alternating Rows

The default cell style used in the header cell of alternating rows can be set by the GcMultiRow.AlternatingRowsDefaultHeaderCellStyle ('AlternatingRowsDefaultHeaderCellStyle Property' in the on-line documentation) property.

Mouse Hover

You can set the style when the mouse is over sections and cells.

For details, refer to Mouse Hover Styles.

Cell being Edited

You can specify the background and foreground colors to be applied when editing a cell. You can use the CellStyle.EditingBackColor ('EditingBackColor Property' in the on-line documentation) and CellStyle.EditingForeColor ('EditingForeColor Property' in the on-line documentation) properties to specify the background and foreground colors during editing.

For details, refer to BackColor and ForeColor.
Displaying a New Row at the Top

You can specify whether to display a new row at the top, or at the bottom of the grid. If you display it at the top, you can customize the color and width of the separator that separates the new row from the other rows.

For more information, refer to Displaying a New Row at the Top.

Automatic Adjustment of the Cell Width

You can automatically adjust the width of the cells placed in the column header section, column footer section, or row section, to fit the cell contents.

In addition, you can implement the following operations when you double-click on the right edge of the header.

- Automatically adjust the column width of multiple columns simultaneously
- Disable the automatic adjustment of column width
- Customize the column width

For more information, refer to Automatic Adjustment of the Cell Width.

Enhancement of Validation Features

You can use the Cell.Validate ('Validate Method' in the on-line documentation) method to perform validation for all cells or for the value being edited.

The following built-in cell validators and built-in validation actions have also been added.

- Built-in Cell Validators
  - String Comparison
  - String Length
  - Encoding Format
- Built-in Validation Actions
  - Value Processing
  - Notification by Message Box
  - Notification by 3 Stage Icon

Data Binding in the Column Header Section

Data binding is supported for the cells placed in the column footer section and column header section.

Retrieving the Method Used to Move Cells

The MoveStatus property for retrieving the method used to move cells has been added to the following events:

- GcMultiRow.CellEnter event
- GcMultiRow.CellLeave event
- GcMultiRow.CellBeginEdit event
- GcMultiRow.CellEndEdit event
- GcMultiRow.RowEnter event
- GcMultiRow.RowLeave event
- GcMultiRow.NewCellPositionNeeded event

You can use the MoveStatus property to retrieve the status that indicates the operation that moved the cell.

For details, refer to Retrieving the Method Used to Move Cells.

Selecting All Cells

You can use the GcMultiRow.SelectAll ('SelectAll Method' in the on-line documentation) method to select all the cells that can be selected.
**Adding Shortcut Key Actions**

The following properties for moving the cell on the basis of the tab order have been added to the SelectionActions class.

- `SelectionActions.MoveToFirstCellByTabOrder ('MoveToFirstCellByTabOrder Property' in the on-line documentation) property`
- `SelectionActions.MoveToFirstCellInRowByTabOrder ('MoveToFirstCellInRowByTabOrder Property' in the on-line documentation) property`
- `SelectionActions.MoveToLastCellByTabOrder ('MoveToLastCellByTabOrder Property' in the on-line documentation) property`
- `SelectionActions.MoveToLastCellInRowByTabOrder ('MoveToLastCellInRowByTabOrder Property' in the on-line documentation) property`

The following properties for canceling the editing operation have been added to the EditingActions class.

- `EditingActions.CancelCellEdit ('CancelCellEdit Property' in the on-line documentation) property`
- `EditingActions.CancelRowEdit ('CancelRowEdit Property' in the on-line documentation) property`

**Selection Color in case of Non-selectable or Invalid Cells**

You can draw the selection color for cells that are set as disabled or non-selectable when selecting a range of cells, as in row selection.

**Setting Coordinates of the Scroll Position**

You can use the GcMultiRow.FirstDisplayedLocation ('FirstDisplayedLocation Property' in the on-line documentation) property to specify the coordinates to be displayed at the upper left corner of the grid.

**Copying the Caption of the Header to the Clipboard**

You can specify whether to copy the caption of the header to the clipboard using the GcMultiRow.ClipboardCopyMode ('ClipboardCopyMode Property' in the on-line documentation) property.

**Background Gradation Dialog**

A new dialog for setting the background gradation has been added.
**User-defined Word Wrap**

You can specify the line break position by number of characters, independent of the cell width. You can also make sure that the line break does not occur in the middle of a double-byte number, and it is recognized as one word.

For details, refer to User-defined Word Wrap.

**User-defined Auto-fit**

You can display single byte values uniformly in the cell, without space-separated justification.

For details, refer to Customizing the Justification of Single-byte Characters.

**Zooming Borders**

You can specify whether to zoom the cell borders. Set the GcMultiRow.AllowBorderStyleZoom ('AllowBorderStyleZoom Property' in the on-line documentation) property to True to zoom the cell borders.

**Displaying the Size Grip**

You can specify whether to display the size grip at the bottom right of the grid. Set the GcMultiRow.ShowSizeGrip ('ShowSizeGrip Property' in the on-line documentation) property to True to display the size grip.

**Enhancement of Automatic Generation of Templates**

Based on the size of an array or matrix of the cell type, you can automatically set the caption of the column header in the Template.CreateGridTemplate ('CreateGridTemplate Method' in the on-line documentation) method that automatically generates a template of the basic layout. In addition, the Template.CreateGridTemplateForColumnMode ('CreateGridTemplateForColumnMode Method' in the on-line documentation) method, which supports the column mode template, has been added.

**Display Time of Tooltips**

You use the following properties to set the display time of the tooltip which is displayed when the mouse pointer pauses over a cell.

- GcMultiRow.CellToolTipAutomaticDelay ('CellToolTipAutomaticDelay Property' in the on-line documentation) property
- GcMultiRow.CellToolTipAutoPopDelay ('CellToolTipAutoPopDelay Property' in the on-line documentation) property
- GcMultiRow.CellToolTipInitialDelay ('CellToolTipInitialDelay Property' in the on-line documentation) property

**Retrieving the Position of the Splitter Line**

You can use the GcMultiRow.GetHorizontalSplitLocations and GcMultiRow.GetVerticalSplitLocations methods to get the location of the splitter line.

For details, refer to Splitter Line and ViewPort.

**Selecting the Section by clicking on the Blank Area**

You can specify whether to select a section when you click on a blank area in that section. Set the GcMultiRow.SelectRowWhenClickBlankArea ('SelectRowWhenClickBlankArea Property' in the on-line documentation) property to True to select a section when you click on a blank area in that section.

**Automatic Restoration of Values of Cells in the Column Header Section**

When you change the template at run-time, the values of cells placed in the column footer section and column header section before the change can be replaced by the cells after the change. Set the RestoreColumnHeaderFooterValue ('RestoreColumnHeaderFooterValue Property' in the on-line documentation) property to True to enable
New Cell Types

This section describes the new features of MultiRow Cell Types.

New Cell Types

- **List Box Cell**
  A cell type in which you can select an item to be entered into the cell from a defined list of items, in the same way as in the standard list box control. For more information, refer to `ListBoxCell`.

- **List Label Cell**
  A cell type that can display labels with a list of items in a bulleted format. You can display bullets simply by adding items. For more information, refer to `ListLabelCell`.

- **Shape Cell**
  A cell type in which you can draw shapes such as circles, triangles, and squares in the cell at design time, without coding. The set shape is displayed at run time. For details, refer to `ShapeCell`.

Enhancement of Existing Cells

- **Column Header Cell**
  In this cell type, you can specify an image for the indicators that appear when you perform filtering and sorting. When filtering, you can clear all filters at once, exclude fixed rows, and select multiple filters. For details, refer to `Filtering Rows using the Column Headers` and `Sorting Rows using Column Headers`.

- **Button Cell**
  The `GcMultiRow.CellContentButtonClick` ('CellContentButtonClick Event' in the on-line documentation) event, that occurs when the user clicks the button cell using the keyboard or mouse, has been added. In addition, you can execute a built-in action when the button cell is clicked. For more information, refer to `Using Button Commands (ButtonCell)`.

- **Header Cell**
  By setting the background color of the header cell to transparent, you can display the gradation set in the background of the section as the background color of the header cell. This feature can also be used in the `ColumnHeaderCell`, `RowHeaderCell`, and `CornerHeaderCell` that inherit the `HeaderCell`.

  You need to set the following properties to display the background gradation of the section, in the background of the header cell.
  
  - Set the `HeaderCell.FlatStyle` ('FlatStyle Property' in the on-line documentation) property to Flat or Popup.
  - Set the `CellStyle.BackColor` ('BackColor Property' in the on-line documentation) property to Transparent.
  - Set the `HeaderCell.UseRealTransparent` ('UseRealTransparent Property' in the on-line documentation) property to True.

Summary Cell

A new dialog for defining operators has been added. In addition, integer division(\(\)), power(\(^\)), and remainder(\(\%\)), have been added as new operators that can be used.

**Image Cell**

In this cell type, you can display animations of GIF format. Set the `ImageCell.SupportsAnimation` property to `True` to enable GIF format animations.

**Row Header Cell**

You can set the format in the `ValueFormat` property using the `ValueFormat` editor.

**Progress Bar Cell**

You can specify whether to use Visual styles as the progress bar style. Set the `ProgressBarCell.UseVisualProgressBarStyle` property to `False`, if you do not want to use Visual styles.

**Cell.Tag property**

The `Cell.Tag` property that can set or get additional data related to the Cell has been added.

**Prefix Characters of Access Keys**

Ampersand characters (&) in the Value property of label cells and button cells can be displayed as prefix characters of access keys. Set the `UseMnemonic` property to `True`, if you want to display the ampersand character as an access key prefix character.

**Using Code**

This example displays a prefix character.

```vbnet
Imports GrapeCity.Win.MultiRow
Private Sub Form1_Load(sender As System.Object, e As System.EventArgs) Handles MyBase.Load
    Dim buttonCell1 As New ButtonCell()
    buttonCell1.Name = "buttonCell1"
    buttonCell1.Value = "Delete(&D)"
End Sub
```
Breaking Changes

This section describes the changes in 7.0.

- **Compatibility with the Previous Version**
- **Breaking Changes from 6.0**

Compatibility with the Previous Version

The MultiRow development team has developed MultiRow for Windows Forms 7.0, taking its compatibility with the previous version (MultiRow for Windows Forms 6.0) into consideration. For example, the default value for each property is the same as in the previous version. Some changes need to be made by the developer to use features added for 7.0.

Compatibility

MultiRow 7.0 provides forward compatibility with the previous version. By rewriting the references to assemblies and license information, and building again, you can use the new features of 7.0 in applications created in the previous version, while maintaining the same behavior as in the previous version. There is a possibility that some behavior varies with the migration to 7.0, but for such cases, an option to maintain the same behavior as that in the previous version is available. The features that are likely to cause unexpected behavior due to source code of the previous version are disabled by default in 7.0. For example, this applies to the feature for input in the header, which was added in 6.0.

You cannot load the source code that is created in 7.0 in the previous version (backward compatibility is not supported).

Breaking Changes from 6.0

This section describes the changes from the previous version (MultiRow for Windows Forms 6.0).

Operating Environments

For 7.0, operation in the following environments is not guaranteed:

- Development tool: Visual Studio 2005
- Framework: .NET Framework 2.0 SP1/3.0 SP1/3.5 (*)

* In MultiRow 7.0, version 3.5 SP1 or later of the .NET Framework is required.

If you need to use the above environment, consider using 6.0.

MultiRow 7.0 works in the following environments.
- Development tool: Visual Studio 2012/ 2013
- Framework: .NET Framework 4.5/ 4.5.1

The MultiRow for Windows Forms 6.0 license is not included in this product.

MultiRow 7.0 SP2 and later works in the following environment.
- Windows 8^Server 2012 R2
- Visual Studio 2013
- .NET Framework 4.5.1

**Automatic Resizing**

If you set the GcMultiRow.AllowUserToAutoFitColumns ("AllowUserToAutoFitColumns Property" in the on-line documentation) property to True, you can change the size of the cell to the optimal width by double-clicking on the bottom or right edge of the header cell.

If you want to allow resizing of cells, but restrict the operation of automatic adjustment of the size of the cell by double-clicking the header cell, you can set the GcMultiRow.AllowUserToAutoFitColumns property to False.

**ScrollBarMode Property**

In 6.0, you could set the display method for both the horizontal and vertical scroll bars using the ScrollBarMode property, but in 7.0, you can use the HorizontalScrollBarMode ("HorizontalScrollBarMode Property" in the on-line documentation) property and the VerticalScrollBarMode ("VerticalScrollBarMode Property" in the on-line documentation) property, to set the display method for the horizontal scroll bar and the vertical scroll bar.

Though you can continue to use the existing ScrollBarMode property without any change in the operation, it is recommended that you use the HorizontalScrollBarMode and VerticalScrollBarMode properties.

**Gradation Settings**

In 6.0, you could set the background gradation using the CellStyle.GradientColors property, but in 7.0 you can set it using the CellStyle.BackgroundGradientEffect ("BackgroundGradientEffect Property" in the on-line documentation) property.

**Using Code**

The following code sets the background gradation.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

Dim textBoxCell1 As New TextBoxCell()
    GcMultiRow1.Template = Template.CreateGridTemplate(New Cell() {textBoxCell1})
    GcMultiRow1.RowCount = 10
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;

    TextBoxCell textBoxCell1 = new TextBoxCell();
    textBoxCell1.Style.BackgroundGradientEffect = new GradientEffect(new Color[] { Color.White, Color.Pink }, GradientStyle.Horizontal, GradientDirection.Forward);
    gcMultiRow1.Template = Template.CreateGridTemplate(new Cell[] { textBoxCell1 });
    gcMultiRow1.RowCount = 10;
```

**MoveStatus Property**

In 7.0 the MoveStatus property for retrieving the method used to move cells has been added. The MoveStatus property can be used in the following events.

- GcMultiRow.CellEnter event
- GcMultiRow.CellLeave event
- GcMultiRow.CellBeginEdit event
- GcMultiRow.CellEndEdit event
Co-existence and Migration from the Previous Version

This section explains the method for migration from and co-existence with the previous version.

- Coexistence with the Previous Version
- Migration from the Previous Version

Coexistence with the Previous Version

MultiRow for Windows Forms 7.0 (referred to as 7.0) is the successor product of MultiRow for Windows Forms 6.0.
The architecture of MultiRow is different in 6.0 and 7.0.

**Coexistence in the Same Environment**

You can install both 7.0 and 6.0 in the same development and runtime environment, and use them simultaneously.

**Using in Combination in the Same Project**

MultiRow 7.0 cannot be used in the same project as MultiRow for Windows Forms 6.0. Only one of them can be used.

**Migration from the Previous Version**

MultiRow for Windows Forms 7.0 (also referred to as 7.0) maintains compatibility with the previous version. This section describes the parts that need to be rewritten when migrating from 6.0.

### Migration of MultiRow

- **Rewriting the Project File**
  
  You need to rewrite the following part of the project file, if the project was created in 6.0.

  ```xml
  <Reference Include="GrapeCity.Framework.MultiRow.v21, Version=2.1.yyyy.mmdd, Culture=neutral, PublicKeyToken=fdd90f5c44b73170" /> <Reference Include="GrapeCity.Win.MultiRow.v60, Version=6.0.yyyy.mmdd, Culture=neutral, PublicKeyToken=0f7a722ee3c2bdd9" />
  ``

  You need to rewrite the project file in the following way to migrate to 7.0.

  ```xml
  <Reference Include="GrapeCity.Framework.MultiRow.v22, Version=2.2.yyyy.mmdd, Culture=neutral, PublicKeyToken=fdd90f5c44b73170" /> <Reference Include="GrapeCity.Win.MultiRow.v70, Version=7.0.yyyy.mmdd, Culture=neutral, PublicKeyToken=0f7a722ee3c2bdd9" />
  ```

  yyyy.mmdd will vary depending on the product assembly version. For assembly versions, refer to the ReadMe installed with the product.

- **Rewriting the License File**
  
  You need to rewrite the following part of the license file in the project created in 6.0.

  ```text
  ```
You need to rewrite the license file in the following way to migrate to 7.0:

GrapeCity.Win.MultiRow.GcMultiRow, GrapeCity.Win.MultiRow.v70, Version=7.0.yyyy.mmdd, Culture=neutral, PublicKeyToken=0f7a722ee3c2bdd9

yyy.mddd will vary depending on the product assembly version. For assembly versions, refer to the ReadMe installed with the product.
Using MultiRow

The following sections explain how to use the MultiRow control.

- MultiRow Architecture
- Quick Start
- Designer Window
- Using the Designer
- Advanced Designer Features
- Template
- Grid
- Headers
- Rows and Cells
- Column Mode
- Managing Data
- Cell Types
- Cell Styles
- Conditional Cell Styles
- User Input Validation
- Connecting to Database
- Print and Print Preview
- File Input and Output
- Multi-touch Features
- Troubleshooting

MultiRow Architecture

This section explains the structure and features of the MultiRow control. Please refer to each page link for details and sample code on how to use these features. To start using MultiRow immediately, you can also refer to the Quick Start section.

- Basic Concepts
- Template Structure
- Template File
- Cell Structure
- Format String

Basic Concepts

MultiRow is a grid control where one record does not need to be confined to one row but can be displayed easily in multiple rows. It can be used with the standard DataGridView provided by the .NET Framework or as a substitute for it. This provides you the freedom to define the grid headings and row layout, and also various borders and character styles. You can also replicate the display of a paper form on your input screen.

MultiRow Operation Workflow

MultiRow is a grid control so it can be added to the form just like a standard control. You can assign a Template, that has the headers and row layout, to the control and implement it to get a tabular screen in your application. The operation workflow is shown below.
Refer to **Quick Start** for operational details.

**Template and Grid**

This grid control uses the assigned template and Datasource to create a tabular screen at runtime. The row count of the grid is used when the grid is not bound to a Datasource.

Basic elements of grid controls like the header, footer, and cells can be added to the template, as per the requirement. Cells placed in the rows are displayed recursively based on the settings of the grid.

All the elements in the template have an index which can be used for various operations at runtime. For example, a cell with a 0 index is referred to as the 0th cell of the nth row at runtime.
In order to make the identification easier, you can also assign keys to cells as you would to fields in a database. Each key name should be unique to the template.

The template can be modified or replaced at runtime.

Grid behavior for cell level operations is set in the template, but settings for all other operations which effect the entire grid, are done at the grid control level. Examples of grid level operations are assigning a data source, automatically adding a row, allowing resizing, virtual mode, row count, zooming, edit mode, and so on. In case any conflict arises between the control and the template, the settings done at the control level are given priority. For example, even when
Allow Header Cell Resize is set in the template, resizing will not be allowed at runtime unless the resizing operation is allowed at the control level.

**Development Cycle using MultiRow**

The development of a tabular screen using MultiRow, involves the repetition of basic steps of designing a template, assigning it to the grid control, and then running the grid control.

The Multirow designer has a feature that allows previewing the running Grid control. The following flow takes place when using a designer.

The template can be assigned to the control at any time.

**Template Structure**

A template consists of a number of regions. Each region refers to the corresponding section on the grid.
ColumnHeaderSection

Column Header Section is the region that is fixed at the top of the grid at runtime. It is used for creating column headers and filter rows. Multiple column headers can be added to the grid and they can be displayed or hidden at runtime. For example, you can create separate column headers for display, printing, and toggling between them. A maximum of 3000 column header sections can be added to one template. The developer can make the cell value of the column header section editable for users, but this feature has been disabled by default for the purpose of maintaining compatibility.

Row

Row is the region which is rendered recursively at runtime depending on the number of records in a grid. When the grid is connected to a database with 300 records, row is rendered 300 times. The developer can place cells in rows as required but the control still operates as a simple one row one record grid even when a complex tabular structure is created. The row layout cannot be the same as the column header section. One template can only have one type of row display.

ColumnFooterSection

Column Footer Section is the region that is fixed at the bottom of the grid at runtime. Any cell can be placed in this section to display grid information at the bottom of the screen (similar to the column header section). Multiple column footer sections can be added to the grid. A maximum of 3000 column footer sections can be added to one template. A developer can make the cell value of the column footer section editable for users, but this feature has been disabled by default for maintaining compatibility.

Section

The column header section and column or row footer sections are all inherited from the common Section ('Section Class' in the on-line documentation) class; therefore, many of their properties are also similar. You can use the Section ('Section Class' in the on-line documentation) class for common implementation in these three sections.
Template File

A template created with the MultiRow designer is added to the project as a Visual Basic or C# source file. This is the same as Form1.vb or Form1.cs getting added to the project when a form is created using a form designer in Visual Studio.

File Configuration

A template file is composed of three files with the same prefix (different extensions), just as a form. For example, when a Visual Basic template is created with a name "Template1", the following files are added to the project.

- Template1.vb
- Template1.Designer.vb
- Template1.resx

The extension becomes .cs in C#. If only one file of the above is shown in the Visual Studio solution explorer, select
**Project - Show All Files** in Visual Studio. The class definition is created in Template1.vb and usually, there is no need to change this file. Template1.Designer.vb is the partial class of Template1.vb. This file stores the information about the sections placed on the template such as cell definitions, layout information, and so on as the source code. Template1.resx is the resource file of the template. This file stores design information such as tables, binaries of images, internationalization resources, locked status of cells, and so on in XML format. Manage these three files as one set, as in the case of forms.

**Template File and Coding**

If the template for MultiRow is created through code and not the designer, creating the template file is optional. Since template is an inherited class of .NET Framework, there are no limits to the implementations that can be done using this class, if the designer is not used. On the other hand, be cautious when the template file created with the designer, is edited manually. If the code or implementation is not supported by the designer, there might be cases where the designer crashes, the edited code is lost, or there might be some other unexpected result.

**Template File and Source Code Management Tool**

Since the template file is a Visual Basic or C# file, common source code management tools can be used for sharing or managing versions or differences. Note that the default encoding of the source code file is Unicode (UTF-8).

**Cell Structure**

The cell is a rectangular object that can be placed onto each section. The cells in MultiRow can be selected or edited by the user in the same way as Excel or DataGridView. The difference compared to other grid controls is that the developer can place the cells of desired size at the desired location.

The cells that have been placed on the designer at design time get replicated at run time by the data using the Clone method. The cells placed in the header and footer are displayed as is.
Cell Selection

There are settings to specify whether a cell can be selected, whether cells are in the selected state, and appearance settings to distinguish the selected cells from the cells not currently selected. For details, refer to Selection Mode, Selected Rows and Cells, or Restricting Cell Selection.

Cell Index

Indices are allocated to cells in the order in which they are added to the sections (Cell.Index). At runtime, you can specify a particular cell placed in the row header or column header section using this index. The cell index can be changed using the Cell.Index property or the designer. Cell indices are suitable for batch processing of a specified range of cells or processing of unspecified multiple cells. Use the cell name for specifying a particular cell as shown below.

Cell Name

You can access a specific cell from the Cells collection using the cell name (Cell.Name ('Name Property' in the online documentation)) as the key.

Using Code

As shown in the code below, this is similar to naming the controls on a form and accessing those controls using the Form.Controls property and control name.

[VB]

```vbnet
Dim textBox1 As New TextBox()  
textBox1.Name = "textBox1"  
Me.Controls.Add(textBox1)  
Me.Controls("textBox1").Text = "Hello"
```

[CS]

```csharp
TextBox textBox1 = new TextBox();  
textBox1.Name = "textBox1";  
this.Controls.Add(textBox1);  
this.Controls["textBox1"].Text = "Hello";
```

A particular cell can be accessed as shown below.

[VB]
Imports GrapeCity.Win.MultiRow

' Create a template
Dim textBoxCell1 As New TextBoxCell()
textBoxCell1.Name = "textBoxCell1"

GcMultiRow1.Template = Template.CreateGridTemplate(New Cell() {textBoxCell1})
GcMultiRow1.RowCount = 3

' Run time operation
GcMultiRow1.Rows(0).Cells("textBoxCell1").Value = "Hello"
GcMultiRow1.SetValue(1, "textBoxCell1", "Hello")

[CS]
using GrapeCity.Win.MultiRow;

// Create a template
TextBoxCell textBoxCell1 = new TextBoxCell();
textBoxCell1.Name = "textBoxCell1";

gcMultiRow1.Template = Template.CreateGridTemplate(new Cell[]{textBoxCell1});
gcMultiRow1.RowCount = 3;

// Run time operation
gcMultiRow1.Rows[0].Cells["textBoxCell1"].Value = "Hello";
gcMultiRow1.SetValue(1, "textBoxCell1", "Hello");

Position and Size

The cell position can be specified relative to the top left corner of the parent section. This is the same as placing a control on the form. The cell size can be specified using the cell height and width. Since the cell is a rectangle, this is enough information to set the position of the cell.

Resizing Cells

The resizing of cells is determined by cell positioning by default. For example, once you have placed the row cells matching the width of the header, if you resize the header at run time, it will automatically be followed by a change in the width of the cell. In the designer, the positioning can be easily specified using built-in snap lines.

Margin and Padding

You can specify the margin and padding as part of the style similar to other controls placed on the form. The margin specifies the blank space around the cell area. Since the margin does not affect the cell contents, they are enabled only
when you are using snap lines at design time. The padding defines the blank space around the inside of the cell edges. The following image shows a margin and padding of 10 pixels on all four sides of buttonCell1. Blue lines show margins and red lines show padding. The margin and padding of buttonCell2 and buttonCell3 were not changed for comparison purposes.

Inheritance

Cells can be inherited to create custom cell types. You can either create new cells from scratch using the basic cells or you can extend the features by inheriting the built-in cells. The cell types thus created can be shared or reused between projects. For example, `FilteringTextBoxCell` ("FilteringTextBoxCell Class" in the on-line documentation) is derived from `TextBoxCell` ("TextBoxCell Class" in the on-line documentation) to provide filtering functionality, and `ColumnHeaderCell` ("ColumnHeaderCell Class" in the on-line documentation) (specifically designed for the top-left header) has been derived from `HeaderCell` ("HeaderCell Class" in the on-line documentation).

Common Settings for Cells

All the cell types have been derived from the common Cell class, just as all the controls have been derived from System.Windows.Forms.Control class. Similar to the controls, the Cell class provides basic features of cells, such as location and size.

Cell Style

The cell styling is not used for individual cells. For example, the settings for background color, image background, and
image settings are not only used in a particular cell but are common for a number of cells. These kinds of settings are provided in a common object as the cell style. This structure is similar to the DataGridViewCellStyle which provides style settings for cells in the DataGridView control. The style settings that are applied to specific cells might differ depending on the cells, but you can specify the main style settings for a number of cells.

**Header Cells**

Header cells are special cells that help execute some important functionalities provided by the grid. For example, you can select cells as well as sort them using the headers. Header cells are basically of 3 types, and correspond to the 3 header locations respectively.

![Header Types](image)

All the header cells are derived from the header cell type and they have common settings.

- Header Type Cells
- Column Header Type Cells
- Row Header Type Cells
- Corner Header Types Cells

**Editing**

MultiRow implements standard controls from the .NET Framework to provide the editing functionality. For example, System.Windows.Forms.TextBox control is displayed when you try to edit the TextBoxCell. As a result, if you would like to customize the editing features, you can use the same events that are used during the editing of the standard controls.

![Editing Control](image)

The cell editing feature in MultiRow is defined using the common interface IEditingControl (`IEditingControl Interface in the on-line documentation`). The following items are defined in this interface: mouse cursor displayed in edit mode, values to be edited, reference to parent GcMultiRow control, current Row number, zoom value, and editing controls.

The following table shows the cell type and the corresponding editing control. Grid events can be used to customize the editing operation for cell types that do not have corresponding editing controls.

<table>
<thead>
<tr>
<th>Cell Type</th>
<th>Cell Editing</th>
<th>Control Inherited From</th>
</tr>
</thead>
<tbody>
<tr>
<td>ComboBoxCell</td>
<td>ComboBoxEditingControl</td>
<td>System.Windows.Forms.ComboBox</td>
</tr>
</tbody>
</table>
MultiRow Windows Forms Developer's Guide

Rendering Mode

MultiRow cells support two types of rendering modes, GDI and GDI+, which are essentially same as for the standard .NET Framework controls. GDI is the default rendering mode which provides the same rendering as that used for the editing control. GDI+ is used for Vertical Display or Distributed Display. For details, refer to GDI+ Compatibility Mode.

Format String

One of the key features of MultiRow is its support of format strings compatible with the .NET Framework.

String Format

The following features use the same format strings as the .NET Framework.

- ProgressBarCell.TextFormat ('TextFormat Property' in the on-line documentation) Property
- PrintInfoCell.FormatString ('FormatString Property' in the on-line documentation) Property
- CellStyle.Format ('Format Property' in the on-line documentation) Property

For details about the .NET Framework format settings, refer to the .NET Framework Developer's Guide Formatting Overview. The following feature uses independent formats.

- RowHeaderCell.ValueFormat ('ValueFormat Property' in the on-line documentation) Property

If you specify "%1%" with this property, it shows continuous numbering, 1, 2, 3, 4, based on the number of rows. Use "%A%" for an alphabetic display. For more details, refer to RowHeaderCell.ValueFormat property.

String Format and Data Types

The .NET Framework provides formats for numeric values, dates, time, and enumerations. To implement these formats, the corresponding data type for the cell needs to be of the same type. For example, even when you specify numeric format for a text type cell, the value entered by the user would still be text type, so formatting would not be applied to this entry. In order to specify numeric format, the data type of the cell also needs to be numeric.

Culture

The culture that can be set with the following properties is compatible with the .NET Framework.

- MaskedTextBoxCell.Culture ('Culture Property' in the on-line documentation) Property
- CellStyle.FormatProvider ('FormatProvider Property' in the on-line documentation) Property

The above members provide essentially the same functionality as the following .NET Framework features.

- System.Windows.Forms.MaskedTextBox.Culture Property

Quick Start
This section shows how to use MultiRow in a simple application.

- **Adding MultiRow Component and Template to the Project**
- **Adding the MultiRow Component to a Visual Studio 2013 Project**
- **Creating an Application using GcMultiRow**
- **Connecting to SQL Server Database**
- **Creating an Application Using the Designer (Grid View)**
- **Creating an Application Using the Designer (Single Template View)**

### Adding MultiRow Component and Template to the Project

Follow the steps below to add the GCMultiRow control to the toolbox and use it in a project. These steps are for Visual Studio 2012, but are applicable to Visual Studio 2008, Visual Studio 2010, and Visual Studio 2013.

#### Adding the Component to the Toolbox

Follow the steps below to register the component in the toolbox.

1. Right click on the toolbox window and select **Add Tab** from the context menu.

2. Enter **MultiRow for WindowsForms 8.0** (or any other desired tab name) as the new tab name.
3. Right-click the new tab and select **Choose Items** from the context menu.
4. Select .NET Framework components in the Choose Toolbox Items dialog box and check the following component under the assembly name GrapeCity.Win.MultiRow(8.0.yyyy.mmdd).
   - GcMultiRow

   - If the runtime of a previous version still exists, then there may be more than one item displayed with the same component name. In this case, select the item with the latest assembly version (the larger number) from the "version" column ("assembly name" column for Visual Studio 2008 or 2010).

5. Click OK.
6. The component is added to the toolbox.
Adding a Component to the Form

Follow the steps below to add the control to the form.

1. Drag and drop the control or component from the toolbox to the form.
2. *Licenses.lxiv* is updated.
3. A reference to the MultiRow assembly is added to the project. Also, since InputManCell and PlusPakCell can be used with this product, a reference to the PlusPak and InputMan assemblies are also added to the project.

Adding a Template to the project

This process is similar to adding a new form in a project.

1. Create a new Windows Application project in Visual Studio or open an existing one.
2. Select **Project** - **Add New Item**.
3. Select **MultiRow 8.0 Template** from the list and click the **Add** button.
4. Verify that the template has been added to the Solution Explorer.

![Solution Explorer](image)

5. Build the project to access the template from the form.

Design the layout in the MultiRow designer using the toolbox and property window just like you would in a form designer. Refer to **Using the Designer** for details on using the template designer.

- The **MultiRow 8.0 Template** is not displayed in the Visual Studio Express Edition. Visual Studio Express Edition does not support IDE enhancement so templates can be created in code only. For details on creating templates in code, refer to **Template**.
- If the previous version, MultiRow for Windows Forms 6.0, is already referenced in the project, then selecting "MultiRow 8.0 Template", adds the 6.0 template. This is because the previous version and 8.0 cannot be used in the same project. In this case, upgrade the project to use 8.0 first.

### Assigning a Template to the Control

Follow the steps below to assign a template to the GcMultiRow control on the form.

1. Select the GcMultiRow control from the toolbox and drag and drop it on the form.
2. Select the GcMultiRow control and click on Smart Tag.

![GcMultiRow Tasks](image)

3. Open the **Choose Template** drop-down list in the Smart Tag.
4. Select the desired template from the drop-down list.

Build the project if the added template is not displayed in the drop-down list.

The template can also be assigned to the control using the GcMultiRow."Template Property' in the online documentation) property.

Adding the MultiRow Component to a Visual Studio 2013 Project

Follow the steps below to add GcMultiRow to a Visual Studio 2013 project.

Creating a Project

1. Start Visual Studio .NET.
2. From the File menu, choose New, Project or select New Project under Start.
3. In the New Project dialog, in the Installed area, select a project type depending on the language environment in which you are developing. For example, choose Windows under Visual Basic.
a. Choose the type of project such as Windows Forms Application.
b. In the Name box, type the name of the new project. The default is WindowsApplication1 for the first Windows Forms application.
c. In the Location box, leave the location path as the designated path, or click Browse to change the path to a new directory.
d. Click OK.

If your project does not display the Solution Explorer, from the View menu, choose Solution Explorer.

4. In the Solution Explorer, right-click on the form name, Form1. Choose Rename from the pop-up menu, then type the new form name you prefer for the new form name.

Adding the Component to the Toolbox

The next step is to add the component to the toolbox. This only has to be done once.

1. If the Toolbox is not displayed, from the View menu choose Toolbox.
2. Once the Toolbox is displayed, look in the GrapeCity Spread category (or in any other category if you have installed MultiRow and placed the toolbox icon in a different category).
3. If the MultiRow component is not in the Toolbox, right-click in the Toolbox, and from the pop-up menu choose Choose Items.
4. In the Customize Toolbox dialog, click the .NET Framework Components tab.
5. In the .NET Framework Components tab, the GcMultiRow component (in the GrapeCity.Win.MultiRow namespace) should be displayed in the list of components. Select the GcMultiRow component check box and click OK. If the GcMultiRow component is not displayed in the list of components, click Browse and browse to the installation path for the MultiRow Windows Forms component. Once there, select the GrapeCity.Win.MultiRow.dll and click Open. The GcMultiRow component is now displayed in the list of components. Select it and click OK.
Adding the Component

The next step is to add the MultiRow component to a project.

1. With an open project, in the **Toolbox** under Windows Forms (or whatever category to which you added it), select the GcMultiRow component.

2. On your Windows Forms page, draw a MultiRow component by dragging a rectangle the size that you would like the initial component or simply double-click on the page. The MultiRow component appears.
3. The GcMultiRow Tasks dialog is displayed. Add a template to the control by selecting an existing template such as default template with the Choose Template option. Close the task dialog.

Your project should now look similar to the picture shown here.

Using the above steps, you have created an application with a very simple tabular interface. Run the application. It looks like a
Creating an Application using GcMultiRow

The following steps create a simple application with a tabular screen that has multiple rows for one record.

Create a Project and Add the Component to the Project

Create a project where the GCMultiRow control can be used. This step can be skipped if GcMultiRow is already registered in the toolbox.

2. Select Choose Toolbox Items from the Tools menu in Visual Studio.
3. Click the .NET Framework Components tab in the Choose Toolbox Items dialog.
4. Enter GcMultiRow in the dialog filter to narrow down the listed options.
5. Select the GcMultiRow component checkbox from the list.
6. Click OK in the dialog box.

After these steps, you can start using GcMultiRow. Check that the GcMultiRow icon has been added to the toolbox.

If the toolbox is not displayed on the screen, choose Toolbox from the View menu.

Placing the Grid Control

Open the project form created above and add the GcMultiRow control to this form.

1. Select GcMultiRow from the toolbox and drag and drop it onto the form.
2. Resize the GcMultiRow control (for example: GcMultiRow1) to the desired size.
3. Select the GcMultiRow control on the form. Click on the Smart Tag and from the Choose Template drop-down, select Default Template.
4. Save the project and run it.

Using the above steps, you have created an application with a very simple tabular interface. Run the application. It looks like a tabular calculation application. You can move from one cell to another using the keyboard, input values into cells, select cells with the mouse, and change them to an editable state using a double-click.
Using the following steps, change the layout of the tabular screen.

**Editing Templates using the Designer**

In MultiRow, the layout of the tabular screen is managed using an object called *Template*. Templates provide developers the freedom to design their layouts since they allow cells of the desired size to be placed at any location. The created layout can then be read into the GcMultiRow control and shown as a tabular screen.

1. Select the GcMultiRow control on the form. Click on the Smart Tag and from the Choose Template drop-down, select **Add New Template**.
2. Click **Next** on the Welcome to MultiRow 7.0 Template Wizard page.
3. In the Choose a Template Type page, select the Unbound Template option and click **Next**.
4. Choose Top to Bottom for setting the layout of the template, and click the **Next** button.
5. Click Next on the Design Template Layout page.
6. Click Next on the Choose a Style page.
7. Click **Finish** on the Complete the MultiRow 7.0 Template Wizard page.

An editable template is added to the project. Check that a new template (for example: Template1.vb) has been added to the project. It can be edited with the designer just like you would edit a form.

**Connecting to SQL Server Database**

MultiRow can connect to a SQL Server database using ADO.NET in the same way as the standard .NET Framework controls.

**Assumptions**

This topic assumes that the following are being used:

- Microsoft SQL Server 2005 or 2008
- Northwind and pubs Sample Databases ([Download Northwind sample database and pubs sample database](http://msdn.microsoft.com/library/ms143221.aspx))

**Reference: Installing Sample Database**

The sample database is not installed by default with Microsoft SQL Server. After installing Microsoft SQL Server, refer to the Microsoft SQL Server documentation and MSDN to install the sample database. An example of the installation steps is provided below.

1. Install Microsoft SQL Server. The steps below are for the Express Edition but most of them can be used irrespective of the Visual Studio version.
2. [Download Northwind and pubs Sample Databases for SQL Server 2000](http://www.microsoft.com/downloads/details.aspx?FamilyId=06616212-0356-46A0-8DA2-EEBC53A68034). This database is for Microsoft SQL Server 2000, but can also be used with Microsoft SQL Server 2005 or 2008.
3. Install the downloaded samples. The default installation folder is C:\SQL Server 2000 Sample Databases.
4. Install Microsoft SQL Server Management Studio Express.
5. Run Microsoft SQL Server Management Studio Express as Administrator and connect to Microsoft SQL Server. Example of server name: (local)\SQLEXPRESS
6. Select Databases from the tree on the left and click on the Attach menu of the context menu.
7. Click the Add button on the Attach Databases page.
8. On the Locate Database Files page, select C:\SQL Server 2000 Sample Databases\NORTHWND.MDF and click OK.
9. Click OK on the Attach Databases page.
10. Expand databases in the tree on the left and check if Northwind has been added.

Connecting to Database

Register the SQL Server datasource into the project. This operation is not dependent on MultiRow.

2. Click Data - Add New Data Source in Visual Studio.
3. On the Data Source Configuration Wizard page, select Database and click Next.
4. On the Choose Your Data Connection page, click on New Connection and add a connection with the following configuration.
   - Data source - Microsoft SQL Server (SqlClient)
   - Server name (Example) - (local)\SQLEXPRESS
   - Select or enter a database name - Northwind
5. After adding the connection, click Next.
6. On the Save the Connection String to the Application Configuration File page, click Next.
7. On the Choose your Database Objects page, check the following table:
   - Orders
8. Click **Finish**.

9. Check that Orders has been added to the **Data Sources** window of Visual Studio.

---

**Designing Templates**

Create a template for MultiRow based on the database definition.

1. Click **Project - Add New Item** in Visual Studio.

2. On the **Add New Item** page, select the **MultiRow 7.0 Templates** and click on the **Add** button.
3. Check that the Template1.vb or Template1.cs design page is displayed.

4. Click **Data - Show Data Sources** in Visual Studio.
5. Select the **Orders** table under the **NorthwindDataSet** from the tree in the **Data Sources** window. Drag and drop it to the **Row** area while pressing the **Ctrl** key.

6. Double-click on the edges of the row and adjust the height of the row.

7. Click on **Template - CellDisplayMode - DataField** in Visual Studio and verify that the data fields have been assigned to the cells.
8. Save the project and use the next set of steps provided in Binding the Grid to Data.

**Binding the Grid to Data**

Place the GeMultiRow control on the form and assign the data source and template to the grid.

1. Open the design view of the form (Form1).
2. Select the **Orders** table from the **Data Sources** window and click **Customize...** from the drop-down list.
3. On the **Options** page that is displayed, check **GeMultiRow** in the list of **Associated controls**.
4. Close the dialog by clicking on the **OK** button.
5. Select the **Orders** table, and click on **GcMultiRow** in the drop-down list of the **Data Sources** window.
6. Drag and drop the **Orders** table onto the design view of the form (Form1).
7. Verify that the BindingNavigator (OrdersBindingNavigator) and GcMultiRow (OrdersGcMultiRow) have been placed on the form.
8. Click on **Build - Build Solution** in Visual Studio.

9. Select the OrdersGcMultiRow control on the form and select **Template1** from the Choose Template option in Smart Tag.

A template and data source have been assigned to the grid using the above steps. When you run the project, you can see that the values from the datasource have been read into the grid.
Creating an Application Using the Designer (Grid View)

In MultiRow, you can create a layout in the designer that can be used in applications as a template. You can create an application with a grid type layout for displaying employee information, as shown below.

Creating a Template

This example shows how to place a GcMultiRow control on the form of a project and create a template. If the GcMultiRow control is not added in the toolbox, please refer to Creating an Application using GcMultiRow.
1. Start Visual Studio, and create a new Windows Forms application project.
2. Select **GcMultiRow** from the toolbox, and drag and drop it onto the form.
3. Select the GcMultiRow control (Example: GcMultiRow1) that you placed, and select the **Dock in parent container** smart tag.
4. Select the GcMultiRow control you placed, and select the **Add New Template** smart tag.

5. In the Wizard that gets displayed, click on the **Next** button.
6. In **Choose a template type**, select **Unbound Template**.

7. In the next screen, select **Top to Bottom** as the Layout Mode.
8. Perform the following settings in Design Template Layout:
   - Check the Generate ColumnHeader checkbox.
   - Check the Generate RowHeader checkbox.
   - Set Horizontal Cells Count to 8.
   - Set Vertical Cells Count to 2.

9. Click Finish to exit the Wizard.

10. Select the GcMultiRow control, then select the Edit Template smart tag, and check that the template is displayed.

Using the above steps, you have created a template with a basic layout.

The next step changes the layout of the tabular screen.
Editing Templates Using the Designer

Since the template that you created in the wizard has a very basic layout, you can edit the template in the designer in order to get the layout that you would like to display in the application.

1. To remove the cells that are not needed in the layout, select the three cells in the row (textBoxCell4, textBoxCell6, textBoxCell9), and press the Delete key.

2. Resize the cells in order to fill the space created due to deletion of the cells. You can resize textBoxCell1 from the bottom, and textBoxCell3 and textBoxCell5 from the right.

3. Similarly, correct the layout of the cells arranged in the column header section.

4. To replace the cell type, select textBoxCell1, and select Replace - Common Cells - ImageCell, from the context menu (right-click menu). The Cell name will be changed to imageCell1.

5. In the same way, replace the following cell types:
6. In the Properties window, set the **ShowSpinButton** property of numericUpDownCell1 and numericUpDownCell2, to **NotShown**.
7. In the Properties window, set the **ShowDropDownButton** property of dateTimePickerCell1 and dateTimePickerCell2, to **NotShown** and set **Short** in the **Format** property.
8. Set the following values in the **Value** property of each cell that is placed in the column header section.

<table>
<thead>
<tr>
<th>Cell Name</th>
<th>Setting value of the Value property</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnHeaderCell1</td>
<td>Photo</td>
</tr>
<tr>
<td>columnHeaderCell2</td>
<td>Name</td>
</tr>
<tr>
<td>columnHeaderCell3</td>
<td>Phonetic</td>
</tr>
<tr>
<td>columnHeaderCell5</td>
<td>Address</td>
</tr>
<tr>
<td>columnHeaderCell7</td>
<td>Department</td>
</tr>
<tr>
<td>columnHeaderCell8</td>
<td>Date of Joining</td>
</tr>
<tr>
<td>columnHeaderCell10</td>
<td>Date of Birth</td>
</tr>
<tr>
<td>columnHeaderCell11</td>
<td>Gender</td>
</tr>
<tr>
<td>columnHeaderCell12</td>
<td>Enrolled Branch</td>
</tr>
<tr>
<td>columnHeaderCell13</td>
<td>ZipCode</td>
</tr>
<tr>
<td>columnHeaderCell14</td>
<td>Region</td>
</tr>
<tr>
<td>columnHeaderCell15</td>
<td>Phone</td>
</tr>
<tr>
<td>columnHeaderCell16</td>
<td>Extension</td>
</tr>
</tbody>
</table>

9. Right-click the mouse anywhere in the template, and add a column header section (columnHeaderSection2) by selecting **Add - ColumnHeader** from the context menu.

10. Click on the title bar of columnHeaderSection2, and move it to the top using drag and drop.
11. Select **LabelCell** from the toolbox, and drag and drop it to columnHeaderSection2.

12. In the Properties window, perform the following settings for the LabelCell (labelCell1) that you placed.  
   - Set [0, 0] in the **Location** property.
   - Set **Employee Details** in the **Value** property.
   - Click the ... button of the **Style.Font** property, and set the **Font Name** property to **Arial**, **Style** property to **Bold**, and **Size** property to **18**.
   - Set the **Style.TextAlign** property to **MiddleCenter**.

13. Resize the labelCell1 to the same size as that of columnHeaderSection2.

### Connecting to the Database

Next, perform the connection settings for the database. In this example, use the Sample.mdb placed under the product samples.

- Sample.mdb is located in the SampleCS folder (or the SampleVB folder) included in the product samples folder.

1. Click the **Project - Add New Data Source** menu in Visual Studio.
2. In the **Choose a Data Source type** screen of the **Database Configuration Wizard** that is displayed, select **Database**, and click the **Next** button.
3. In the **Choose a Database Model** screen, select **Data set**, and click the **Next** button.
4. In the **Choose Your Data Connection** screen, click on **New Connection**, and add a connection with the following configuration.
   - Data Source: Microsoft Access Database File
   - Database file name: Sample.mdb

5. After adding the connection, click the **Next** button.
6. Click on the **Next** button in the **Save the Connection String to the Application Configuration File** screen.
7. In the **Choose Your Database Objects** screen, check the checkbox of the following table:
   - Employees

8. Click on **Finish**.
9. Check that **Employees** has been added to the **Data Source** window of Visual Studio.
10. Set the following values in the **DataField** property of each cell that is placed in the **Row**.
<table>
<thead>
<tr>
<th>Cell Name</th>
<th>Setting value of the DataField property</th>
</tr>
</thead>
<tbody>
<tr>
<td>rowHeaderCell1</td>
<td>Employee Code</td>
</tr>
<tr>
<td>imageCell1</td>
<td>Photo</td>
</tr>
<tr>
<td>textBoxCell2</td>
<td>Name</td>
</tr>
<tr>
<td>textBoxCell3</td>
<td>Phonetics</td>
</tr>
<tr>
<td>textBoxCell5</td>
<td>Address</td>
</tr>
<tr>
<td>numericUpDownCell1</td>
<td>Department</td>
</tr>
<tr>
<td>dateTimePickerCell1</td>
<td>Date of Joining</td>
</tr>
<tr>
<td>dateTimePickerCell2</td>
<td>Date of Birth</td>
</tr>
<tr>
<td>textBoxCell11</td>
<td>Gender</td>
</tr>
<tr>
<td>textBoxCell12</td>
<td>Enrolled Branch</td>
</tr>
<tr>
<td>textBoxCell13</td>
<td>ZipCode</td>
</tr>
<tr>
<td>numericUpDownCell2</td>
<td>RegionCode</td>
</tr>
<tr>
<td>textBoxCell15</td>
<td>Phone</td>
</tr>
<tr>
<td>textBoxCell16</td>
<td>Extension</td>
</tr>
</tbody>
</table>

11. Click the **View - Other Windows - Data Sources** menu of Visual Studio.
12. From the tree in the **Data Source** window, drag and drop the **Employees** table, which is placed under **SampleDataSet**, into the area of the GcMultiRow control.

![Visual Studio Data Source](image)

13. In the **Properties** window, set the **GcMultiRow1.AllowUserToAddRows** property to False.
14. If you run the project, the values of the data source are loaded into the grid.
15. Adjust the size and position of the cells, using the designer.

**Setting Sorting and Filtering**

You can set sorting and filtering using the ColumnHeaderCell function.

The following steps perform sorting of **Date of Joining** and filtering by **Enrolled Branch**.

1. In the **Design** screen of the template, select the columnHeaderCell12 (Enrolled Branch).
2. From the drop-down list of the **DropDownList** property in the **Properties** window, select the **Filter and Sort DropDownList**.
3. Expand the child level of the **DropDownList** property, and select **textBoxCell12** from the drop-down list of **CellName** property.
4. Select the columnHeaderCell8 (Date of Joining).
5. Set the **SelectionMode** property in the **Properties** window to **None**.
6. Set the **SortMode** property in the **Properties** window to **Automatic**.
7. You can use sorting and filtering when running the project.

Creating an Application Using the Designer (Single Template View)

This topic shows how to create an application with a single template layout using the same data as that in Creating an Application Using the Designer (Grid View).
Creating a Template

This example shows how to place a GcMultiRow control on the form of a project and create a template.

If the GcMultiRow control is not registered in the toolbox, please refer to Creating an Application using GcMultiRow.

1. Start Visual Studio, and create a new Windows Forms application project.
2. In the Properties window, set the Size property of Form1 to (690, 370).
3. From the toolbox, select Panel, and drag and drop it onto the form.
4. Select the Panel control (for example: Panel1) placed on the form, and perform the following settings in the Properties window.
   - Anchor property: Top, Bottom, Left, Right
   - Location property: (12, 12)
   - Size property: (650, 280)
5. From the toolbox, select Button and drag and drop it at the bottom of the form. Place four Button controls in all.
6. In the Properties window, set the following values in the Anchor property and Text property of the Button control.

<table>
<thead>
<tr>
<th>Button control</th>
<th>Anchor property</th>
<th>Text property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Button1</td>
<td>Bottom, Left</td>
<td>First</td>
</tr>
<tr>
<td>Button2</td>
<td>Bottom, Left</td>
<td>Next</td>
</tr>
<tr>
<td>Button3</td>
<td>Bottom, Left</td>
<td>Previous</td>
</tr>
<tr>
<td>Button4</td>
<td>Bottom, Left</td>
<td>Last</td>
</tr>
</tbody>
</table>
7. Select GcMultiRow from the toolbox, and drag and drop it to Panel1.
8. Select the GcMultiRow control (for example: GcMultiRow1) that you placed, and then select the Dock in Parent container smart tag.
9. Select the GcMultiRow control that you placed, and then select the Add New Template smart tag.
10. In the Wizard that gets displayed, click Next.
11. In Choose a template type, select Empty template.
12. Click Finish to exit the Wizard.
13. Select the GcMultiRow that you placed, and then select the Edit Template smart tag. Check that the template is displayed.

Using the above steps, you have performed the template settings.

The next step is to create the screen layout.

Placing Cells using the Designer

Since the template created in the Wizard is an empty template, you need to place the cells one by one.

1. Right click the mouse anywhere in the ColumnHeaderSection1, and select Delete from the context menu.
2. Select Row, and set the following settings in the Properties window.
   - BackColor property: Silver
   - Height property: 250
   - Width property: 650
3. Add a RowHeaderCell to the template from the toolbox, and set the following values to each of its properties.

<table>
<thead>
<tr>
<th>Name property</th>
<th>Location property</th>
<th>Size property</th>
<th>Text property</th>
</tr>
</thead>
</table>

Copyright © GrapeCity, inc. All rights reserved.
4. Select all the cells using drag and drop.

5. Set the following settings in the **Properties** window.
   - FlatStyle property: Flat
   - ResizeMode property: None
   - SelectionMode property: None
   - ShowIndicator property: False

6. Select **Template1** from the drop-down list in the **Properties** window, and set the following settings.
   - RowsDefaultCellStyle.Font.Name : Arial
   - RowsDefaultCellStyle.TextAlign : MiddleCenter

7. From the toolbox, add TextBoxCells to the template and set the following values in each of its properties.

<table>
<thead>
<tr>
<th>Name property</th>
<th>Location property</th>
<th>Size property</th>
</tr>
</thead>
<tbody>
<tr>
<td>TextBoxCell1</td>
<td>120, 8</td>
<td>348, 24</td>
</tr>
<tr>
<td>TextBoxCell2</td>
<td>120, 32</td>
<td>118, 25</td>
</tr>
<tr>
<td>TextBoxCell3</td>
<td>328, 33</td>
<td>140, 25</td>
</tr>
<tr>
<td>TextBoxCell4</td>
<td>120, 57</td>
<td>118, 26</td>
</tr>
<tr>
<td>TextBoxCell5</td>
<td>328, 57</td>
<td>140, 25</td>
</tr>
<tr>
<td>TextBoxCell6</td>
<td>198, 156</td>
<td>98, 31</td>
</tr>
<tr>
<td>TextBoxCell7</td>
<td>198, 108</td>
<td>270, 24</td>
</tr>
<tr>
<td>TextBoxCell8</td>
<td>198, 132</td>
<td>270, 24</td>
</tr>
<tr>
<td>TextBoxCell9</td>
<td>374, 156</td>
<td>94, 31</td>
</tr>
<tr>
<td>TextBoxCell10</td>
<td>119, 187</td>
<td>349, 58</td>
</tr>
</tbody>
</table>
8. Select all by clicking the TextBoxCell, while holding down the **Ctrl** key, and set the following settings in the **Properties** window.

   - **Style.BackColor** property: White

9. From the toolbox, add a DateTimePickerCell to the template and set the following values in each of its properties.

<table>
<thead>
<tr>
<th>Name property</th>
<th>Location property</th>
<th>Size property</th>
</tr>
</thead>
<tbody>
<tr>
<td>DateTimePickerCell1</td>
<td>120, 83</td>
<td>118, 25</td>
</tr>
<tr>
<td>DateTimePickerCell2</td>
<td>328, 82</td>
<td>140, 26</td>
</tr>
</tbody>
</table>

10. Select all by clicking the DateTimePickerCell, while holding down the **Ctrl** key, and set the following settings in the **Properties** window.

   - **ShowDropDownButton** property: NotShown
   - **Style.BackColor** property: White

11. From the toolbox, add an ImageCell to the template and set the following values in each of its properties.

<table>
<thead>
<tr>
<th>Name property</th>
<th>ImageLayout property</th>
<th>Location property</th>
<th>Size property</th>
<th><strong>Style.BackColor</strong> property</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImageCell1</td>
<td>Stretch</td>
<td>468, 33</td>
<td>175, 186</td>
<td>Control</td>
</tr>
</tbody>
</table>

12. From the toolbox, add a LabelCell to the template and set the following values in each of its properties.

<table>
<thead>
<tr>
<th>Name property</th>
<th>Location property</th>
<th>Size property</th>
<th><strong>Style.BackColor</strong> property</th>
</tr>
</thead>
<tbody>
<tr>
<td>LabelCell1</td>
<td>468, 219</td>
<td>175, 26</td>
<td>Control</td>
</tr>
</tbody>
</table>

Using the above steps, you have placed cells in the template.

**Setting the Cell Border**

Use the following steps to set the borders of the cells that you placed on the template.

1. Right click on RowHeaderCell1, and select **Border** from the Context Menu.

2. In **Choose Border Type** of the **Border** dialog, set **RoundedBorder**.
3. Set the following values for each item.
   - **Style**: Double
   - **Color**: Silver
   - **RoundCorner**: 0.40

4. Using the border buttons, set the left, top, and top-left borders.

5. Change the **Style** to Thin, and set the right and bottom borders.

6. If you close the **Borders** dialog by clicking the **OK** button, rounded borders are set on the top left.

7. Use the same steps to set border related settings for other cells.
Using the above steps, you have set the cell borders.

**Connecting to the Database**

Next, follow the steps below to set the connection settings of the database. For more information on how to add a data source, refer to *Creating an Application Using the Designer (Grid View)*.

1. After **Employees** has been added to the **Data Sources** window of Visual Studio, set the following values for the **DataField** property of each cell placed in the rows.

<table>
<thead>
<tr>
<th>Cell Name</th>
<th>DataField Property Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>textBoxCell1</td>
<td>Employee Code</td>
</tr>
<tr>
<td>textBoxCell2</td>
<td>Name</td>
</tr>
<tr>
<td>textBoxCell3</td>
<td>Phonetic</td>
</tr>
<tr>
<td>textBoxCell4</td>
<td>Enrolled Branch</td>
</tr>
<tr>
<td>textBoxCell5</td>
<td>Department</td>
</tr>
<tr>
<td>dateTimePickerCell1</td>
<td>Date of Joining</td>
</tr>
<tr>
<td>dateTimePickerCell2</td>
<td>Date of Birth</td>
</tr>
<tr>
<td>textBoxCell6</td>
<td>Address</td>
</tr>
<tr>
<td>textBoxCell7</td>
<td>Address</td>
</tr>
<tr>
<td>textBoxCell8</td>
<td>Phone</td>
</tr>
<tr>
<td>textBoxCell9</td>
<td>Extension</td>
</tr>
<tr>
<td>imageCell1</td>
<td>Photo</td>
</tr>
</tbody>
</table>

2. Click the **View - Other Windows - Data Sources** menu of Visual Studio.

3. Select the **Employees** table under **SampleDataSet**, from the tree in the **Data Sources** window, and drag and drop it into the area of the GaMultiRow control.

4. If you run the project, the values of the data source are loaded into the grid.
Using Code

The code below implements the single template view by changing the size of the GcMultiRow control, when the size or zoom of GcMultiRow control is changed.

[VB]

Private Sub Form1_Load(sender As System.Object, e As System.EventArgs) Handles MyBase.Load
    EmployeesTableAdapter.Fill(SampleDataSet.Employees)
    GcMultiRow1.MultiSelect = False
    GcMultiRow1.AllowUserToAddRows = False
    GcMultiRow1.ScrollBars = ScrollBars.None
    GcMultiRow1.DataSource = EmployeesBindingSource
    UpDataGcMultiRowBounds()
End Sub

Private Sub UpDataGcMultiRowBounds()
    If Me.gcMultiRow1.Template Is Nothing OrElse Me.gcMultiRow1.Rows.Count &< 1 Then Return
    Dim mrWidth As Integer = Me.gcMultiRow1.SectionWidth
    Dim mrHeight As Integer = Me.gcMultiRow1.Rows(0).Height
    ' Zoom
    mrWidth = CType(Math.Ceiling(mrWidth * Me.gcMultiRow1.ZoomFactor), Integer)
    mrHeight = CType(Math.Ceiling(mrHeight * Me.gcMultiRow1.ZoomFactor), Integer)
    ' Border Size Zoom
    mrWidth += 4
    mrHeight += 4
    ' Fit to Panel size
    If mrWidth > Me.panel1.Width Then
        mrWidth = Me.panel1.Width
    End If
    If mrHeight > Me.panel1.Height Then
        mrHeight = Me.panel1.Height
    End If
    ' Placement position
    Dim x As Integer = (Me.panel1.Width - mrWidth) / 2
    Dim y As Integer = (Me.panel1.Height - mrHeight) / 2
    ' Adjustment of the position
    If x &< 0 Then x = 0
    End If
    If y &< 0 Then y = 0
    End If
    Me.gcMultiRow1.Location = New Point(x, y)
End Sub

Private Sub gcMultiRow1_ZoomFactorChanged(ByVal sender As Object, ByVal e As EventArgs) Handles GcMultiRow1.ZoomFactorChanged
    Me.UpDataGcMultiRowBounds()
End Sub

Private Sub GcMultiRow1_Resize(sender As System.Object, e As System.EventArgs) Handles GcMultiRow1.Resize
    Me.UpDataGcMultiRowBounds()
End Sub

[CS]

private void Form1_Load(object sender, EventArgs e)
{
    EmployeesTableAdapter.Fill(sampleDataSets.Employees);
    gcMultiRow1.Template = new Template1();
    gcMultiRow1.MultiSelect = false;
    gcMultiRow1.AllowUserToAddRows = false;
    gcMultiRow1.AllowUserToAddRows = false;
    gcMultiRow1.ScrollBars = ScrollBars.None;
    gcMultiRow1.DataSource = EmployeesBindingSource;
    UpDataGcMultiRowBounds();
}

private void UpDataGcMultiRowBounds()
{
    if (gcMultiRow1.Template == null || gcMultiRow1.Rows.Count &lt; 1)
    {
        return;
    }
}
// Set the size of MultiRow
int mrWidth = gcMultiRow1.SectionWidth;
int mrHeight = gcMultiRow1.Rows[0].Height;
// Zoom mrWidth = (int)Math.Ceiling(mrWidth * gcMultiRow1.ZoomFactor);
mrHeight = (int)Math.Ceiling(mrHeight * gcMultiRow1.ZoomFactor);
// Border Size
mrWidth += 4; mrHeight += 4;
// Fit to Panel size
if (mrWidth > panel1.Width)
    { mrWidth = panel1.Width; }
if (mrHeight > panel1.Height)
    { mrHeight = panel1.Height; }
gcMultiRow1.Size = new Size(mrWidth, mrHeight);
// Placement position
int x = (panel1.Width - mrWidth) / 3;
int y = (panel1.Height - mrHeight) / 2;
// Adjustment of the position
if (x < 0)
    { x = 0; }
if (y < 0)
    { y = 0; }
gcMultiRow1.Location = new Point(x, y);
private void gcMultiRow1_ZoomFactorChanged(object sender, EventArgs e)
    { UpDataGcMultiRowBounds(); }
private void gcMultiRow1_Resize(object sender, EventArgs e)
    { UpDataGcMultiRowBounds(); }
}
private void Form1_Load(object sender, EventArgs e)
{ button1.Click += new EventHandler(button1_Click);
button2.Click += new EventHandler(button2_Click);
button3.Click += new EventHandler(button3_Click);
button4.Click += new EventHandler(button4_Click);
}
private void button1_Click(object sender, EventArgs e)
{ // Move to the first page
ScrollActions.VerticalScrollToFirstPage.Execute(gcMultiRow1);
}
private void button2_Click(object sender, EventArgs e)
{ // Move to the previous page
ScrollActions.ScrollUp.Execute(gcMultiRow1);
}
private void button3_Click(object sender, EventArgs e)
{ // Move to the next page
ScrollActions.ScrollDown.Execute(gcMultiRow1);
}
private void button4_Click(object sender, EventArgs e)
{ // Move to the last page
ScrollActions.VerticalScrollToLastPage.Execute(gcMultiRow1);
}

In addition, the following code switches the display data, by setting the ScrollActions ('ScrollActions Class' in the on-line documentation) property in the Click event of the Button.

[VB]

Private Sub Button1_Click(sender As System.Object, e As System.EventArgs) Handles Button1.Click
' Move to the first page
ScrollActions.VerticalScrollToFirstPage.Execute(GcMultiRow1)
End Sub
Private Sub Button2_Click(sender As System.Object, e As System.EventArgs) Handles Button2.Click
' Move to the previous page
ScrollActions.ScrollUp.Execute(GcMultiRow1)
End Sub
Private Sub Button3_Click(sender As System.Object, e As System.EventArgs) Handles Button3.Click
' Move to the next page
ScrollActions.ScrollDown.Execute(GcMultiRow1)
End Sub
Private Sub Button4_Click(sender As System.Object, e As System.EventArgs) Handles Button4.Click
' Move to the last page
ScrollActions.VerticalScrollToLastPage.Execute(GcMultiRow1)
End Sub

[CS]

class button1_Click
private void button1_Click(object sender, EventArgs e)
{ // Move to the first page
ScrollActions.VerticalScrollToFirstPage.Execute(gcMultiRow1);
}
private void button2_Click(object sender, EventArgs e)
{ // Move to the previous page
ScrollActions.ScrollUp.Execute(gcMultiRow1);
}
private void button3_Click(object sender, EventArgs e)
{ // Move to the next page
ScrollActions.ScrollDown.Execute(gcMultiRow1);
}
private void button4_Click(object sender, EventArgs e)
Designer Window

The following topics give a brief description about the designer windows. For details on working with the designer, refer to Using the Designer or Advanced Designer Features

- Integration with IDE
- Document Window
- Toolbox
- Menu Bar
- Toolbar
- Status Bar
- Properties Window
- Template Property List 7.0
- Template Explorer 7.0
- Template - Named Cell Styles 7.0
- Designer Options

Integration with IDE

The MultiRow designer is integrated into Visual Studio. This allows the product to share menu commands and operations with Visual Studio which makes it easier to use and learn. This topic explains issues related to integrating the MultiRow designer with Visual Studio.

Visual Studio Editions

Because Visual Studio 2008 Express Editions, Visual Studio 2010 Express Edition, Visual Studio 2012 Express, and Visual Studio 2013 Express do not support IDE extendibility, the MultiRow designer cannot be integrated with the Visual Studio IDE in these versions. You can create templates using the GcMultiRow control and code with these editions.

Upgrading Visual Studio

MultiRow needs to be uninstalled and reinstalled when the version or edition of Visual Studio is upgraded. Examples of
upgrades:

- Upgrade from Visual Studio 2008 to Visual Studio 2010
- Adding Visual Studio 2010 to an environment that has Visual Studio 2008 installed on it

Upgrading Windows

When upgrading or reinstalling Windows, uninstall MultiRow before upgrading. Once you are done with the upgrade, reinstall MultiRow with the MultiRow installer supported by the newly installed Windows. If required, deactivate the license before uninstalling MultiRow. Examples of upgrades:

- Upgrade Windows XP to Windows Vista

Unless specified, the installation of a Windows Service Pack is not considered to be an upgrade.

Order while installing with Visual Studio

The MultiRow designer will not be installed properly if the product is installed before installing Visual Studio. Uninstall MultiRow and reinstall it after installing Visual Studio. Also, exit all Visual Studio applications before running the MultiRow installer.

General Troubleshooting

If you are no longer able to use the MultiRow designer in Visual Studio, uninstalling and then reinstalling MultiRow may solve the issue.

Document Window

The document window allows you to see the design view, runtime view, and print preview of a template. The view can be changed using tabs provided at the bottom of the window.

Appearance

The ruler displays tick marks that are the standard for the design view. You can set the unit of these tick marks with **Designer Options**. The default unit is pixel. When you drag over an area of this ruler using the mouse, the cells inside
the dragged area are selected. When an edge of the tick mark is dragged, it changes the size of the section.

**Section Title Bar**

The section title bar displays the header or row name. The section can be collapsed by clicking on the mark on the left side of the icon. Click the mark again to display the section.

If the section width is less than the section name, part of the section name is clipped and is not displayed. The complete name is displayed in the tooltip when the mouse cursor is over the section.

**Tabs**

The designer **Display Modes of the Designer** can be changed by clicking on the tabs.

**Version Information**

The version of the designer (File version) is displayed in the bottom right corner of the document window.

**Toolbox**

The list of cell types available in the template is displayed in the designer toolbox. The cells can be placed by dragging and dropping the cell types from the toolbox onto the Document window. You can also register the user-defined cell types in the toolbox. This operation is similar to using the form designer.

**Common Cells 7.0**

Common cells is a list of cell types that can be used in the default state. For more details on each cell type, refer to **Cell Types**.

- ButtonCell
- CheckBoxCell
- ColumnHeaderCell
- ComboBoxCell
- CornerHeaderCell
- DateTimePickerCell
- DomainUpDownCell
- FilteringTextBoxCell
- HeaderCell
- ImageCell
- LabelCell
- LinkLabelCell
- ListBoxCell
- ListLabelCell
- MaskedTextBoxCell
Design Time Components 7.0

Design time components provide design functions to support the alignment of cells.

- Table

Windows Forms Components

Windows Forms components provide invisible objects like images, data sources, dialog boxes, and so on that are used with cells. The functions of individual objects are the same as in Windows forms.

- BindingSource
- ColorDialog
- DataSet
- FolderBrowserDialog
- FontDialog
- ImageList
- OpenFileDialog
- PageSetupDialog
- PrintDialog
- SaveFileDialog

User-defined Cells

Newly created cells and cells inherited from default cell types can be added to the toolbox. For more details on how to create user-defined cells, refer to User-Defined Cell.

Menu Bar

A Template menu is displayed in the Visual Studio menu bar when the MultiRow Document window is active.
A Template menu is displayed in the Visual Studio menu bar when the MultiRow Document window is active. The following commands can be executed in the Template menu.

- **Input and Output of Template XML**
- Adding a column header section and column footer section - Working with Column Headers in the Designer, Working with Column Footers in the Designer
- Switching Display Modes of the Designer
- Switching Cell Display Modes in the Designer

**Toolbar**

The MultiRow 7.0 Designer toolbar and the Visual Studio toolbar can be used when the MultiRow Document Window is active.

**Items**

A few of the toolbar commands are valid only when a cell is selected in the Document Window.

If the MultiRow 7.0 Designer toolbar is not displayed, follow the steps below to display the toolbar.

- In Visual Studio, open View - Toolbars - MultiRow 7.0 Designer.
- In Visual Studio, open View - Toolbars - Formatting.

**Status Bar**

The designer status bar is similar to the Visual Studio status bar.

**Items Displayed on Status Bar**

The coordinates and size of the selected objects are displayed in the status bar. For example, when a cell is selected, the
values of the Cell.Location property and Cell.Size property are displayed.

Properties Window

This is a window provided by Visual Studio to edit properties. The behavior and usage of this dialog are similar to how you would edit a form in Visual Studio.

Appearance

You can modify the properties of a selected cell, section, or template in the Properties Window.

Template Property List 7.0

Template Property List 7.0 provides a list of cells and their main properties. Properties of multiple cells can be compared and modified in a tabular form using this window. This window can be docked at any desired position in Visual Studio.

View the Template Property List by selecting the Other Windows - Template PropertyList 7.0 option from the Visual Studio View menu.
**Template Property List** of MultiRow for Windows Forms 6.0 is not supported in 7.0. The **Template Property List 7.0** is used in 7.0.

### ToolBar

<table>
<thead>
<tr>
<th>Icon</th>
<th>Text</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Select DataSource</td>
<td>Selects a data source from the choices displayed in the drop-down list of the DataField column combobox.</td>
</tr>
<tr>
<td></td>
<td>Cells in Active Section</td>
<td>Displays only cells of the active section when turned ON. Displays cells of all sections when turned OFF.</td>
</tr>
<tr>
<td>-</td>
<td>Freezing Columns</td>
<td>Specifies the columns to be displayed as frozen columns in the <strong>Template Property List</strong>.</td>
</tr>
<tr>
<td>-</td>
<td>CellName Filter</td>
<td>Displays only those cells which contain the string entered in the filter in the cell name.</td>
</tr>
<tr>
<td>-</td>
<td>Display number</td>
<td>Displays the total number of cells, and the number of cells after filtering.</td>
</tr>
</tbody>
</table>

### Columns list

The following cell properties can be displayed as columns in the template property list.

- Cell.Name
- Cell.Value
- Cell.DataField
- Cell.Enabled
- Cell.ReadOnly
- Cell.Visible
- Cell.TabIndex
- Cell.TabStop
- Cell.CellIndex
- Cell.ResizeMode
- Cell.Selectable
- Cell.TooltipText
- Cell.SectionName

### Data Sources Window

Desired tables from the project data source can be selected from the **Data Sources** window. The table selected here is displayed in the drop-down list of the DataField column as an input option.
You use this window to modify the columns to be displayed, set the alignment of the displayed columns, set the column width, and so on.

**Filter**
Cells displayed in the list can be narrowed down to show only the cells of the currently active section by clicking on the funnel icon 🍀. Entering the desired string in the Filter Cell Names text box displays only those cells that contain the entered string in the cell names.

**Template Explorer 7.0**

You can see the list of sections and cells in a hierarchical structure when using the Template Explorer 7.0 while the MultiRow Document window is active. Components included in the template are not displayed in Template Explorer 7.0.

View the Template Explorer by selecting the Other Windows - Template Explorer 7.0 option from the Visual Studio View menu.

> The MultiRow for Windows Forms 6.0 Template Explorer is not supported in 7.0. Use Template Explorer 7.0 instead.

**Appearance**

This window can be docked to any desired position within Visual Studio.

![Template Explorer 7.0](image)

**ToolBar**

<table>
<thead>
<tr>
<th>Icon</th>
<th>Text</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Type Name Display Style</td>
<td>Switches the display style of the type name (None, Short, Full).</td>
</tr>
<tr>
<td></td>
<td>Expand All</td>
<td>All items of the tree are expanded and displayed.</td>
</tr>
<tr>
<td></td>
<td>Collapse All</td>
<td>All items of the tree are collapsed and hidden.</td>
</tr>
<tr>
<td></td>
<td>Move Down</td>
<td>Moves the selected item down by one position in the tree.</td>
</tr>
<tr>
<td></td>
<td>Move Up</td>
<td>Moves the selected item up by one position in the tree.</td>
</tr>
<tr>
<td></td>
<td>Settings</td>
<td>Displays designer settings. For more details on settings, please refer to Designer Options.</td>
</tr>
</tbody>
</table>

**Renaming Cells**

Select the desired cell in this window and press the F2 key to modify the Name property of the cell.
Align Cells

Cells are displayed in the order of the index in this window. The cell index can be changed by selecting the cell and dragging and dropping it at a different position in this window.

Template - Named Cell Styles 7.0

**Template - Named Cell Styles 7.0** is a window that provides a list of templates of named cell styles. You can use this window to apply the named cell style selected in the template to an existing cell using a single click. You can also add, delete, or edit the named cell styles. This window is similar to the **Apply Styles** pane in Visual Studio 2008 which can be used while creating ASP.NET Web Applications.

Displaying the Window

Use the following step to display the **Named Style Manager Window 7.0** in case it is not displayed.

- Select the **Other Windows - Template - Named Cell Styles 7.0** option from the Visual Studio **View** menu.

Creating a Named Cell Style

A new named cell style can be added by clicking the **New Cell Style** link displayed at the top of the window. New named cell styles created here are added to the Template.NamedCellStyles property.

Added styles can be edited using the following dialog. The settings in this dialog are the same as the cell style settings.
Changing Named Cell Styles

The styles that are added are displayed in the window as a list.

Styles can be copied or moved by clicking on the drop-down button displayed on the right side of each style.

To apply a style to a particular cell, select the cell, and then select the style. If a style has been applied to a cell or cells, the icon on the left hand side of the style changes as shown below.

Designer Options

There are two ways to save designer options. Designer options are saved in the development environment with the Windows user account. They cannot be saved for every template or project.
Visual Studio Options

Visual Studio options can be displayed by clicking **Tools - Options**. Designer options can be changed by selecting **MultiRow 7.0 Template Designer** from the tree on the left pane.

Template Explorer 7.0

The designer settings dialog can be displayed by clicking the **Settings** button from the **Template Explorer 7.0** toolbar.

Option Items
<table>
<thead>
<tr>
<th><strong>Property</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>GDIPlusPropertiesReminder</td>
<td>Specifies the processing method for UseCompatibleTextRendering. The default value is AlwaysShowDialogToAsk.</td>
</tr>
<tr>
<td>LayoutMode</td>
<td>Switches the layout modes. See <strong>Layout Modes of the Designer</strong> for more information. The mode changes to snapline mode when the property value is SnapLines, to grid mode when the value is Grid, and no mode is applied when the value is None. The default value is SnapLines.</td>
</tr>
<tr>
<td>SelectionRangeMode</td>
<td>Specifies the method to select a cell, using the drag operation of the mouse. The default value is IntersectedCells.</td>
</tr>
<tr>
<td>AllowBorderLineZoom</td>
<td>Indicates whether to zoom the border lines on the designer. The default value is False.</td>
</tr>
<tr>
<td>Annotations</td>
<td>Specifies the annotation settings. Specifies the indicator, author, creation date, and time for annotations.</td>
</tr>
<tr>
<td>CellDisplayModeBackColor</td>
<td>Specifies the backcolor of the cell's display information. The default value is SystemColors.Highlight.</td>
</tr>
<tr>
<td>CellDisplayModeForeColor</td>
<td>Specifies the forecolor of the cell's display information. The default value is SystemColors.HighlightText.</td>
</tr>
<tr>
<td>DotColor</td>
<td>Specifies the color of the grid points when the layout mode of the designer is set to Grid mode. The default value is Color.Gray.</td>
</tr>
<tr>
<td>DotSize</td>
<td>Specifies the interval between grid points when the layout mode of the designer is set to Grid mode. The default value is Size(10, 10).</td>
</tr>
<tr>
<td>FocusFlowActiveLineColor</td>
<td>Indicates the color of the active focus flow line.</td>
</tr>
<tr>
<td>FocusFlowLineColor</td>
<td>Indicates the color of the focus flow line.</td>
</tr>
<tr>
<td>GridLineColor</td>
<td>Specifies the color of the grid to be displayed in the designer background. The default value is Color.WhiteSmoke.</td>
</tr>
<tr>
<td>RulerUnit</td>
<td>Specifies the unit for the ruler of the template designer. The unit is pixel when the property value is set to Pixels, centimeter when value is Centimeters, and inch when value is Inches. The default value is pixels.</td>
</tr>
<tr>
<td>ShowCellBoundsTip</td>
<td>Specifies whether to display the cell position and size in the tooltip when the cell is either moved or resized in the designer. The default value is False.</td>
</tr>
<tr>
<td>ShowDesignerVersion</td>
<td>Specifies whether to display version and license information on the designer. The default value is True.</td>
</tr>
<tr>
<td>ShowGridLine</td>
<td>Specifies whether to display gridlines on the designer background. The default value is True.</td>
</tr>
<tr>
<td>ShowSectionIcons</td>
<td>Specifies whether to display the icons for designer sections. The default value is True.</td>
</tr>
<tr>
<td>SnapLineColor</td>
<td>Specifies the color of the snaplines in the designer. The default value is Color.RoyalBlue.</td>
</tr>
<tr>
<td>TableColumnCaptionFormat</td>
<td>Indicates the format used for the column caption in the table. The default value is %A%.</td>
</tr>
<tr>
<td>TableRowCaptionFormat</td>
<td>Indicates the format for the row caption in the table. The default value is %1%.</td>
</tr>
</tbody>
</table>
| VisualStyle | Specifies the visual style of the designer. When this property is set to VisualStudio, the
design has a base color of gray. When set to Office, blue is the base color. The default value is VisualStudio.

Refer to **Register User Defined Cell in ToolBox** for how to register a user defined cell in the toolbox.

**Importing and Exporting Options**

Designer options can be restored or a backup can be made using the Visual Studio menu **Tools - Import and Export settings**. The restored settings are not applied if the designer **Document Window** is kept open while restoring. The settings are applied once the **Document Window** is closed and then reopened.

**Using the Designer**

You can easily create a template, which is the base for the grid control, using the designer. This section explains the basic features of a designer. Refer to the **Advanced Designer Features** section for designing a template using advanced features.

- **Changing Height and Width of a Row in Designer**
Changing Height and Width of a Row in Designer

You can change the height and width of a row using either of the following methods.

**Dragging a Section**

You can change the width of a row by simply dragging the right border of the row section. By doing this, the width of the column header section as well as the row header section is changed simultaneously. The row height can be modified by dragging the bottom border of the row section.

**Double-Click on the Section Edge**

You can automatically adjust the width of the row to the width of the cell on the far right by double-clicking on the right edge of the row section. You can automatically adjust the height of the row to the height of the bottom-most cell by double-clicking on the bottom edge of the row section. If the row section has no cells, nothing happens.

**Properties Window**

You can change the width of a row in the Properties window using these steps.

1. Select a template (for example - Template1) from the drop-down list of the Properties window.
2. Change the value of the **Width** property.

The row height can be changed in the following way.

1. Select a row (for example - Row) from the drop-down list of the Properties window.
2. Change the value of the **Height** property.

**Working with Column Headers in the Designer**

The column header is a region that contains the column header cell. The developer can add multiple column header sections and toggle between them at runtime.

**Adding Column Header Section**

A column header section can be added using either of the following methods.
Select Template - Add - ColumnHeader from the menu.
Select Add - ColumnHeader from the context menu.

Changing Column Header Section Settings

Settings for the column header section can be changed from the Properties window.

Deleting Column Header Section

Follow the steps below to delete the column header section.

1. Place the mouse cursor on the column header section title and open the context menu.
2. Select Delete from the context menu.

Working with Column Footers in the Designer

The column footer is a region that contains static information. The developer can add multiple column footer sections and toggle between them at runtime.

- A header cell placed in a column footer section cannot be selected.

Adding Column Footer Section

A column footer section can be added using either of the following methods.

- Select Template - Add - ColumnFooter from the menu.
- Select Add - ColumnFooter from the context menu.

Changing Column Footer Section Settings

Settings for the column footer section can be changed from the Properties window.

Deleting Column Footer Section

Follow the steps below to delete the column footer section.

1. Place the mouse cursor on the column footer section title and open the context menu.
2. Select Delete from the context menu.

Adding Cells in the Designer

You can create a grid layout in MultiRow by adding cells in the template. Cells added in a template can be used for displaying information or for user input just like any other control on a form. Steps for adding a cell in the Designer are the same as adding a control to a Form.

- The cell name (Cell.Name property value) is automatically assigned once the cell is added to the designer. This value can be changed by the developer.
- ColumnHeaderCell (‘ColumnHeaderCell Class’ in the on-line documentation), CornerHeaderCell (‘CornerHeaderCell Class’ in the on-line documentation), and FilteringTextBoxCell (‘FilteringTextBoxCell Class’ in the on-line documentation) can be added to column headers only. The RowHeaderCell (‘RowHeaderCell Class’ in the on-line documentation) can only be added to rows.

Using the Designer
Follow the steps below to add a cell by specifying its position and size.

1. Open the designer.
2. Click on the cell you want to add to the template from the Visual Studio toolbox.
3. Click on a location to specify the top left corner of the cell and drag the mouse to the position you want the bottom right corner of the cell to be within the template.
4. A control with the specified size is added at the desired position.

Follow the steps below to add a cell with a default size at a specified position.

1. Open the designer.
2. Click on the cell you want to add from the Visual Studio toolbox and drag it to the template.
3. A control with the default size is added at the desired position.

Follow the steps below to add a cell with a default size at a default position.

1. Open the designer.
2. Double-click on the cell you want to add to the template in the Visual Studio toolbox.
3. A control with the default size is added at the default location.

Drag and drop a field to the template from the Data Sources window in Visual Studio, to add a String type cell with a default size and position and a data field assigned to it.

Replacing Cells in the Designer

You can replace any cell in a template with another cell type in MultiRow. When replacing the cell type, the position, size, and style settings do not change. Any automatically generated cell names, for example, "textBoxCell1", also get replaced with the name of the new cell type. If the name has been customized, for example, "NametextBoxCell1", the existing cell name is used as is.

Replacing a Cell

You can replace a cell by clicking on that cell in the designer and then using the context menu (right-click menu) to select the cell type you want to replace it with.

Replacing a Cell with User Defined Cell Type

Select the User Defined Cell Type option to replace a cell with a user-defined cell.
Moving Cells in the Designer

Cells can be moved between sections, within a template, to another template within a project, or to a template being used in another solution. Moving a cell in the designer is similar to moving controls on a form in the Form Designer.

- Cell name (Cell.Name property value) changes automatically if the same cell already exists at the location it is being moved to.
- **ColumnHeaderCell** ('ColumnHeaderCell Class' in the on-line documentation), **CornerHeaderCell** ('CornerHeaderCell Class' in the on-line documentation), and **FilteringTextBoxCell** ('FilteringTextBoxCell Class' in the on-line documentation) can be moved to column header or column footer only. **RowHeaderCell** ('RowHeaderCell Class' in the on-line documentation) can be moved to rows only.

Using the Menu to Move Cells

Follow the step below to move the cells using the clipboard.

- Select the cell you want to move and then go to the **Edit** menu - **Cut** in Visual Studio.

Follow the step below to paste cells from the clipboard.

- Select the location where you want to move the cells and then select the **Edit** menu - **Paste** in Visual Studio.

You can also select multiple cells and move them simultaneously. Refer to **Selecting Cells in the Designer** for details on selecting cells in the designer.

Using Keyboard Operations to Move Cells

Follow the steps below to move the selected cells using the clipboard.

- **Ctrl + X**

Follow the step below to paste cells from the clipboard to the template.

- **Ctrl + V**

Using the Mouse to Move Cells

Select the cells to be moved and drag and drop them at the desired location.

Copying Cells in the Designer

Cells can be copied (duplicated) in the designer using the clipboard. Copying a cell in the designer is similar to copying controls on a form.

- If the same cell already exists at the destination where the cell is being copied to, the cell name (Cell.Name property value) changes automatically before copying.
- The column header cell can be copied to the column header or column footer only. The row header cell can be copied to rows only.
Using the Menu to Copy Cells

Follow the steps below to copy cells to the clipboard.

- Select the cell or cells you want to copy and then select **Copy** in the **Edit** menu in Visual Studio.

Follow the steps below to copy cells from the clipboard.

- Select the location where you want to copy the cells and then select **Paste** from the **Edit** menu in Visual Studio.

The contents of the cells copied to the clipboard can be removed explicitly or can be used until they are overwritten by another copy operation. You can also select multiple cells and copy them simultaneously. Refer to **Selecting Cells in the Designer** for details on selection of cells in the designer.

Using Keyboard to Copy Cells

Use the keys below to copy the selected cells to the clipboard.

- **Ctrl + C**

Use the keys below to copy cells from the clipboard to the template.

- **Ctrl + V**

Using Mouse to Copy Cells

Select the cells to be copied and drag and drop them at the desired location while the **Ctrl** key is pressed.

Selecting Cells in the Designer

Certain operations such as moving, copying, and resizing cells in the designer can be done collectively for all the cells, if you have selected the group of cells before performing these operations. Steps for selecting cells in the designer are the same as selecting controls in the form designer.

Cell selection will not work for more than one section if the cells are in multiple sections. The selection for cells that are outside the active section will be removed automatically.

Selection using Menu

Menu usage allows you only to select all the cells at one time.

- Select **Edit - Select All** in Visual Studio.

Selection using Keyboard

When a cell is clicked while pressing the **Ctrl** key, the current cell is selected in addition to the previously selected cell. When a selected cell is clicked while pressing the **Ctrl** key, the selection of the cell is removed. When a cell is clicked without pressing the **Ctrl** key, the selection of all the previously selected cells is removed and only the clicked cell is selected. The selected state of all the cells is removed when clicking on the background section of the selected cells.

Selection using Mouse

Follow the steps below to select multiple cells.

1. Click the background of the section.
2. Drag the mouse to include the cells you want to select, into the area indicated by the dotted line.
3. All the cells in the dotted line area are selected.
Cell selection can also be done by dragging the designer ruler.

You can change the range selection mode when selecting cells by dragging the mouse or ruler, using the `SelectionRangeMode` Designer Options.

**Resizing Cells in the Designer**

You can change the size of cells using either the mouse or the keyboard. Steps for resizing cells in the designer are the same as for resizing controls on a form.

**Resizing using Menu**

If multiple cells have been selected, they can be resized using the Visual Studio `Format` menu.

- Select multiple cells that need to be resized and select the desired option under `Format - Adjust to Same Size`.

**Resizing using Keyboard**

Follow the step below to resize the selected cell.

- **Shift + Arrow Key**
  Use the left or right arrow key for increasing the width of the cell and use the top or bottom arrow key for increasing the height. Use the arrow keys without pressing the `Shift` key to change the position of the cells.

The above procedure works in grid units when `Layout Modes of the Designer` is set to Grid. Follow the steps below if you wish to resize the cells in pixel units.

- **Ctrl + Shift + Arrow keys**

**Resizing using Mouse**

Select the cell you want to resize and drag the square handle that is displayed around the cell to change the size according to your requirements.

It is also possible to select multiple cells and change the size simultaneously. In this case, size is increased equally for all the cells.

**Get or Set Cell Properties in the Designer**

Settings for cells can be changed in the designer with the `Properties` Window or through commands.
Properties Window

The Properties window is provided by Visual Studio. You can get or modify the property values of cells selected in the designer in the Properties window. Follow the steps below to display the Properties window.

- Select Properties Window from the View menu in Visual Studio.

Property List Window

The property list window contains a list of all the cells and their properties in the template. This window is useful in verifying settings for common properties. You can:

- Display Name ('Name Property' in the on-line documentation), Value ('Value Property' in the on-line documentation), DataField ('DataField Property' in the on-line documentation), Enabled ('Enabled Property' in the on-line documentation), ReadOnly ('ReadOnly Property' in the on-line documentation), Visible ('Visible Property' in the on-line documentation), TabIndex ('TabIndex Property' in the on-line documentation), and TabStop ('TabStop Property' in the on-line documentation) properties of the cell.
- Select a data source and enter the DataField ('DataField Property' in the on-line documentation) property value of the cell from the list.
- Select properties displayed in the list.

Follow the steps below to display the property list window.

- Select Other Windows - Template PropertyList 7.0 from the View menu in Visual Studio.

Cell Display Mode

Use the cell display mode to show the following information in the designer cells.

- Cell.Index property value
- Cell.Name ('Name Property' in the on-line documentation) property value
- Cell.DataField ('DataField Property' in the on-line documentation) property value
- Cell Type

Refer to Cell Display Modes in the Designer for details.

Changing the Tab Order

Follow the steps below to change the tab order.

1. Select Tab Order from the View menu in Visual Studio.
2. Click the tab order displayed in the cell to rearrange it.

The tab order can also be changed with the Cell.TabIndex property in the Properties Window.

Toolbar

You can change the format or position of the cell using the Toolbar.

Changing Cell Index in the Designer

The index automatically gets assigned in the order in which the cells are added to the designer. You can change the order of the index later. Cell indices are used to identify the cells in the GcMultiRow control at runtime.
Displaying Cell Index

Follow either of the steps below to display a cell index in the designer.

- Select **Cell Display Mode - Index** from the Visual Studio **Template** menu.
- Select **Columns - More** from the **Template PropertyList 7.0** toolbar and check **CellIndex** in the **Choose Details** screen, then click the **OK** button.

Changing Cell Index

You can change the order of the cell index using any of the following methods.

- Drag and drop the cell in **Template Explorer 7.0**. The index changes when you drag and drop the cell from one position to another.
- Use **Send to Back** and **Bring to Front** from the context menu. You can use the context menu commands (displayed when selecting cells in the designer) to move the cell index to the top or bottom.
- Deleting and placing the cells again in the template. The initial value of the cell index is determined by the order in which it has been placed. When deleting the cell, the next cell index moves up to that position.
- Changing the sequence of cells in the source code of the template. You can change the order in which the cells were added by opening Template1.Designer.vb or Template1.Designer.cs files in the code editor.
- You can change the cell index of the cells by selecting **Reorder** from the context menu (right-click menu) of the template, and specifying **OverThenDown** for **CellIndex** or **DownThenOver** for **CellIndex**.

Undo Changes in Designer

You can reverse the changes that have been made in the template through the designer and revert back to the previous state (Undo). Steps to undo changes are the same as for the form designer or code editor.

Undo using Menu

Follow the step below to revert back to the previous state by canceling the changes made just before.

- Select **Edit - Undo** from the Visual Studio menu.

Follow the step below to reverse the **Undo** operation.

- Select **Edit - Redo** from the Visual Studio menu.

These operations cannot be executed when menu commands are displayed in grey.
Copying using Keyboard
Follow the step below to cancel the previous change and revert to the previous state.

- Ctrl + Z

Follow the step below to reverse the Undo operation.

- Ctrl + Y

Locking Cells in the Designer
The locking feature has been provided to make sure that no changes are applied to already placed cells while designing the template. The locking feature can only be used for mouse or keyboard operations during design time display; however, settings for locked cells can be modified through the Properties window. Sections cannot be locked.

![Locking Cells Diagram]

Locking Cells
Follow either of these methods to lock a cell.

- Select Format - Lock Cells in Visual Studio.
- Select Lock Cells from the context menu in the document window.

Once the cell has been locked, a Lock icon is displayed on the top left instead of a handle for changing the size.

Unlock Cells
A cell lock can be removed by performing the locking again for the locked cell.

Changing Tab Order of Cells in the Designer
The tab order for cells can be changed in the designer in the same way as for the controls on a form.

Document Window
Follow the steps below to change the cell tab order in the Document Window.

1. Select View - Tab Order in Visual Studio.
2. Click on cells in the sequence you want to assign the tab order.

![Document Window Diagram]

Properties Window
Follow the steps below to change the tab order of the cells through the Properties window.

1. Select the target cell from the drop-down list displayed in the Properties window.
2. Change the value of the TabIndex property.
Template Property List Window

You can also change the tab order by changing the value of the tab index column in the Template Property List window.

Context Menu

You can change the tab order of the cells by selecting Reorder from the context menu (right-click menu) of the template, and specifying OverThenDown for TabIndex or DownThenOver for TabIndex.

Focus flow

If you use the FocusFlow command of the MultiRow Template Designer 7.0 Toolbar, the arrow of the focus flow is displayed in accordance with the tab order.

You can change the tab order by dragging the focus flow using the mouse.

• The focus flow only works in the Row section. It does not work in the column header section and the column footer section.
• If there are cells with the same TabIndex, the cell with the smaller CellIndex value is prioritized.
• Cells for which the TabStop property is set to False, are ignored.

Display Modes of the Designer

The designer provides three types of display modes. You can switch the display mode using the tabs displayed at the bottom of the display area.

Design Mode

You can change the settings for the template or the cell in this mode.
**Runtime Mode**

In Runtime mode, you can verify how the GcMultiRow control would behave if you loaded the design template into it. The GcMultiRow control uses default settings for properties. In this mode, values entered into cells are retained until the document window is closed.

**Print Preview Mode**

In Print Preview mode, you can verify how the GcMultiRow control will print when you load the design template into it. The GcMultiRow control uses default settings for properties. Printing related properties can be set from the toolbar.

![Print Setting](image)

**Cell Display Modes in the Designer**

The designer provides multiple cell display modes. You can change the cell display mode from the Template - Cell Display Mode menu option or the MultiRow 7.0 Designer toolbar.

You can change the background and foreground color of the cell display mode as described in Designer Options.

**CellIndex**

The value of the Cell.Index property is displayed in the cell in this mode. Refer to Changing Cell Index in the Designer to change the cell index.

![Cell Index](image)

**CellName**
The value of the Cell.Name property is displayed in the cell in this mode.

![Image](image1.jpg)

**DataField**

The value of the Cell.DataField property is displayed in the cell in this mode. When the Cell.DataField value has not been set, "(None)" is displayed in the cell.

![Image](image2.jpg)

**CellType**

The cell type is displayed in the cell in this mode. For example, "TextBoxCell" is displayed in a String type cell.

![Image](image3.jpg)

**Icon**

The cell type icon is displayed in the cell in this mode.

![Image](image4.jpg)

**None**

No property value is displayed in the cell in this mode. This is the default mode.

![Image](image5.jpg)

**Layout Modes of the Designer**

The designer provides three types of layout modes. You can change the layout mode by setting the LayoutMode property in **Designer Options**.

**Snaplines Mode**

The snaplines mode makes it easier to match cell alignment and size with other cells. Once you enable this mode and then change the size and position of the cells, ruled lines are displayed to assist you with cell alignment. They are referred to as snaplines. It is easy to align the cells when snaplines are displayed.

**Grid Mode**

You can adjust the position and size of cells based on the points being displayed in the grid using grid mode. By default grid points are separated by 10 pixels; however, you can change this unit with the DotSize property in **Designer Options**.
None Mode

If none of the modes are selected, the feature to assist in cell alignment is disabled.

Temporarily Disable Layout Mode

You can disable any of these modes by pressing the Alt key. In this case, objects can be resized or moved in pixel units.

Setting the Column Mode Template in the Designer

You can design a template in the Designer using the column mode template.

Setting the Column Mode

If you set the Template.LayoutMode (‘LayoutMode Property’ in the on-line documentation) property to LeftToRight, the horizontal display of the ruler and layout section is replaced with a vertical display.

Setting the Column Mode Template using the Wizard

You can create a template for the column mode template, using the Wizard. For details, refer to Create Template using Wizard.

Advanced Designer Features

The designer provides various features that enable efficient designing of the template. This section explains the advanced features of the designer. For designer basics, refer to Using the Designer.

- Create Template using Wizard
- Create Template from Data Sources Window
- Create Template from Table
- Create Templates Using Tracing Mode
- Adjust Character Size with Placeholders
- Template Localization
- Hints for using the Designer
- Conversion from DataGridView
Create Template using Wizard

You can create a blueprint of the template in an interactive manner using the wizard. The template created using the wizard can be edited using the Template Designer just like normal templates.

Here, the process has been described with Visual Studio 2012 as an example, but the process is the same with Visual Studio 2008, Visual Studio 2010, and Visual Studio 2013.

Create and Add New Templates

This example explains how to create and add a new template using the wizard.

1. Create a new Windows application project in Visual Studio or open an existing one.
2. From the Visual Studio menu, select **Project - Add New Item**.
3. Select **MultiRow 7.0 Template Wizard** from the list and click the **Add** button.

4. Create the template using the wizard.
5. Select the template mode.
6. Check the template added to the project in the **Solution Explorer**.

**Assign Template to the Control**

Use the following steps to assign a template created with the wizard to a GcMultiRow control placed on the form.

1. Select the GcMultiRow control on the form, and then select **Select Template - Add New Template** from the Smart Tag.

2. Create a template using the wizard.

3. Select the GcMultiRow control and open the designer by clicking **Edit Template** in the Smart Tag.

**Create Template from Data Sources Window**

You can add a specific data source to the Visual Studio Data Sources window, and then drag and drop the displayed tables and fields to the designer, to create a basic layout of the template.

**Drag and Drop Tables**

When you drag and drop tables to the template, all fields included in the table are placed as cell types. Pressing the Ctrl key while dragging and dropping generates the column headers and row cells at the same time.

**Drag and Drop Fields**
If you drag and drop each table field onto the template, the cells and the connection to that field are placed. Pressing the Ctrl key while dragging and dropping generates the column headers and row cells at the same time. Press the Shift key while dragging and dropping the fields to the target cells to assign the fields to already existing cells.

**Change Cell Type**

You can change the cell type that corresponds to the field. Select the field in the Data Sources window, and then use the drop-down menu to select the cell type that you want to link to.

You can change the cell types for each data type, when you select **Customize** from the drop-down menu.
Create Template from Table

You can use tables to create multiple tabular screens with rows and columns just like the table creation feature of Microsoft Word or the report designer of Visual Studio. You can select whether you would like to preserve the created tabular screen as a tabular screen, or if you would like to change it to individual cells.

If you change it to individual cells, it is not possible to revert back to the table unless there is an **Undo** option. Since tables are a design time feature, runtime behavior is not affected if the cells are created from a table. Also, you cannot work with tables at runtime.

**Table Structure**
Add Tables

To add tables, drag and drop the Table from the toolbox window in the designer.

To place a table that includes cells with a default size, drag a specific range in the designer section to match a selected area.
Move Tables

Tables can be moved in the same way as cells, using the mouse or keyboard. Refer to Moving Cells in the Designer.

Delete Tables

To delete tables, select the table and then press the Delete key or click Delete in the context menu.

Ungroup Tables

To split the cells of the table into individual cells, you need to click Ungroup Table from the context menu. Once the table is converted to individual cells, it is not possible to go back to the original table, unless the Undo option is enabled.

Delete Cells

To delete cells in the table, you need to select the cells inside the table, and then select Delete from the context menu. After the cells have been deleted, a pattern showing that the cells do not exist is shown.
Merge and Split Cells

You can merge multiple cells in the table and split the merged cells. When you merge multiple cells, they are changed to a single cell of the same size.

Insert Rows

To insert rows into the table, select the cells or the row headers inside the table and select Insert from the context menu. If you select multiple adjacent rows, and perform row insertion, then the selected number of rows is inserted.

Insert in the context menu does not work if you select multiple rows that are not adjacent.

Delete Rows

In order to delete the rows from the table, select the row headers from the table, and either press the Delete key or click Delete in the context menu.

Change the Row Width

In order to change the width of the table rows, drag the edges of the table row headers.

Insert Columns

To insert columns into the table, select the cells inside the table, or the column headers, and click Insert in the context menu. If you select multiple adjacent columns, and perform column insertion, then the selected number of columns is inserted.

Insert in the context menu does not work if you select multiple columns that are not adjacent.

Delete Columns

To delete columns from the table, select the column header in the table, and either press the Delete key or click Delete in the context menu.

Change the Column Height

To change the table column height, drag the column header edge with the mouse.
Table and Cell Borders

When multiple cell boundaries overlap, the cell borders that have a higher index are given priority. This may cause cases where the results of applied borders might not be as expected, due to the ordering of cell indices. In this case, select the target cells and change the indices.

See the following example:

1. Create the table and select four cells at the top left.

2. Select Borders in the context menu, and under that, select Normal Border. Select Outline and Inside.
3. Click **OK** to close the dialog.

You can see that there is a difference between the applied borders and the preview on the Borders dialog. This is because when the cell boundaries overlap the neighboring cells, the cell borders with higher indices are given preference. In the following image the numbers of the bottom right cells are higher than the selected cells, and since the cells are displayed with grey borders by default, the result looks like this.

Select four cells at the top left, and from the context menu, select **Bring to Front**. This changes the index numbers of the cells, and you get the expected result.

**Table Captions**

By default, the caption of the table is displayed as "A, B, C, ..." for the column header, and as "1, 2, 3, ..." for the row header. If you want to change the caption of the table, set the **TableRowCaptionFormat** and **TableColumnCaptionFormat** in the designer options.

> It is not possible to change the height of the column header, or the width of the row header, even if a caption that exceeds the display area of the cell is set.

**Create Templates Using Tracing Mode**

If you use the tracing mode, you can load an underlay image in the background of the template on which you are working. You can then place cells using this underlying image.

> The underlay image set in the tracing mode is not visible at runtime. In order to display an image in the background of the row at runtime, use the **Row.BackgroundImage** property.
**Set the Underlay Image**

In order to add an underlay image to the template, use the following steps.

1. Click on the grey area in the template, or select the class that inherits the template from the **Properties** window (for example: Template1).
2. Select the property Template.TracingImage in the **Properties** window, and select the desired image.
3. Confirm that the loaded image is being displayed in the background of the column header or row, in the template.
4. Adjust the position of the underlay image by dragging the background of the column header section or the rows.

**Change the Opacity of Underlay Image**

By default, the opacity of the underlay image is 50%. Use the Template.TracingImageOpacity property to change the opacity to the desired value.

**Move the Underlay Image**

You can adjust the position of the underlay image by dragging the background of the column header section or rows. In order to prevent any change in the position, set the value of the Template.TracingImageLocked property to True.

**Hide the Underlay Image**

In order to temporarily hide the underlay image, set the value of the Template.TracingImageVisible property to False.

**Delete the Underlay Image**

In order to delete the underlay image, select the Template.TracingImage property from the **Properties** window, and select **Reset** from the context menu.

**Adjust Character Size with Placeholders**

The designer provides a DesignTimeValue property that you can use to set the characters that are displayed only at design time (Design-time Placeholder). By using this property at design time, you can display the value for estimating the cell size, and hide this value at run time, without the need for code. For example, using a fixed width font and a fixed number of characters, you can estimate the size of the string using dummy values like "999-9999" or "XXXXXXX".

DesignTimeValue is a special property that can only be used at design time and cannot be used at runtime. Use the Cell.Value ('Value Property' in the on-line documentation) property to display the same value at design time as well as runtime.

**Tutorial**

Since the DesignTimeValue property is mainly used to estimate the width of the string, it is recommended to use it with fixed width fonts.

The following steps explain how to specify the fixed width font and dummy string.

1. Open the designer.
2. Select **TextBoxCell** from the toolbox and drag it to the **Row** section.
3. Using the **Properties** window, set the **Font** property of textBoxCell1 to Arial, 11 points.
4. Set the **MaxLength** property of textBoxCell1 to 10 through the **Properties** window.
5. Set the **DesignTimeValue** property of textBoxCell1 to **XXXXXXXXXX** in the **Properties** window.
6. Change the width of textBoxCell1 depending on the string that is displayed in it.
7. Click the **Runtime** tab of the **Document** window, and check whether exactly 10 characters can be entered in the display area of textBoxCell1.

### Data Type of DesignTimeValue Property

The specification of the DesignTimeValue property is same as the Value property of that cell. For example, in the date type cell (DateTimePickerCell), since the Cell.Value (**Value Property** in the on-line documentation) property has a Date type value, the type of the DesignTimeValue property is also Date type.

### Storage Location of DesignTimeValue Property

The DesignTimeValue property is not a hidden property of the cell, but is design information. The value of the DesignTimeValue property is stored in the Template.resx file, and is not output to Template.Designer.vb (or Template.Designer.cs).

Related topics are **Locking Cells in the Designer** and **Create Template from Table**.

### Template Localization

The template consists of language resources as per the localization framework of the .NET Framework, similar to the form (System.Windows.Forms.Form), and thus can be easily interchanged to display various languages. For details on the globalization support of Visual Studio and the .NET Framework, refer to MSDN [Globalizing and Localizing Applications](https://docs.microsoft.com/en-us/dotnet/framework/wcf/localization).

#### Localizing a Template

The following steps show how to create forms and templates which have Japanese and English resources.

2. Open the form (Form1), place two button controls, and set the **Text** property to **Japanese** and **English** (for example: Button1, Button2).

![Form1](image)

3. From the Visual Studio menu, select **Project - Add New Item**, and add two forms (for example: Form2).
4. Place a Label control and a GcMultiRow control on Form2 (for example: Label1 and GcMultiRow1).
5. Put "This is an English message." in the **Text** property of Label1.
6. Set the **Localizable** property of Form2 to True.
7. Set the **Language** property of Form2 to **Japanese**.
8. Set the **Text** property of Label1 to Japanese text.

9. Open Form1 and write the following code in the Button control click event.

**[VB]**

```vbnet
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    Dim form As New Form2()
    form.ShowDialog()
End Sub

Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
    Dim form As New Form2()
    form.ShowDialog()
End Sub
```

**[CS]**

```csharp
private void button1_Click(object sender, EventArgs e)
{
    Form2 form = new Form2();
    form.ShowDialog();
}
```

Copyright © GrapeCity, inc. All rights reserved.
```csharp
private void button2_Click(object sender, EventArgs e)
{
    Form2 form = new Form2();
    form.ShowDialog();
}
```

10. Run the project and click on the English and Japanese buttons on Form1, to see the differences.
You can see the result of form localization with the above example.
The next example localizes the templates that are displayed in the GcMultiRow control.

1. From the Visual Studio menu, select Project - Add New Item.
2. Select MultiRow 7.0 Template from the list and click on the Add button.
3. Create a screen that looks like the following. Use corner header cells, column header type cells, row header type cells, and string type cells.

4. Add Zip Code and Address to the Value property of the two column headers (ColumnHeaderCell).

5. Set the Localizable property of the template to True.
6. Set the Language property of the template to Japanese.
7. Change the Value property of the two column headers (ColumnHeaderCell) to:

8. Open Form2. Select the template (Template1) from the smart tag of the GcMultiRow control.
9. Run the project and click the **Japanese** and **English** buttons respectively on Form1 to see the differences.

When you click the **Japanese** button on Form1, the Japanese screen is displayed as shown below.

Click the **English** button on Form1 to display the English screen as shown below.
Hints for using the Designer

You may find the following hints useful when using the designer.

**Select the Template**

You can click in one of the following areas to select a template faster than selecting it from the Properties window.

- Box on the top left of the designer

- Empty area of the designer (grey section)

**Appearance of the Designer**

You can set the appearance using the VisualStyle property in the settings option of the MultiRow designer. For details, refer to **Designer Options**.

**Change the Template Language (Source Code)**

The designer maintains the created template as source code. Initially, the source code language is the same as that of the project. If you change the source code of a project that has been created in Visual Basic, to a C# project, you can avoid manually rewriting the template source code. You can use the clipboard to make copies of the objects present on the template. The source code is then automatically generated based on the language used in the project.

Use the following steps:

1. Change the project to be converted (for example - Visual Basic).
2. Open the template in the project that needs to be converted, select all the cells, and copy to the clipboard.
3. Open the project you are converting to (for example - C#).
4. Create a new template and copy the cells from the clipboard.
   - If you have your specific source code written in the Template.vb (Template.cs), this section cannot be changed using the clipboard.
   - You cannot copy the template itself or the section. Reset or add the properties manually.
   - It is necessary to repeat the clipboard operation for each section.
   - You cannot copy and paste the contents on the clipboard, between the designer of this product and the designer of the previous version (MultiRow for Windows Forms 6.0).

**Delete the Unnecessary ColumnHeaderSection**

There is a ColumnHeaderSection added by default in the designer. To delete this, click to select the
ColumnHeaderSection, and click the **Delete** key or click **Delete** from the context menu.

**Differentiating between Snaplines Mode and Grid Mode**

The designer provides Snaplines Mode and Grid Mode. The default setting is Snaplines Mode and this mode provides snaplines for resizing and movement of cells similar to what can be done in the Visual Studio Windows Form designer. Grid Mode provides grid points for resizing and movement of cells similar to what can be done when LayoutMode is set to SnapToGrid in the Visual Studio Windows Form designer.

Snaplines mode is useful in situations where a tabular screen needs to be created and where MultiRow cells need to be kept together without any space between them. This is because Snaplines facilitate aligning the cell spaces and the column headers.

Grid mode can be used to allocate spacing, and is useful in creating complicated tabular screens. Since the spacing between grid points can be used as spaces, you can reduce the load when adjusting the layout.

For example, you could create a report screen using Snaplines mode and a list screen using Grid mode. In that case, when you resize or move the cells (while keeping the **Alt** key pressed), you could temporarily disable the Layout mode to make it easier to make fine adjustments.

**Initialize the Property Values**

You can initialize property values by selecting specific properties from the Properties window and clicking **Reset**. This is similar to how this is done in the form designer.

**Conversion from DataGridView**

The layout of the DataGridView control can be converted to a GcMultiRow template at design time.

**Converting the Template**

The following steps convert the layout of the DataGridView control to a GcMultiRow template.

1. Select the GcMultiRow control, and then select **Convert DataGridView** from the context menu (right-click menu).

   - You will not be able to select **Convert DataGridView** in the context menu, if the DataGridView has not been placed on the form.

2. The **Convert DataGridView** dialog is displayed.
3. In **Select a DataGridView**, select the control you want to convert from the DataGridView controls placed on the form. You can set the **Remove previous DataGridView** and **Open template after converting** option items, if necessary.
4. Click on the **OK** button to perform the conversion.

**Rules for Conversion**

The conversion from DataGridView is based on the following rules.

- Only the property settings are converted. Events and methods are not converted.
- After conversion, **SelectionActions.MoveDown** ('MoveDown Property' in the on-line documentation) property is set on the Enter key, as a default value.
- The header at the top left of the DataGridView control is converted to CornerHeaderCell.
- Row header of the DataGridView control is converted to RowHeaderCell.
- If a column does not exist in the DataGridView control, the template is empty after the conversion.
- The Undo or redo of the DataGridView conversion operation can be performed.

**Properties After Conversion**

When you convert the DataGridView, the properties are converted to the following:
### DataGridView

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Cell.Name</td>
</tr>
<tr>
<td>AutoSizeMode</td>
<td>-</td>
</tr>
<tr>
<td>ColumnType</td>
<td>-</td>
</tr>
<tr>
<td>DataPropertyName</td>
<td>Cell.DataField</td>
</tr>
<tr>
<td>DefaultCellStyle</td>
<td>Cell.Style</td>
</tr>
<tr>
<td>DividerWidth</td>
<td>-</td>
</tr>
<tr>
<td>FillWeight</td>
<td>-</td>
</tr>
<tr>
<td>Frozen</td>
<td>GcMultiRow.FreezeLeftCellIndex</td>
</tr>
<tr>
<td>MinimumWidth</td>
<td>Cell.MinimumSize.Width</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>-</td>
</tr>
<tr>
<td>DataGridviewColumn.Resizeable</td>
<td>-</td>
</tr>
<tr>
<td>DataGridviewColumnHeadersHeightSizeMode</td>
<td>ColumnHeaderCell.ResizeMode</td>
</tr>
</tbody>
</table>

### DataGridViewText

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxInputLength</td>
<td>TextBoxCell.MaxLength</td>
</tr>
</tbody>
</table>

### DataGridViewButtonColumn

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FlatStyle</td>
<td>ButtonCell.FlatStyle</td>
</tr>
<tr>
<td>Text</td>
<td>-</td>
</tr>
<tr>
<td>UseColumnTextForButtonValue</td>
<td>-</td>
</tr>
</tbody>
</table>

### DataGridViewCheckBoxColumn

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FalseValue</td>
<td>CheckBoxCell.FalseValue</td>
</tr>
<tr>
<td>IndeterminateValue</td>
<td>CheckBoxCell.IndeterminateValue</td>
</tr>
<tr>
<td>ThreeState</td>
<td>CheckBoxCell.ThreeState</td>
</tr>
<tr>
<td>TrueValue</td>
<td>CheckBoxCell.TrueValue</td>
</tr>
<tr>
<td>Sorted</td>
<td>CheckBoxCell.Sorted</td>
</tr>
</tbody>
</table>

### DataGridViewComboBoxColumn

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoComplete</td>
<td>ComboBoxCell.AutoCompleteMode and ComboBoxCell.AutoCompleteSource</td>
</tr>
<tr>
<td>DisplayMember</td>
<td>ComboBoxCell.DisplayMember</td>
</tr>
<tr>
<td>DisplayStyle</td>
<td>ComboBoxCell.ShowDropDownButton</td>
</tr>
<tr>
<td>DisplayStyleForCurrentCellOnly</td>
<td>-</td>
</tr>
<tr>
<td>DropDownWidth</td>
<td>ComboBoxCell.DropDownWidth</td>
</tr>
<tr>
<td>FlatStyle</td>
<td>ComboBoxCell.FlatStyle</td>
</tr>
<tr>
<td>Items</td>
<td>ComboBoxCell.Items</td>
</tr>
<tr>
<td>MaxDropDownItems</td>
<td>ComboBoxCell.MaxDropDownItems</td>
</tr>
<tr>
<td>Sorted</td>
<td>ComboBoxCell.Sorted</td>
</tr>
</tbody>
</table>

### DataGridviewColumn

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoSizeMode</td>
<td>-</td>
</tr>
<tr>
<td>ColumnType</td>
<td>-</td>
</tr>
<tr>
<td>DataPropertyName</td>
<td>Cell.DataField</td>
</tr>
<tr>
<td>DefaultCellStyle</td>
<td>Cell.Style</td>
</tr>
<tr>
<td>DividerWidth</td>
<td>-</td>
</tr>
<tr>
<td>FillWeight</td>
<td>-</td>
</tr>
<tr>
<td>Frozen</td>
<td>GcMultiRow.FreezeLeftCellIndex</td>
</tr>
<tr>
<td>MinimumWidth</td>
<td>Cell.MinimumSize.Width</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>-</td>
</tr>
<tr>
<td>DataGridviewColumn.Resizeable</td>
<td>-</td>
</tr>
<tr>
<td>DataGridviewColumnHeadersHeightSizeMode</td>
<td>ColumnHeaderCell.ResizeMode</td>
</tr>
</tbody>
</table>

### DataGridviewColumn.SortMode

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NotSortable</td>
<td>NotSortable</td>
</tr>
<tr>
<td>Programmatic</td>
<td>Programmatic</td>
</tr>
<tr>
<td>Automatic</td>
<td>Automatic</td>
</tr>
</tbody>
</table>

### DataGridviewCell

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ToolTipText</td>
<td>Cell.ToolTipText</td>
</tr>
<tr>
<td>Visible</td>
<td>-</td>
</tr>
<tr>
<td>Width</td>
<td>Cell.Width</td>
</tr>
</tbody>
</table>

### DataGridviewComboBox

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoComplete</td>
<td>ComboBoxCell.AutoCompleteMode and ComboBoxCell.AutoCompleteSource</td>
</tr>
<tr>
<td>DisplayMember</td>
<td>ComboBoxCell.DisplayMember</td>
</tr>
<tr>
<td>DisplayStyle</td>
<td>ComboBoxCell.ShowDropDownButton</td>
</tr>
<tr>
<td>DisplayStyleForCurrentCellOnly</td>
<td>-</td>
</tr>
<tr>
<td>DropDownWidth</td>
<td>ComboBoxCell.DropDownWidth</td>
</tr>
<tr>
<td>FlatStyle</td>
<td>ComboBoxCell.FlatStyle</td>
</tr>
<tr>
<td>Items</td>
<td>ComboBoxCell.Items</td>
</tr>
<tr>
<td>MaxDropDownItems</td>
<td>ComboBoxCell.MaxDropDownItems</td>
</tr>
<tr>
<td>Sorted</td>
<td>ComboBoxCell.Sorted</td>
</tr>
</tbody>
</table>

### DataGridviewComboBoxColumn

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoComplete</td>
<td>ComboBoxCell.AutoCompleteMode and ComboBoxCell.AutoCompleteSource</td>
</tr>
<tr>
<td>DisplayMember</td>
<td>ComboBoxCell.DisplayMember</td>
</tr>
<tr>
<td>DisplayStyle</td>
<td>ComboBoxCell.ShowDropDownButton</td>
</tr>
<tr>
<td>DisplayStyleForCurrentCellOnly</td>
<td>-</td>
</tr>
<tr>
<td>DropDownWidth</td>
<td>ComboBoxCell.DropDownWidth</td>
</tr>
<tr>
<td>FlatStyle</td>
<td>ComboBoxCell.FlatStyle</td>
</tr>
<tr>
<td>Items</td>
<td>ComboBoxCell.Items</td>
</tr>
<tr>
<td>MaxDropDownItems</td>
<td>ComboBoxCell.MaxDropDownItems</td>
</tr>
<tr>
<td>Sorted</td>
<td>ComboBoxCell.Sorted</td>
</tr>
</tbody>
</table>

### DataGridviewColumn

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoSizeMode</td>
<td>-</td>
</tr>
<tr>
<td>ColumnType</td>
<td>-</td>
</tr>
<tr>
<td>DataPropertyName</td>
<td>Cell.DataField</td>
</tr>
<tr>
<td>DefaultCellStyle</td>
<td>Cell.Style</td>
</tr>
<tr>
<td>DividerWidth</td>
<td>-</td>
</tr>
<tr>
<td>FillWeight</td>
<td>-</td>
</tr>
<tr>
<td>Frozen</td>
<td>GcMultiRow.FreezeLeftCellIndex</td>
</tr>
<tr>
<td>MinimumWidth</td>
<td>Cell.MinimumSize.Width</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>-</td>
</tr>
<tr>
<td>DataGridviewColumn.Resizeable</td>
<td>-</td>
</tr>
<tr>
<td>DataGridviewColumnHeadersHeightSizeMode</td>
<td>ColumnHeaderCell.ResizeMode</td>
</tr>
</tbody>
</table>

### DataGridviewColumn.SortMode

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NotSortable</td>
<td>NotSortable</td>
</tr>
<tr>
<td>Programmatic</td>
<td>Programmatic</td>
</tr>
<tr>
<td>Automatic</td>
<td>Automatic</td>
</tr>
</tbody>
</table>

### DataGridviewCell

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ToolTipText</td>
<td>Cell.ToolTipText</td>
</tr>
<tr>
<td>Visible</td>
<td>-</td>
</tr>
<tr>
<td>Width</td>
<td>Cell.Width</td>
</tr>
</tbody>
</table>

### DataGridviewComboBox

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoComplete</td>
<td>ComboBoxCell.AutoCompleteMode and ComboBoxCell.AutoCompleteSource</td>
</tr>
<tr>
<td>DisplayMember</td>
<td>ComboBoxCell.DisplayMember</td>
</tr>
<tr>
<td>DisplayStyle</td>
<td>ComboBoxCell.ShowDropDownButton</td>
</tr>
<tr>
<td>DisplayStyleForCurrentCellOnly</td>
<td>-</td>
</tr>
<tr>
<td>DropDownWidth</td>
<td>ComboBoxCell.DropDownWidth</td>
</tr>
<tr>
<td>FlatStyle</td>
<td>ComboBoxCell.FlatStyle</td>
</tr>
<tr>
<td>Items</td>
<td>ComboBoxCell.Items</td>
</tr>
<tr>
<td>MaxDropDownItems</td>
<td>ComboBoxCell.MaxDropDownItems</td>
</tr>
<tr>
<td>Sorted</td>
<td>ComboBoxCell.Sorted</td>
</tr>
</tbody>
</table>

### DataGridviewComboBoxColumn

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoComplete</td>
<td>ComboBoxCell.AutoCompleteMode and ComboBoxCell.AutoCompleteSource</td>
</tr>
<tr>
<td>DisplayMember</td>
<td>ComboBoxCell.DisplayMember</td>
</tr>
<tr>
<td>DisplayStyle</td>
<td>ComboBoxCell.ShowDropDownButton</td>
</tr>
<tr>
<td>DisplayStyleForCurrentCellOnly</td>
<td>-</td>
</tr>
<tr>
<td>DropDownWidth</td>
<td>ComboBoxCell.DropDownWidth</td>
</tr>
<tr>
<td>FlatStyle</td>
<td>ComboBoxCell.FlatStyle</td>
</tr>
<tr>
<td>Items</td>
<td>ComboBoxCell.Items</td>
</tr>
<tr>
<td>MaxDropDownItems</td>
<td>ComboBoxCell.MaxDropDownItems</td>
</tr>
<tr>
<td>Sorted</td>
<td>ComboBoxCell.Sorted</td>
</tr>
</tbody>
</table>

### DataGridviewCell

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ToolTipText</td>
<td>Cell.ToolTipText</td>
</tr>
<tr>
<td>Visible</td>
<td>-</td>
</tr>
<tr>
<td>Width</td>
<td>Cell.Width</td>
</tr>
</tbody>
</table>

### DataGridviewComboBox

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoComplete</td>
<td>ComboBoxCell.AutoCompleteMode and ComboBoxCell.AutoCompleteSource</td>
</tr>
<tr>
<td>DisplayMember</td>
<td>ComboBoxCell.DisplayMember</td>
</tr>
<tr>
<td>DisplayStyle</td>
<td>ComboBoxCell.ShowDropDownButton</td>
</tr>
<tr>
<td>DisplayStyleForCurrentCellOnly</td>
<td>-</td>
</tr>
<tr>
<td>DropDownWidth</td>
<td>ComboBoxCell.DropDownWidth</td>
</tr>
<tr>
<td>FlatStyle</td>
<td>ComboBoxCell.FlatStyle</td>
</tr>
<tr>
<td>Items</td>
<td>ComboBoxCell.Items</td>
</tr>
<tr>
<td>MaxDropDownItems</td>
<td>ComboBoxCell.MaxDropDownItems</td>
</tr>
<tr>
<td>Sorted</td>
<td>ComboBoxCell.Sorted</td>
</tr>
</tbody>
</table>

### DataGridviewComboBoxColumn

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoComplete</td>
<td>ComboBoxCell.AutoCompleteMode and ComboBoxCell.AutoCompleteSource</td>
</tr>
<tr>
<td>DisplayMember</td>
<td>ComboBoxCell.DisplayMember</td>
</tr>
<tr>
<td>DisplayStyle</td>
<td>ComboBoxCell.ShowDropDownButton</td>
</tr>
<tr>
<td>DisplayStyleForCurrentCellOnly</td>
<td>-</td>
</tr>
<tr>
<td>DropDownWidth</td>
<td>ComboBoxCell.DropDownWidth</td>
</tr>
<tr>
<td>FlatStyle</td>
<td>ComboBoxCell.FlatStyle</td>
</tr>
<tr>
<td>Items</td>
<td>ComboBoxCell.Items</td>
</tr>
<tr>
<td>MaxDropDownItems</td>
<td>ComboBoxCell.MaxDropDownItems</td>
</tr>
<tr>
<td>Sorted</td>
<td>ComboBoxCell.Sorted</td>
</tr>
</tbody>
</table>

### DataGridviewCell

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ToolTipText</td>
<td>Cell.ToolTipText</td>
</tr>
<tr>
<td>Visible</td>
<td>-</td>
</tr>
<tr>
<td>Width</td>
<td>Cell.Width</td>
</tr>
</tbody>
</table>

### DataGridviewComboBox

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoComplete</td>
<td>ComboBoxCell.AutoCompleteMode and ComboBoxCell.AutoCompleteSource</td>
</tr>
<tr>
<td>DisplayMember</td>
<td>ComboBoxCell.DisplayMember</td>
</tr>
<tr>
<td>DisplayStyle</td>
<td>ComboBoxCell.ShowDropDownButton</td>
</tr>
<tr>
<td>DisplayStyleForCurrentCellOnly</td>
<td>-</td>
</tr>
<tr>
<td>DropDownWidth</td>
<td>ComboBoxCell.DropDownWidth</td>
</tr>
<tr>
<td>FlatStyle</td>
<td>ComboBoxCell.FlatStyle</td>
</tr>
<tr>
<td>Items</td>
<td>ComboBoxCell.Items</td>
</tr>
<tr>
<td>MaxDropDownItems</td>
<td>ComboBoxCell.MaxDropDownItems</td>
</tr>
<tr>
<td>Sorted</td>
<td>ComboBoxCell.Sorted</td>
</tr>
</tbody>
</table>

### DataGridviewComboBoxColumn

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoComplete</td>
<td>ComboBoxCell.AutoCompleteMode and ComboBoxCell.AutoCompleteSource</td>
</tr>
<tr>
<td>DisplayMember</td>
<td>ComboBoxCell.DisplayMember</td>
</tr>
<tr>
<td>DisplayStyle</td>
<td>ComboBoxCell.ShowDropDownButton</td>
</tr>
<tr>
<td>DisplayStyleForCurrentCellOnly</td>
<td>-</td>
</tr>
<tr>
<td>DropDownWidth</td>
<td>ComboBoxCell.DropDownWidth</td>
</tr>
<tr>
<td>FlatStyle</td>
<td>ComboBoxCell.FlatStyle</td>
</tr>
<tr>
<td>Items</td>
<td>ComboBoxCell.Items</td>
</tr>
<tr>
<td>MaxDropDownItems</td>
<td>ComboBoxCell.MaxDropDownItems</td>
</tr>
<tr>
<td>Sorted</td>
<td>ComboBoxCell.Sorted</td>
</tr>
</tbody>
</table>

### DataGridviewCell

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ToolTipText</td>
<td>Cell.ToolTipText</td>
</tr>
<tr>
<td>Visible</td>
<td>-</td>
</tr>
<tr>
<td>Width</td>
<td>Cell.Width</td>
</tr>
</tbody>
</table>

### DataGridviewComboBox

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoComplete</td>
<td>ComboBoxCell.AutoCompleteMode and ComboBoxCell.AutoCompleteSource</td>
</tr>
<tr>
<td>DisplayMember</td>
<td>ComboBoxCell.DisplayMember</td>
</tr>
<tr>
<td>DisplayStyle</td>
<td>ComboBoxCell.ShowDropDownButton</td>
</tr>
<tr>
<td>DisplayStyleForCurrentCellOnly</td>
<td>-</td>
</tr>
<tr>
<td>DropDownWidth</td>
<td>ComboBoxCell.DropDownWidth</td>
</tr>
<tr>
<td>FlatStyle</td>
<td>ComboBoxCell.FlatStyle</td>
</tr>
<tr>
<td>Items</td>
<td>ComboBoxCell.Items</td>
</tr>
<tr>
<td>MaxDropDownItems</td>
<td>ComboBoxCell.MaxDropDownItems</td>
</tr>
<tr>
<td>Sorted</td>
<td>ComboBoxCell.Sorted</td>
</tr>
</tbody>
</table>

### DataGridviewComboBoxColumn

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoComplete</td>
<td>ComboBoxCell.AutoCompleteMode and ComboBoxCell.AutoCompleteSource</td>
</tr>
<tr>
<td>DisplayMember</td>
<td>ComboBoxCell.DisplayMember</td>
</tr>
<tr>
<td>DisplayStyle</td>
<td>ComboBoxCell.ShowDropDownButton</td>
</tr>
<tr>
<td>DisplayStyleForCurrentCellOnly</td>
<td>-</td>
</tr>
<tr>
<td>DropDownWidth</td>
<td>ComboBoxCell.DropDownWidth</td>
</tr>
<tr>
<td>FlatStyle</td>
<td>ComboBoxCell.FlatStyle</td>
</tr>
<tr>
<td>Items</td>
<td>ComboBoxCell.Items</td>
</tr>
<tr>
<td>MaxDropDownItems</td>
<td>ComboBoxCell.MaxDropDownItems</td>
</tr>
<tr>
<td>Sorted</td>
<td>ComboBoxCell.Sorted</td>
</tr>
</tbody>
</table>
### DataGridViewImageColumn class

- **Description**
  - ImageCell.Layout

  If the DataGridViewImageColumn.ImageLayout property is NotSet, the ImageCell.Layout property is set to ImageCellLayout.Normal.

### DataGridViewLinkColumn class

- **ActiveLinkColor**
  - LinkLabelCell.ActiveLinkColor

- **LinkBehavior**
  - LinkLabelCell.LinkBehavior

- **LinkColor**
  - LinkLabelCell.LinkColor

- **Text**
  - LinkLabelCell.Text

  If the DataGridViewLinkColumn.UseColumnTextForLinkValue property is set to True, it is converted to the LinkLabelCell.Value property; otherwise, the DataGridViewLinkColumn.Text property is ignored.

### DataGridViewCellStyle class

- **NullValue**
  - CellStyle.NullValue

- **Format**
  - CellStyle.Format

- **Alignment**
  - CellStyle.TextAlign

- **Padding**
  - CellStyle.Padding

- **WrapMode**
  - CellStyle.WordWrap and CellStyle.MultiLine

  Based on the setting of the DataGridViewCellStyle.WrapMode property, the CellStyle.WordWrap property and CellStyle.MultiLine property are set to the following values.

<table>
<thead>
<tr>
<th>DataGridViewCellStyle.WrapMode</th>
<th>GcMultiRow.WordWrap</th>
<th>GcMultiRow.MultiLine</th>
</tr>
</thead>
<tbody>
<tr>
<td>NotSet</td>
<td>Inherit</td>
<td>Inherit</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

### DataGridView class

- **Name**
  - GcMultiRow.Name

- **AccessibleDescription**
  - GcMultiRow.AccessibleDescription

- **AccessibleName**
  - GcMultiRow.AccessibleName

- **AccessibleRole**
  - GcMultiRow.AccessibleRole

- **AllowDrop**
  - GcMultiRow.AllowDrop

- **AllowUserToAddRows**
  - GcMultiRow.AllowUserToAddRows

- **AllowUserToDeleteRows**
  - GcMultiRow.AllowUserToDeleteRows

- **AllowUserToOrderColumns**
  - GcMultiRow.AllowUserToOrderColumns

- **AllowUserToResizeRows**
  - GcMultiRow.AllowUserToResizeRows

- **AllowUserToResizeColumns**
  - GcMultiRow.AllowUserToResizeColumns

  If the DataGridView.AllowUserToResizeRows property and the DataGridView.AllowUserToResizeColumns property are both set to False, the GcMultiRow.AllowUserToResize property is set to False; otherwise, the GcMultiRow.AllowUserToResize property is set to True.

- **AlternatingRowsDefaultCellStyle**
  - GcMultiRow.AlternatingRowsDefaultCellStyle

- **Anchor**
  - GcMultiRow.Anchor

- **AutoSizeColumnsMode**
  - GcMultiRow.HorizontalAutoSizeMode

  Based on the condition, the GcMultiRow.HorizontalAutoSizeMode property after conversion, is set to the following values.

<table>
<thead>
<tr>
<th>DataGridView.HorizontalAutoSizeMode</th>
<th>GcMultiRow.HorizontalAutoSizeMode</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllCells</td>
<td>AllCells</td>
</tr>
<tr>
<td>AllCellsExceptHeader</td>
<td>AllCellsExceptHeader</td>
</tr>
<tr>
<td>ColumnHeader</td>
<td>ColumnHeader</td>
</tr>
<tr>
<td>DisplayedCell</td>
<td>DisplayedCell</td>
</tr>
<tr>
<td>DisplayedCellExceptHeader</td>
<td>DisplayedCellExceptHeader</td>
</tr>
<tr>
<td>Fill</td>
<td>Fill</td>
</tr>
<tr>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>NotSet</td>
<td>NotSet</td>
</tr>
</tbody>
</table>

### DataGridViewRow class

- **BackgroundColor**
  - GcMultiRow.BackColor

- **BorderStyle**
  - GcMultiRow.BorderStyle

- **CausesValidation**
  - GcMultiRow.CausesValidation

- **CellBorderStyle**

  Based on the condition, the color of CellStyle.Border of GcMultiRow after conversion, is set to the following values.

<table>
<thead>
<tr>
<th>DataGridView.CellBorderStyle</th>
<th>GcMultiRow.Border of GcMultiRow</th>
</tr>
</thead>
<tbody>
<tr>
<td>SingleCustom</td>
<td>Color that is set to DataGridView.GridColor</td>
</tr>
<tr>
<td>Raised</td>
<td>Color that is set to DataGridView.GridColor</td>
</tr>
<tr>
<td>Sunken</td>
<td>The Top and Bottom of the border will be set to None.</td>
</tr>
<tr>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>
If any of the following is set:
- SingleHorizontal
- RaisedHorizontal
- SunkenHorizontal

Color that is set to DataGridView.GridColor

The Left and Right of the border will be set to None.

ClipboardCopyMode

GcMultiRow.AllowClipboard and GcMultiRow.ClipboardCopyModeM

Based on the condition, the GcMultiRow.AllowClipboard property and GcMultiRow.ClipboardCopyMode property after conversion, are set to the following values.

<table>
<thead>
<tr>
<th>DataGridViewClipboardCopyMode</th>
<th>GcMultiRow.AllowClipboard</th>
<th>GcMultiRow.ClipboardCopyMode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disabled</td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>EnableWithAutoHeaderText</td>
<td>True</td>
<td>EnableWithHeaderText</td>
</tr>
<tr>
<td>EnableWithoutHeaderText</td>
<td>True</td>
<td>EnableWithoutHeaderText</td>
</tr>
<tr>
<td>EnableAlwaysIncludeHeaderText</td>
<td>True</td>
<td>EnableWithHeaderText</td>
</tr>
</tbody>
</table>

ColumnHeadersBorderStyle

- ColumnHeadersDefaultCellStyle

GcMultiRow.ColumnsDefaultHeaderCellStyle

ColumnHeadersHeight

ColumnHeaderSection.Height

Resizable and ColumnHeadersHeightSizeMode

ColumnHeaderCell.ResizeMode

Based on the condition, the ColumnHeaderCell.ResizeMode property after conversion, is set to the following values.

<table>
<thead>
<tr>
<th>DataGridViewColumnResizable</th>
<th>ColumnHeadersHeightSizeMode</th>
<th>GcMultiRow.ResizeMode</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>EnableResizing</td>
<td>Both</td>
</tr>
<tr>
<td>True</td>
<td>DisableResizing or AutoResize</td>
<td>Vertical</td>
</tr>
<tr>
<td>False</td>
<td>EnableResizing</td>
<td>None</td>
</tr>
<tr>
<td>False</td>
<td>DisableResizing or AutoResize</td>
<td>None</td>
</tr>
</tbody>
</table>

Locked

GcMultiRow.Locked

MinimumSize

GcMultiRow.MinimumSize

Modifiers

- GcMultiRow.Modifiers

ReadOnly

GcMultiRow.ReadOnly

RowHeadersBorderStyle

- RowHeadersDefaultCellStyle

GcMultiRow.RowsDefaultHeaderCellStyle

RowHeadersWidth

RowHeaderCell.Width

Resizable and RowHeadersWidthSizeMode

RowHeaderCell.ResizeMode

Based on the condition, the RowHeaderCell.ResizeMode property after conversion, is set to the following values.

<table>
<thead>
<tr>
<th>DataGridRowTemplateResizable</th>
<th>RowHeadersWidthSizeMode</th>
<th>GcMultiRow.ResizeMode</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>EnableResizing</td>
<td>Both</td>
</tr>
<tr>
<td>True</td>
<td>If any of the following is set:</td>
<td>Vertical</td>
</tr>
</tbody>
</table>

Copyright © GrapeCity, inc. All rights reserved.
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DisableResizing</td>
<td>False if any of the following is set: DisableResizing, AutoSizeToAllHeaders, AutoSizeToDisplayedHeaders, AutoSizeToFirstHeader.</td>
</tr>
<tr>
<td>EnableResizing</td>
<td>False if RowTemplate.Resizable is NotSet.</td>
</tr>
<tr>
<td>HorizontalAutoSizeMode</td>
<td>True if RowTemplate.Resizable is NotSet. Tab key: ComponentActions.NextControl, Shift + Tab key: ComponentAtions.PreviousControl.</td>
</tr>
<tr>
<td>ShowCellErrors</td>
<td>If RowTemplate.ShowCellErrors is NotSet, then the setting value of DataGridView.AllowUserToResizeRows is used instead of RowTemplate.Resizable.</td>
</tr>
<tr>
<td>ShowCellToolTips</td>
<td>The setting value after conversion, will have one of the four aforementioned patterns.</td>
</tr>
<tr>
<td>ShowEditingIcon</td>
<td>GcMultiRow.ShowEditingIcon</td>
</tr>
<tr>
<td>ShowRowErrors</td>
<td>GcMultiRow.ShowRowErrors</td>
</tr>
<tr>
<td>Size</td>
<td>GcMultiRow.Size</td>
</tr>
<tr>
<td>StandardTab</td>
<td>Based on the condition, the following actions are assigned to the Tab and Shift + Tab keys of GcMultiRow after conversion.</td>
</tr>
</tbody>
</table>

**Using Annotations**

The Designer provides annotations that you can use to set comments that are displayed only at design time. By using annotations, developers can share design time information as comments.

**Configuring of Annotations**

The configuration is as follows.
- **ConnectingLine**
  A line that connects the annotation with the cell in which the annotation is added.

- **AuthorInitials**
  The number is added after the AuthorInitials. By default, "Memo" is displayed.

- **AuthorName**
  Name of the annotation's author. By default, it displays the login user name (SystemInformation.UserName property) information.

- **SeparatorSymbol**
  Hyphen character that is displayed before the date.

- **Date**
  The date on which the annotation was created. By default, it is displayed in the [yyyy/MM/dd HH:mm:ss] format.

- **CaptionBar**
  The area that displays the title of the annotation.

- **Shadow**
  Displays all the annotation with a shadow.

- **TextBoxEditing Control**
  This area is used to enter a comment.

- **Border**
  Border of the annotation.

### Customizing Annotations

You can change the display of the AuthorInitials, AuthorName, and Date. You can change the display by setting the Annotations.AuthorInitials, Annotations.AuthorName, and Annotations.DateFormat in **Designer Options**.

### Toolbar Operations

Annotations can perform the following operations, using the toolbar commands:

- **Create**
Creates an annotation for the selected cell. Cannot be operated if an annotation is already added to the cell that is selected.

- **Show/Hide**
  Shows or hides the annotation of the selected cell. Cannot be used if there is no annotation in the cell that is selected.

- **Show All/Hide All**
  Shows all or hides all the annotations on the Designer.

- **Delete**
  Deletes the annotation of the selected cell. Cannot be used if there is no annotation in the cell that is selected.

**Limitations**

- You cannot change the date or author's name of the created annotation.
- The font style of CaptionBar is displayed in italics. You cannot change the style.
- Annotations cannot be resized.
- The scroll bar is not displayed if a character string that is longer than the size of the annotation, is entered.
- Annotations do not support zoom.

**Template**

This section explains the settings required to create a template using code. To create a template using Designer, refer to **Using the Designer** and **Advanced Designer Features**.

- **Adding Cells**
- **Adding Column Headers**
- **Adding Column Footers**
- **Template Width and Height**
- **Column Mode Template**

**Adding Cells**

Use the Template.Row.Add method to add cells to the template using code.

**Using Code**

The following sample code adds a **ColumnHeaderCell** ('ColumnHeaderCell Class' in the on-line documentation) to the column header and a **TextBoxCell** ('TextBoxCell Class' in the on-line documentation) to rows in the template.

[Vb]

```vbnet
Imports GrapeCity.Win.MultiRow

Dim template As Template = New Template()
Dim columnHeaderSection1 As ColumnHeaderSection = New ColumnHeaderSection()
Dim columnHeaderCell1 As ColumnHeaderCell = New ColumnHeaderCell()
Dim textBoxCell1 As TextBoxCell = New TextBoxCell()

Dim textBoxCell11 As TextBoxCell = New TextBoxCell()
template.Width = textBoxCell11.Size.Width
template.Row.Cells.Add(textBoxCell11)
template.Row.Height = textBoxCell11.Size.Height

columnHeaderSection1.Cells.Add(columnHeaderCell1)
columnHeaderSection1.Height = columnHeaderCell1.Size.Height
```

Copyright © GrapeCity, inc. All rights reserved.
Adding Column Headers

Use the ColumnHeaderSectionCollection object in the Template. ColumnHeaders (ColumnHeaders Property' in the on-line documentation) property to work with column headers in the template.

Adding a Column Header

Create an instance of the ColumnHeaderSection (ColumnHeaderSection Class' in the on-line documentation) class and use the ColumnHeaderSectionCollection. Add ('Add Method' in the on-line documentation) method to add a column header in code.

Using Code

This example creates an instance of the ColumnHeaderSection class and adds a column header.

[VB]

Imports GrapeCity.Win.MultiRow

Dim template As Template = New Template()  
Dim columnHeaderSection As ColumnHeaderSection = new ColumnHeaderSection()  
Dim columnHeaderCell1 As ColumnHeaderCell = New ColumnHeaderCell()  
columnHeaderCell1.Location = New Point(0, 0)  
columnHeaderCell1.Value = "Col1"  
columnHeaderSection.Cells.Add(columnHeaderCell1)  
columnHeaderSection.Height = columnHeaderCell1.Size.Height

Dim textBoxCell1 As TextBoxCell = New TextBoxCell()  
textBoxCell1.Name = "textBoxCell1"
Deleting a Column Header

Use ColumnHeaderSectionCollection.Remove ('Remove Method' in the on-line documentation) or RemoveAt ('RemoveAt Method' in the on-line documentation) methods to delete the column header.

Adding Column Footers

Use the ColumnFooterSectionCollection object in the Template.ColumnFooters ('ColumnFooters Property' in the on-line documentation) property to work with column footers in the template.

Adding a Column Footer

Create an instance of the ColumnFooterSection ('ColumnFooterSection Class' in the on-line documentation) class and use the ColumnFooterSectionCollection.Add ('Add Method' in the on-line documentation) method to add a column footer in code.

Using Code

This example creates an instance of the ColumnFooterSection class and adds a footer.

[VB]
Imports GrapeCity.Win.MultiRow

Dim template As Template = New Template()

Dim columnHeaderSection As ColumnHeaderSection = new ColumnHeaderSection()
Dim columnHeaderCell1 As ColumnHeaderCell = New ColumnHeaderCell()
columnHeaderCell1.Location = New Point(0, 0)
columnHeaderCell1.Value = "Col1"
columnHeaderSection.Cells.Add(columnHeaderCell1)
columnHeaderSection.Height = columnHeaderCell1.Height

Dim textBoxCell1 As TextBoxCell = New TextBoxCell()
textBoxCell1.Name = "textBoxCell1"
textBoxCell1.Location = New Point(0, 0)

Dim columnFooterSection As ColumnFooterSection = new ColumnFooterSection()
Dim headerCell1 As HeaderCell = new HeaderCell()
headerCell1.Location = New Point(0, 0)
headerCell1.Value = "Footer"
columnFooterSection.Cells.Add(headerCell1)
columnFooterSection.Height = headerCell1.Height

template.Width = textBoxCell1.Width
template.Row.Cells.Add(textBoxCell1)
template.Row.Height = textBoxCell1.Height
template.ColumnHeaders.Add(columnHeaderSection)
template.ColumnFooters.Add(columnFooterSection)

GcMultiRow1.Template = template
GcMultiRow1.RowCount = 10

[CS]

using GrapeCity.Win.MultiRow;

Template template = new Template();

ColumnHeaderSection columnHeaderSection = new ColumnHeaderSection();
ColumnHeaderCell columnHeaderCell1 = new ColumnHeaderCell();
columnHeaderCell1.Location = new Point(0, 0);
columnHeaderCell1.Value = "Col1";
columnHeaderSection.Cells.Add(columnHeaderCell1);
columnHeaderSection.Height = columnHeaderCell1.Height;

TextBoxCell textBoxCell1 = new TextBoxCell();
textBoxCell1.Name = "textBoxCell1";
textBoxCell1.Location = new Point(0, 0);

ColumnFooterSection columnFooterSection = new ColumnFooterSection();
HeaderCell headerCell1 = new HeaderCell();
headerCell1.Location = new Point(0, 0);
headerCell1.Value = "Footer";
columnFooterSection.Cells.Add(headerCell1);
columnFooterSection.Height = headerCell1.Height;

template.Width = textBoxCell1.Width;
Deleting a Column Footer

Use ColumnFooterSectionCollection.Remove ('Remove Method' in the on-line documentation) or RemoveAt ('RemoveAt Method' in the on-line documentation) methods to delete the column footer.

Template Width and Height

You can set default values for the width and height of a template. Grids created at runtime are based on the template width and height.

Template Width

Set the template width using the Template.Width ('Width Property' in the on-line documentation) property. The value set is used for the width of rows, column headers, and column footers.

Using Code

The following code sets the template width to 800 pixels.

[VB]

Imports GrapeCity.Win.MultiRow

' Assign default template.
Dim template1 As Template = Template.Default
template1.Width = 800
GcMultiRow1.Template = template1

[CS]

using GrapeCity.Win.MultiRow;

' Assign default template.
Template template1 = Template.Default;
template1.Width = 800;
gcMultiRow1.Template = template1;

To modify the width of an existing template, assign the GcMultiRow.Template property value to a Template class object and set the width in it. Then reassign the object to the GcMultiRow.Template property.

Using Code

This example modifies the width of a template.

[VB]

Imports GrapeCity.Win.MultiRow
' Assign the default template.
GcMultiRow1.Template = Template.Default
Dim template1 As Template = GcMultiRow1.Template
' Modify template width.
template1.Width = 800
GcMultiRow1.Template = template1

' This code is not applicable.
' GcMultiRow1.Template.Width = 800

[CS]

using GrapeCity.Win.MultiRow;

// Assign the default template.
gcMultiRow1.Template = Template.Default;
Template template1 = gcMultiRow1.Template;
// Modify template width.
template1.Width = 800;
gcMultiRow1.Template = template1;

// This code is not applicable.
// gcMultiRow1.Template.Width = 800;

Template Height

You can set the height of rows, column headers, and column footers in the template.

Using Code

The following sample code modifies the height of rows, column headers, and column footers and changes the background color to display each region clearly.

[VB]

Imports GrapeCity.Win.MultiRow

Dim template As Template = New Template()
template.Row.Height = 20
template.Row.BackColor = Color.LightBlue

Dim columnHeaderSection1 As ColumnHeaderSection = New ColumnHeaderSection()
columnHeaderSection1.Height = 20
columnHeaderSection1.BackColor = Color.Azure
template.ColumnHeaders.Add(columnHeaderSection1)

Dim columnFooterSection1 As ColumnFooterSection = New ColumnFooterSection()
columnFooterSection1.Height = 20
columnFooterSection1.BackColor = Color.LightCoral
template.ColumnFooters.Add(columnFooterSection1)

GcMultiRow1.Template = template
GcMultiRow1.RowCount = 100

[CS]
using GrapeCity.Win.MultiRow;

Template template = new Template();
template.Row.Height = 20;
template.Row.BackColor = Color.LightBlue;
template.Row.Border = new Border(LineStyle.Thin, Color.Black);

ColumnHeaderSection columnHeaderSection1 = new ColumnHeaderSection();
columnHeaderSection1.Height = 20;
columnHeaderSection1.BackColor = Color.Azure;
template.ColumnHeaderSections.Add(columnHeaderSection1);

ColumnFooterSection columnFooterSection1 = new ColumnFooterSection();
columnFooterSection1.Height = 20;
columnFooterSection1.BackColor = Color.LightCoral;
template.ColumnFooters.Add(columnFooterSection1);

gcMultiRow1.Template = template;
gcMultiRow1.RowCount = 100;

Column Mode Template

By using the Column Mode, you can repeatedly display rows from left to right and use them. This topic describes the features of the Column Mode Template.

Setting the Column Mode

To use the column mode, set the Template.**LayoutMode** *(LayoutMode Property* in the on-line documentation*) property to LeftToRight.

**LayoutMode property**  TopToBottom  LeftToRight

**Design Time**

![Layout Mode Examples](image)

**Runtime**

Expands Vertically  Expands Horizontally

Layout
In the case the template is in row mode and column mode, the objects which are get or set by the following properties and methods, are different.

<table>
<thead>
<tr>
<th>Property/Method</th>
<th>TopToBottom</th>
<th>LeftToLeft</th>
</tr>
</thead>
<tbody>
<tr>
<td>Template.\textit{Width} ('Width Property' in the on-line documentation)</td>
<td>When getting: Width of template When setting: Width of template</td>
<td>When getting: Overall width of all the sections When setting: Width of template</td>
</tr>
<tr>
<td>Template.\textit{Height} ('Height Property' in the on-line documentation)</td>
<td>When getting: Overall height of all the sections When setting: Cannot be set</td>
<td>When getting: Height of template When setting: Height of template</td>
</tr>
<tr>
<td>Section.\textit{Width} ('Width Property' in the on-line documentation)</td>
<td>When getting: Width of template When setting: Width of the section</td>
<td>When getting: Section width When setting: Width of the section</td>
</tr>
<tr>
<td>Section.\textit{Height} ('Height Property' in the on-line documentation)</td>
<td>When getting: Height of the section When setting: Height of the section</td>
<td>When getting: Height of template When setting: Height of template</td>
</tr>
<tr>
<td>GcMultiRow.\textit{SectionWidth} ('SectionWidth Property' in the on-line documentation)</td>
<td>When getting: Width of template</td>
<td>When getting: Overall width of all the sections</td>
</tr>
<tr>
<td>GcMultiRow.\textit{SectionHeight} ('SectionHeight Property' in the on-line documentation)</td>
<td>When getting: Overall height of all the sections</td>
<td>When getting: Height of template</td>
</tr>
<tr>
<td>RowCollection.\textit{GetRowsHeight} ('GetRowsHeight Method' in the on-line documentation)</td>
<td>Return value: Overall height of the specified row</td>
<td>Return value: Height of template</td>
</tr>
<tr>
<td>RowCollection.\textit{GetRowsWidth} ('GetRowsWidth Method' in the on-line documentation)</td>
<td>Return value: Width of template</td>
<td>Return value: Overall width of the specified row</td>
</tr>
</tbody>
</table>

**TopToBottom Template**

![Diagram showing TopToBottom template]

**LeftToLeft Template**

![Diagram showing LeftToLeft template]
Operation of SelectionActions

The main operations of the SelectionActions in the Column Mode are as follows:

**SelectionActions**  
**Action**

MoveDown/MoveUp/MoveLeft/MoveRight

MoveToFirstCell

MoveToLastCell
MoveToPreviousCell

MoveToNextCell

MoveToFirstCellInRow

MoveToLastCellInRow

MoveToFistRow
MoveToLastRow

Sorting
In the Column Mode, sorting is performed on cells in the horizontal direction of the grid.

Merging
In the Column Mode, the auto-merge feature is enabled for cells in the horizontal direction of the grid.

Filtering
In the Column Mode, filtering is performed on cells in the horizontal direction of the grid.

Style of the Current Row
If you set the style of the current row, the style is applied to the column in the following manner:

Copying the Value of the Cell to the Clipboard
In the Column Mode, if you have copied the value of the cell of GcMultiRow control to the clipboard, it will be copied in reverse to what is being displayed in the rows and columns.

Grid
The GcMultiRow control creates grids based on templates.

- Assigning and Allocating a Template
Assigning and Allocating a Template

Rows with the same layout as the template are created at runtime by assigning a template to the GcMultiRow control. The template assigned to the grid is read-only. In order to change the template you need to reassign it to the grid.

**Assigning a Template**

Use the GcMultiRow.Template (Template Property' in the on-line documentation) property to assign a template to the grid. You can assign an instance of the Template (Template Class' in the on-line documentation) class or an instance of the class inherited from the Template class to the Template property.

**Using Code**

The following code assigns an empty template.

**[VB]**


**[CS]**

gcMultiRow1.Template = new GrapeCity.Win.MultiRow.Template();

حذر: The current values in the grid are discarded when a template is assigned to the grid. If the grid is bound to data, the values are reloaded. In unbound mode, you can set the GcMultiRow.RestoreValue (RestoreValue Property' in the on-line documentation) property to True to restore the cell values once the template is changed.

**Assigning Default Template**

When the GcMultiRow control is placed on a form using the Visual Studio form designer, it is placed without a template. To set the default template using the form designer, select "Default Template" from the smart tag. You can use the Template.Default (Default Property' in the on-line documentation) property to set the default template while creating a template with code.

**Using Code**
This example sets the default template.

[VB]


[CS]


Referencing a Template

Refer to the Template ('Template Class' in the on-line documentation) class instance with the GcMultiRow.Template ('Template Property' in the on-line documentation) property to reference the template assigned to the grid.

Using Code

The following code lists the current column width of the grid and the column width of the template.

[VB]

Imports GrapeCity.Win.MultiRow

' Set the default template.
GcMultiRow1.Template = Template.Default
' Change the cell width.
GcMultiRow1.Rows(0).Cells(1).HorizontalResize(100)

Console.WriteLine("Width of grid cell: {0}", GcMultiRow1.Rows(0).Cells(1).Width)

[CS]

using GrapeCity.Win.MultiRow;

// Set the default template.
gcMultiRow1.Template = Template.Default;
// Change the cell width.
gcMultiRow1.Rows[0].Cells[1].HorizontalResize(100);

Console.WriteLine("Width of template cell: {0}", gcMultiRow1.Template.Row.Cells[1].Width);
Console.WriteLine("Width of grid cell: {0}", gcMultiRow1.Rows[0].Cells[1].Width);

If you have changed the Template ('Template Class' in the on-line documentation) class instance with the GcMultiRow.Template ('Template Property' in the on-line documentation) property, this change will not be reflected in the grid. To reflect the change, you need to reassign the template.

Edit Modes

GcMultiRow supports three types of edit modes. You can set the edit mode using the GcMultiRow.EditMode ('EditMode Property' in the on-line documentation) property and the GcMultiRow.ReadOnly ('ReadOnly Property' in the on-line documentation) property. Refer to the Shortcut Keys section for details on shortcuts for each of these modes. Refer to the Editing Cells section for details regarding the advanced customization of editing cells.
Default Mode

You can select or edit the cells in the default mode. By default, the input mode begins when user enters in the cell or double clicks on it. Input is confirmed by pressing the **Enter** key or by moving the focus to other cells. This mode is the same as that in Excel or DataGridView. This is the default edit mode for the GcMultiRow control.

Using Code

The following code sets the grid to the default edit mode.

[VB]

```vbnet
```

[CS]

```csharp
```

Continuous Input Mode

In continuous input mode, the cells are always in edit mode. This mode is the same as the TextBox control. The cells can be edited as soon as the user moves the focus to the cell.

Using Code

This example puts the grid into continuous input mode.

[VB]

```vbnet
```

[CS]

```csharp
```

- Continuous input mode is disabled when the GcMultiRow.**ReadOnly** (ReadOnly Property' in the on-line documentation) property is set to True or the GcMultiRow.**ViewMode** (ViewMode Property' in the on-line documentation) property is set to Display.
- It is not possible to select cells by dragging when continuous input mode is enabled.
- Cell editing starts automatically when the cells are moved using the keyboard, so scrolling is slower when using the keyboard in the continuous input mode.
- In continuous input mode, editing results cannot be confirmed until you have moved off the cell. Be
careful when referencing the value of a cell in edit mode because the values in the cell editing control and the values in the grid cell are different. The cell editing control retains the value being edited and the grid cell contains the value before it was edited. This behavior is also true in cases where continuous input mode has been implemented using a procedure other than using the GcMultiRow.EditMode (‘EditMode Property’ in the on-line documentation) property.

Continuous input mode works for all editable cell types. In order to decide whether to use continuous input mode based on cell types, you can set the GcMultiRowEditMode (‘EditMode Property’ in the on-line documentation) property to a value other than EditOnEnter, and call the GcMultiRow.BeginEdit (‘BeginEdit Method’ in the on-line documentation) method in the CellEnter (‘CellEnter Event’ in the on-line documentation) event.

Using Code

In the following sample code, continuous input mode is only enabled in string type cells.

[VB]

Imports GrapeCity.Win.MultiRow

Private Sub GcMultiRow1_CellEnter( ByVal sender As System.Object, ByVal e As CellEventArgs ) Handles GcMultiRow1.CellEnter
If TypeOf GcMultiRow1.CurrentCell Is TextBoxCell Then
GcMultiRow1.BeginEdit(False)
End If
End Sub

[CS]

using GrapeCity.Win.MultiRow;

private void gcMultiRow1_CellEnter(object sender, CellEventArgs e)
{
    if (gcMultiRow1.CurrentCell is TextBoxCell)
        gcMultiRow1.BeginEdit(false);
}

Read-Only Mode

Setting the GcMultiRow.ReadOnly (‘ReadOnly Property’ in the on-line documentation) property to True, allows you to set the grid to read-only regardless of the settings in the GcMultiRow.EditMode (‘EditMode Property’ in the on-line documentation) property.

Using Code

The following example sets the grid to be read-only.

[VB]

GcMultiRow1.ReadOnly = True

[CS]

gcMultiRow1.ReadOnly = true;
You can set the GcMultiRow.**EditMode** (**EditMode Property** in the on-line documentation) property to False and the ReadOnly property of each of the cells to True, in order to allow cell editing but restrict editing of the content (similar to a read-only text box). You can select the display mode if you do not wish to allow cell selection. Refer to the **View Modes** section for details regarding display mode.

**Selection Mode**

The GcMultiRow control provides two types of user selection modes. The developer can add or delete the selected cells, retrieve the selection, and set the appearance of the selected cells. Refer to **Selecting Cells using Headers** for details on the selection of cells using headers.

**Multi Select Mode**

GcMultiRow supports the selection of multiple cells by default. The user can select multiple cells by dragging the cells, clicking on the header, or by clicking or dragging the cells while pressing the **Ctrl** key. The following image shows cells being dragged in the multi select mode. The triangle icon depicts the mouse cursor.

You can select discontinuous cells when you click on the cells while pressing the **Ctrl** key.

**Using Code**

The following code sets the **MultiSelect** (**MultiSelect Property** in the on-line documentation) property for the GcMultiRow control.

**[VB]**

```vbnet
GcMultiRow1.MultiSelect = True
```

**[CS]**

```csharp
gcMultiRow1.MultiSelect = true;
```

**Single Select Mode**

This specifies whether the developer will let the user select a single cell or multiple cells at a time. To enable the single select mode, set the GcMultiRow.MultiSelect property to False.
Using Code

The following code sets the single select mode for the GcMultiRow control.

[VB]

GcMultiRow1.MultiSelect = False

[CS]

gcMultiRow1.MultiSelect = false;

Cell Selection

You can select or delete the cell selection using the Cell.Selected (Selected Property’ in the on-line documentation) property.

Using Code

The following example selects a cell.

[VB]

GcMultiRow1.Rows(0).Cells(1).Selected = True

[CS]

gcMultiRow1.Rows[0].Cells[1].Selected = true;

For selecting and deleting the selection of rows, use the Row.Selected property (Section.Selected (Selected Property’ in the on-line documentation) property).

Using Code

The following example selects a row.

[VB]

GcMultiRow1.Rows(0).Selected = True

[CS]

gcMultiRow1.Rows[0].Selected = true;

Retrieving the Selection

To determine whether the cells or rows are selected, refer to Selected Rows and Cells.
Cell Appearance in the Selected State

The background and the foreground of the cells change to display them in a selected state when the cells are selected. The backcolor and forecolor use the CellStyle.SelectionBackColor ('SelectionBackColor Property' in the on-line documentation) property and the CellStyle.SelectionForeColor ('SelectionForeColor Property' in the on-line documentation) property, but the effect of the rendering varies depending on the rendering process and the style settings. For details, refer to Cell Styles.

Reversing the Selection

Set the GcMultiRow.AllowUserToReverseSelect ('AllowUserToReverseSelect Property' in the on-line documentation) property to True to reverse the selection of the already selected cells, or, in other words, to delete the selection of some of the cells from a selected range. In this case, the user can reverse the selection by pressing Ctrl with the mouse click.

Using Code

The following code lets users reverse the selection by pressing the Ctrl key while clicking.

[VB]
GcMultiRow1.AllowUserToReverseSelect = True

[CS]
gcMultiRow1.AllowUserToReverseSelect = true;

Multiple Selection using Shift Key

Set the GcMultiRow.AllowUserToShiftSelect ('AllowUserToShiftSelect Property' in the on-line documentation) property to True to be able to select multiple cells, either by specifying the start and end of the selection while pressing the Shift key, or by dragging the mouse over the selection while pressing the Shift key.

Using Code

The following example sets the grid to allow users to select multiple cells while pressing the Shift key.

[VB]
GcMultiRow1.AllowUserToShiftSelect = True

[CS]
gcMultiRow1.AllowUserToShiftSelect = true;

To allow mouse handling and to disable keyboard handling, delete the shortcut keys that are associated with this operation.

Using Code

The following code lets the user press Shift and use the mouse to select multiple cells, but not to use the keyboard and Shift to select multiple cells.

[VB]
Imports GrapeCity.Win.MultiRow
GcMultiRow1.ShortcutKeyManager.Unregister(SelectionActions.ShiftDown)
GcMultiRow1.ShortcutKeyManager.Unregister(SelectionActions.ShiftLeft)
GcMultiRow1.ShortcutKeyManager.Unregister(SelectionActions.ShiftPageDown)
GcMultiRow1.ShortcutKeyManager.Unregister(SelectionActions.ShiftPageUp)
GcMultiRow1.ShortcutKeyManager.Unregister(SelectionActions.ShiftRight)
GcMultiRow1.ShortcutKeyManager.Unregister(SelectionActions.ShiftToFirstCell)
GcMultiRow1.ShortcutKeyManager.Unregister(SelectionActions.ShiftToFirstCellInRow)
GcMultiRow1.ShortcutKeyManager.Unregister(SelectionActions.ShiftToFirstRow)
GcMultiRow1.ShortcutKeyManager.Unregister(SelectionActions.ShiftToLastCell)
GcMultiRow1.ShortcutKeyManager.Unregister(SelectionActions.ShiftToLastCellInRow)
GcMultiRow1.ShortcutKeyManager.Unregister(SelectionActions.ShiftToLastRow)
GcMultiRow1.ShortcutKeyManager.Unregister(SelectionActions.ShiftToNextRow)
GcMultiRow1.ShortcutKeyManager.Unregister(SelectionActions.ShiftToPreviousRow)
GcMultiRow1.ShortcutKeyManager.Unregister(SelectionActions.ShiftUp)

[CS]

using GrapeCity.Win.MultiRow;

gcMultiRow1.ShortcutKeyManager.Unregister(SelectionActions.ShiftDown);
gcMultiRow1.ShortcutKeyManager.Unregister(SelectionActions.ShiftLeft);
gcMultiRow1.ShortcutKeyManager.Unregister(SelectionActions.ShiftPageDown);
gcMultiRow1.ShortcutKeyManager.Unregister(SelectionActions.ShiftPageUp);
gcMultiRow1.ShortcutKeyManager.Unregister(SelectionActions.ShiftRight);
gcMultiRow1.ShortcutKeyManager.Unregister(SelectionActions.ShiftToFirstCell);
gcMultiRow1.ShortcutKeyManager.Unregister(SelectionActions.ShiftToFirstCellInRow);
gcMultiRow1.ShortcutKeyManager.Unregister(SelectionActions.ShiftToFirstRow);
gcMultiRow1.ShortcutKeyManager.Unregister(SelectionActions.ShiftToLastCell);
gcMultiRow1.ShortcutKeyManager.Unregister(SelectionActions.ShiftToLastCellInRow);
gcMultiRow1.ShortcutKeyManager.Unregister(SelectionActions.ShiftToLastRow);
gcMultiRow1.ShortcutKeyManager.Unregister(SelectionActions.ShiftToNextRow);
gcMultiRow1.ShortcutKeyManager.Unregister(SelectionActions.ShiftToPreviousRow);
gcMultiRow1ShortcutKeyManager.Unregister(SelectionActions.ShiftUp);

View Modes

The GcMultiRow control supports four types of view modes. View modes can be set using the GcMultiRow.ViewMode ('ViewMode Property' in the on-line documentation) property.

Default Mode

The current cell is selected with the default mode. The GcMultiRow.MultiSelect ('MultiSelect Property’ in the on-line documentation) property determines whether the selection of multiple cells is allowed.

The following image shows a cell being clicked in default mode. The triangle icon in the image indicates the mouse cursor.

```
   | A | B |
---+---+---
 1 |
 2 |
 3 |
```

Set the GcMultiRow.ViewMode ('ViewMode Property' in the on-line documentation) property to Default to
use the default mode.

**Using Code**

This example sets the `ViewMode` property to `Default`.

**[VB]**

```vbnet
```

**[CS]**

```csharp
```

**Row Mode**

In row mode, the entire row is selected by default. When row mode is enabled, the `GcMultiRow.MultiSelect` property is set to rows.

Set the `GcMultiRow.ViewMode` property to `Row` in order to use row mode.

**Using Code**

This example sets the ViewMode property to Row.

**[VB]**

```vbnet
```

**[CS]**

```csharp
```

**Display Mode**

In display mode, the user cannot work on or edit the control except for scrolling and resizing.

To use display mode, set the `GcMultiRow.ViewMode` property to `Display`.
Using Code

This example sets the ViewMode property to Display.

[VB]


[CS]


The following operations are disabled in display mode.

- Current cell and current row
- Adding rows using row editor (GcMultiRow.AllowUserToAddRows ('AllowUserToAddRows Property' in the on-line documentation))
- Deleting rows using Ctrl + Delete keys
- Selecting cell and moving focus
- Editing cell values

When display mode is enabled, the GcMultiRow.CurrentCell ('CurrentCell Property' in the on-line documentation) and GcMultiRow.CurrentRow ('CurrentRow Property' in the on-line documentation) properties return null reference (Nothing in Visual Basic).

ListBox Mode

In list box mode, you can toggle the row selection state by using the mouse click or by pressing the Space key. You need to set the GcMultiRow.ViewMode ('ViewMode Property' in the on-line documentation) property to ListBox in order to use list box mode.

Using Code

This example sets the ViewMode property to ListBox.

[VB]


[CS]


List box mode operates in the same way as a standard Listbox control (System.Windows.Forms.ListBox) that has the following settings.

[VB]

ListBox1.SelectionMode = SelectionMode.MultiSimple

[CS]

listBox1.SelectionMode = SelectionMode.MultiSimple;

The following operations are disabled in list box mode.
• Editing cell values
• Cutting and pasting cell values

Scrolling

The GcMultiRow control allows you to display or hide the scroll bar, set the scroll unit, set the scroll amount, specify rendering at the time of scrolling, set scroll tips, and so on.

Show or Hide the Scroll Bar

You can hide or display the scroll bar using the ScrollBars ('ScrollBars Property' in the on-line documentation) property.

Using Code

The following code only displays vertical scroll bars.

[VB]

GcMultiRow1.ScrollBars = ScrollBars.Vertical

[CS]

gcMultiRow1.ScrollBars = ScrollBars.Vertical;

You can temporarily hide the scroll bar, for example, when the entire grid is displayed in the view.

If you use the GcMultiRow.HorizontalScrollBarMode ('HorizontalScrollBarMode Property' in the on-line documentation) property for the horizontal scroll bar, and GcMultiRow.VerticalScrollBarMode ('VerticalScrollBarMode Property' in the on-line documentation) property for the vertical scroll bar, and set the respective properties to Automatic, the GcMultiRow control will display the scroll bar only when necessary.

Using Code

This example displays scroll bars when needed.

[VB]


[CS]


The user can still scroll the grid using the mouse if the grid’s data exceeds the display range of the grid even when the scroll bar has been hidden.

Scrolling By Pixels

The GcMultiRow control is scrolled row by row by default. The vertical scrolling can be executed in pixels by setting the ScrollMode ('ScrollMode Enumeration' in the on-line documentation) enumeration to Pixel. Horizontal scrolling is always executed in pixels.

Using Code

This example sets the ScrollMode property.
Changing the Scroll Count

The GcMultiRow.\texttt{VerticalScrollCount} (\texttt{VerticalScrollCount Property' in the on-line documentation}) and the GcMultiRow.\texttt{HorizontalScrollCount} (\texttt{HorizontalScrollCount Property' in the on-line documentation}) properties can be used to modify how much area is moved in one scroll. Scrolling speed can be modified based on the template size or layout using these settings.

Using Code

The following code sets a single scroll count for vertical scrolling to 10 pixels.

[VB]

\begin{verbatim}
GcMultiRow1.RowCount = 200
GcMultiRow1.VerticalScrollCount = 10
\end{verbatim}

[CS]

\begin{verbatim}
qcMultiRow1.RowCount = 200;
qcMultiRow1.VerticalScrollCount = 10;
\end{verbatim}

Use the GcMultiRow.\texttt{MouseWheelCount} (\texttt{MouseWheelCount Property' in the on-line documentation}) property to modify the scroll count for vertical scrolling when using the mousewheel.

\begin{itemize}
  \item Scrolling using the tiltwheel of a mouse (for left-right scrolling) is not supported.
\end{itemize}

Suppress Grid Updates while Dragging the Scroll Bar

By default the GcMultiRow grid is updated as soon as the user scrolls the grid. For example, when the user is dragging a scroll bar, the data in the grid is updated in real-time with the position of the scroll bar. This behavior ensures a fast response; however, if the grid is bound to data, especially when using virtual mode, it may affect the performance. Grid updates can be suppressed while dragging is underway, and resumed after the dragging is complete, with the GcMultiRow.\texttt{ScrollOnThumbTrack} (\texttt{ScrollOnThumbTrack Property' in the on-line documentation}) property. As a result, the load on the grid and the data source can be reduced.

Using Code

The following code allows the grid to be updated while the user is using the horizontal scroll bar. The grid is updated only after the dragging is complete if using the vertical scroll bar.

[VB]

\begin{verbatim}
\end{verbatim}
ScrollTip

Use the GcMultiRow.ShowScrollTip (‘ShowScrollTip Property’ in the on-line documentation) property to display information about the row and cell in a tooltip while scrolling. If this property is set to Both, the scrolltips display the row number during vertical scrolling, and pixel position during horizontal scrolling. Control the display positions of the scrolltips using the GcMultiRow.VerticalScrollTipAlignment (‘VerticalScrollTipAlignment Property’ in the on-line documentation) property for vertical scrolling and the GcMultiRow.HorizontalScrollTipAlignment (‘HorizontalScrollTipAlignment Property’ in the on-line documentation) property for horizontal scrolling. Scrolltip text can be set by using the ScrollTipTextNeeded (‘ScrollTipTextNeeded Event’ in the on-line documentation) event.

Using Code

The following code displays the index numbers of the scrolling destination cells in the form of tooltips while scrolling with the vertical and horizontal scroll bars.

[VB]

Imports GrapeCity.Win.MultiRow

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    GcMultiRow1.RowCount = 200
    GcMultiRow1.ShowScrollTip = ScrollBars.Vertical
End Sub

Private Sub GcMultiRow1_ScrollTipTextNeeded(ByVal sender As System.Object, ByVal e As ScrollTipTextNeededEventArgs) Handles GcMultiRow1.ScrollTipTextNeeded
    If e.Direction = Orientation.Vertical Then
        e.ScrollTipText = e.ScrollRowIndex.ToString()
    End If
End Sub

[CS]

using GrapeCity.Win.MultiRow;

private void Form1_Load(object sender, EventArgs e)
{
    gcMultiRow1.RowCount = 200;
    gcMultiRow1.ShowScrollTip = ScrollBars.Vertical;
}

private void gcMultiRow1_ScrollTipTextNeeded(object sender, ScrollTipTextNeededEventArgs e)
{
    if (e.Direction == Orientation.Vertical)
        e.ScrollTipText = e.ScrollRowIndex.ToString();
}

Scroll Bar Styles

The scroll bar style can be modified using the GcMultiRow.ScrollBarStyle (‘ScrollBarStyle Property’ in the on-line documentation) property.
documentation) property. For details, refer to Visual Style.

**Scrolling By Rows Using Code**

The cell displayed at the top left of the grid can be specified using the GcMultiRow. `FirstDisplayedCellPosition` (‘FirstDisplayedCellPosition Property’ in the on-line documentation) property.

**Using Code**

The following code scrolls the grid such that the first cell of the fifth row is displayed at the top left.

**[VB]**

```vbnet
GcMultiRow1.RowCount = 10
```

**[CS]**

```csharp
gcMultiRow1.RowCount = 10;
gcMultiRow1.FirstDisplayedCellPosition = new GrapeCity.Win.MultiRow.CellPosition(4, 0);
```

**Scrolling through Coordinates Using code**

You can specify the coordinates to be displayed on the top left of the grid using the GcMultiRow. `FirstDisplayedLocation` (‘FirstDisplayedLocation Property’ in the on-line documentation) property.

**Using Code**

This example sets the FirstDisplayedLocation property.

**[VB]**

```vbnet
GcMultiRow1.RowCount = 10
GcMultiRow1.FirstDisplayedLocation = New Point(200, 100)
```

**[CS]**

```csharp
gcMultiRow1.RowCount = 10;
gcMultiRow1.FirstDisplayedLocation = new Point(200, 100);
```

**Scrolling And Continuous Input Mode**

Usually, the scrolling speed decreases when continuous input mode is active. This happens because the cells invariably transit into an editable state with movement. For details on continuous input mode, refer to Edit Modes.

**Visual Style**

The GcMultirow control can use the visual effects (Visual Style) provided by the OS or provide an appearance similar to Office applications. The visual style can be set for cells and scroll bars.

- The cell backcolor (CellStyle. `BackColor (‘BackColor Property’ in the on-line documentation)) is not displayed when the visual style is enabled. Refer to the Visual Style and BackColor topic under the BackColor and ForeColor section for details on how to disable backcolor for each cell.

**Cell Style**
The cell style can be set using the FlatStyle property for each cell type. This property can be used in the following cell types:

- **HeaderCell** (HeaderCell Class' in the on-line documentation) (Including ColumnHeaderCell (ColumnHeaderCell Class' in the on-line documentation), RowHeaderCell (RowHeaderCell Class' in the on-line documentation), and CornerHeaderCell (CornerHeaderCell Class' in the on-line documentation))
- **ButtonCell** (ButtonCell Class' in the on-line documentation)
- **ComboBoxCell** (ComboBoxCell Class' in the on-line documentation)
- **CheckBoxCell** (CheckBoxCell Class' in the on-line documentation)
- **RadioGroupCell** (RadioGroupCell Class' in the on-line documentation)
- **PopupCell** (PopupCell Class' in the on-line documentation) (ButtonFlatStyle property instead of FlatStyle property)

**Office2007 Header Cell Style**

The header cell (ColumnHeaderCell, RowHeaderCell, and CornerHeaderCell) supports the Office 2007-like style. Set the Office2007Style property to enable this style. Refer to the Changing the Header Appearance section for details.

**Scroll bar Styles**

The scroll bar styles can be set using the GcMultiRow.ScrollBarStyle (ScrollBarStyle Property’ in the on-line documentation) property.

**Splitter Line Styles**

You can set styles for splitter lines using the GcMultiRow.SplitOffice2007Style (SplitOffice2007Style Property’ in the on-line documentation) property.

**Zooming**

The GcMultiRow control supports magnifying and scaling down (zooming in or out) the contents.

- Borders, scroll bars, and tooltips are not affected by zooming.
- The zoom factor is from 10% to 400%.

**Allow User to Zoom**

The user can zoom in and out of the grid contents by scrolling the mouse wheel in the forward and backward direction while pressing the Ctrl key when the GcMultiRow.AllowUserToZoom (AllowUserToZoom Property’ in the on-line documentation) property has been set to True.

**Using Code**

This example sets the AllowUserToZoom property to True.

[VB]

```vbnet
GcMultiRow1.AllowUserToZoom = True
```

[CS]

```csharp
gcMultiRow1.AllowUserToZoom = true;
```

To restrict zooming by the user, set the GcMultiRow.AllowUserToZoom property to False.

**Get or Set Zoom Factor**
Developers can get or set the current grid zoom factor using the GcMultiRow.**ZoomFactor** ("ZoomFactor Property" in the on-line documentation) property.

### Using Code

The following code sets the zoom factor of the GcMultiRow control to 200%.

**[VB]**

```vbnet
GcMultiRow1.ZoomFactor = 2.0F
```

**[CS]**

```vbnet
gcMultiRow1.ZoomFactor = 2.0f;
```

The following code assigns the **Ctrl** + ↑ arrow key to zoom in, and **Ctrl** + ↓ to zoom out.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

GcMultiRow1ShortcutKeyManager.Register(New ZoomInAction(), Keys.Control Or Keys.Up)
GcMultiRow1ShortcutKeyManager.Register(New ZoomOutAction(), Keys.Control Or Keys.Down)

Public Class ZoomInAction
    Implements IAction

    Public Function CanExecute(ByVal target As GcMultiRow) As Boolean Implements IAction.CanExecute
        If target.AllowUserToZoom = True Then
            Return True
        End If
        Return False
    End Function

    Public ReadOnly Property DisplayName() As String Implements IAction.DisplayName
        Get
            Return Me.ToString()
        End Get
    End Property

    Public Sub Execute(ByVal target As GcMultiRow) Implements IAction.Execute
        Try
            target.ZoomFactor *= 2
        Catch ex As Exception
            Throw
        End Try
    End Sub
End Class

Public Class ZoomOutAction
    Implements IAction

    Public Function CanExecute(ByVal target As GcMultiRow) As Boolean Implements IAction.CanExecute
```

---

**Copyright © GrapeCity, inc. All rights reserved.**
Implements IAction.CanExecute
If target.AllowUserToZoom = True Then
    Return True
End If
Return False
End Function

Public ReadOnly Property DisplayName() As String _
    Implements IAction.DisplayName
Get
    Return Me.ToString()
End Get
End Property

Public Sub Execute(ByVal target As GcMultiRow) _
    Implements IAction.Execute
Try
    target.ZoomFactor /= 2
Catch ex As Exception
    Throw
End Try
End Sub
End Class

[CS]

using GrapeCity.Win.MultiRow;

gcMultiRow1.ShortcutKeyManager.Register(new ZoomInAction(), Keys.Control | Keys.Up);
gcMultiRow1.ShortcutKeyManager.Register(new ZoomOutAction(), Keys.Control | Keys.Down);

public class ZoomInAction : IAction
{
    public bool CanExecute(GcMultiRow target)
    {
        if (target.AllowUserToZoom == true)
            return true;
        return false;
    }

    public string DisplayName
    {
        get { return this.ToString(); } }

    public void Execute(GcMultiRow target)
    {
        try
        {
            target.ZoomFactor *= 2;
        }
        catch (Exception)
        {
            throw;
        }
    }
}
public class ZoomOutAction : IAction
{
    public bool CanExecute(GcMultiRow target)
    {
        if (target.AllowUserToZoom == true)
        {
            return true;
        }
        return false;
    }

    public string DisplayName
    {
        get { return this.ToString(); }
    }

    public void Execute(GcMultiRow target)
    {
        try
        {
            target.ZoomFactor /= 2;
        }
        catch (Exception)
        {
            throw;
        }
    }
}

Splitter Line and ViewPort

You can use splitter lines to split the grid into multiple screens and scroll each screen (viewport) individually. The user can freely adjust the size of the screen using the splitter line. Refer to the Freezing Rows section for details on how to freeze a section of the screen and keep it unaffected by the user’s actions.

Displaying or Hiding the Splitter Line

By default, the user can add or delete the splitter line by simply dragging the splitter line that is displayed at the end of a scroll bar. You can hide the splitter line using the GcMultiRow.SplitMode ('SplitMode Property' in the on-line documentation) property.

Using Code

This example hides the splitter line.

[VB]

[CS]

Retrieving the Location of the Splitter Line

You can retrieve the location of the splitter using the GcMultiRow.GetHorizontalSplitLocations
('GetHorizontalSplitLocations Method' in the on-line documentation) and
GcMultiRow.GetVerticalSplitLocations ('GetVerticalSplitLocations Method' in the on-line
documentation) methods.

Using Code

This example gets the splitter location.

[VB]
Dim hsLocations As List(Of Integer) = GcMultiRow1.GetHorizontalSplitLocations()
Console.WriteLine("Count of horizontal splitter bars:{0}", hsLocations.Count)
Console.WriteLine("Location of the 1st horizontal splitter bar:{0}", hsLocations(0))

Dim vsLocations As List(Of Integer) = GcMultiRow1.GetVerticalSplitLocations()
Console.WriteLine("Count of vertical splitter bars:{0}", vsLocations.Count)
Console.WriteLine("Location of the 1st vertical splitter bar:{0}", vsLocations(0))

[CS]
List<int> hsLocations = gcMultiRow1.GetHorizontalSplitLocations();
Console.WriteLine("Count of horizontal splitter bars:{0}", hsLocations.Count);
Console.WriteLine("Location of the 1st horizontal splitter bar:{0}", hsLocations[0]);

List<int> vsLocations = gcMultiRow1.GetVerticalSplitLocations();
Console.WriteLine("Count of vertical splitter bars:{0}", vsLocations.Count);
Console.WriteLine("Location of the 1st vertical splitter bar:{0}", vsLocations[0]);

Adding ViewPort

Each separate screen that was created by splitting the display screen is called a viewport. In other words, adding a
splitter line and adding a viewport are synonymous to each other. The developer can use the
GcMultiRow.AddViewport ('AddViewport Method' in the on-line documentation) method to add a viewport.

Using Code

The following code adds a vertical splitter line to divide the grid horizontally.

[VB]
GcMultiRow1.AddViewport(-1, 0)

[CS]
gcMultiRow1.AddViewport(-1, 0);

The following code adds a horizontal splitter line to divide the grid vertically.

[VB]
GcMultiRow1.AddViewport(0, -1)

[CS]
gcMultiRow1.AddViewport(0, -1);
Getting the Number of ViewPorts

The number of horizontal view ports can be retrieved using the GcMultiRow.GetHorizontalViewportCount ("GetHorizontalViewportCount Method' in the on-line documentation) method. The number of vertical view ports can be retrieved using the GcMultiRow.GetVerticalViewportCount ("GetVerticalViewportCount Method' in the on-line documentation) method. The number of splitter lines is one less than the number of view ports.

Deleting Splitter Line

You can delete the splitter line using the GcMultiRow.RemoveSplit ("RemoveSplit Method' in the on-line documentation) method.

Using Code

This example removes the splitter line.

[VB]
GcMultiRow1.RemoveSplit(0, Orientation.Vertical)

[CS]
gcMultiRow1.RemoveSplit(0, Orientation.Vertical);

A System.ArgumentOutOfRangeException exception occurs if a value exceeding the number of splitter lines is set.

Splitter Line Styles

You can change the backcolor or width of the bar using the GcMultiRow.SplitStyle ("SplitStyle Property' in the on-line documentation) property.

Using Code

This example sets the color and width.

[VB]
GcMultiRow1.AddViewport(-1, 0)
Dim splitStyle1 As New GrapeCity.Win.MultiRow.SplitStyle()
splitStyle1.BackColor = Color.Azure
splitStyle1.DarkColor = Color.Blue
splitStyle1.LightColor = Color.LightBlue
splitStyle1.Width = 6
GcMultiRow1.SplitStyle = splitStyle1

[CS]
gcMultiRow1.AddViewport(-1, 0);
splitStyle1.BackColor = Color.Azure;
splitStyle1.DarkColor = Color.Blue;
splitStyle1.LightColor = Color.LightBlue;
splitStyle1.Width = 6;
gcMultiRow1.SplitStyle = splitStyle1;

The Office2007 style has priority if a value other than None is set in the GcMultiRow.SplitOffice2007Style
Using Code

This example sets the style.

[VB]
GcMultiRow1.AddViewport(-1, 0)

[CS]
gcMultiRow1.AddViewport(-1, 0);

SplitStyle Editor

If you select the GcMultiRow control and then click the ellipsis button of the GcMultiRow.SplitStyle property in the Properties window, you get a dialog on which you can customize the appearance of the splitter lines.

Context Menu

You can set the context menu (right-click menu) on the grid as well as on the cell editing control in the GcMultiRow control.

Grid Context Menu

Assign the context menu to be displayed on the grid with the GcMultiRow.ContextMenuStrip property or the GcMultiRow.ContextMenuStrip property. If a menu has been assigned to both properties, the GcMultiRow.ContextMenuStrip property has priority.

Using Code

This example sets the context menu.

[VB]
Dim menu As ContextMenuStrip = New ContextMenuStrip()
menu.Items.AddRange(New ToolStripItem() { _
    New ToolStripMenuItem("Test 1"), New ToolStripMenuItem("Test 2") })
GcMultiRow1.ContextMenuStrip = menu

[CS]
ContextMenu menu = new ContextMenuStrip();
menu.Items.AddRange(new ToolStripItem[] { _
    new ToolStripMenuItem("Test 1"), new ToolStripMenuItem("Test 2") });
gcMultiRow1.ContextMenuStrip = menu.
By default, the context menu can be displayed by right clicking on it with the mouse, or by using the application key. The application key can be changed using the `Shift + F10` key.

### Mouse Cursor Object Type

Use the `GcMultiRow.HitTest` method (in the on-line documentation) method to find the type of object being pointed to by the mouse. For example, you can have different actions when the mouse points to the header and when the mouse points to the cell.

#### Using Code

The following code changes the background color of a cell in a row when the mouse button is clicked on that cell.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow
Private Sub GcMultiRow1_MouseDown(ByVal sender As System.Object, ByVal e As System.Windows.Forms.MouseEventArgs) Handles GcMultiRow1.MouseDown
    Dim gcMultiRow As GcMultiRow = TryCast(sender, GcMultiRow)
    Dim hitTestInfo As HitTestInfo = gcMultiRow.HitTest(e.X, e.Y)
    If hitTestInfo.Type = HitTestType.Row Then
        If hitTestInfo.CellIndex > -1 Then
        End If
    End If
End Sub
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;
private void gcMultiRow1_MouseDown(object sender, MouseEventArgs e)
{
    GcMultiRow gcMultiRow = sender as GcMultiRow;
    HitTestInfo hitTestInfo = gcMultiRow.HitTest(e.X, e.Y);
    if (hitTestInfo.Type == HitTestType.Row)
    {
        if (hitTestInfo.CellIndex > -1)
        {
        }
    }
}
```

### Row Context Menu

The context menu can be set at the row level as well.

#### Using Code

The following code assigns the context menu to the first row. This menu will be displayed on places other than the cells, or when the context menu has not been specified for the cells.

**[VB]**

```vbnet
Dim menu As ContextMenuStrip = New ContextMenuStrip()
menu.Items.AddRange(New ToolStripItem() { New ToolStripMenuItem("Test 1"), New ToolStripMenuItem("Test 2") })
GcMultiRow1.Rows(0).ContextMenuStrip = menu
```

**[CS]**

```csharp
ContextMenuStrip menu = new ContextMenuStrip();
menu.Items.AddRange(new ToolStripItem[] { new ToolStripMenuItem("Test 1"), new ToolStripMenuItem("Test 2") });
gcMultiRow1.Rows[0].ContextMenuStrip = menu;
```

### Cell Context Menu

The context menu can be specified for each cell by using the Cell.ContextMenuStrip property.

#### Using Code

The following code describes how to set the context menu for the first cell of the first row.

**[VB]**

```vbnet
Dim menu As ContextMenuStrip = New ContextMenuStrip()
menu.Items.AddRange(New ToolStripItem() { New ToolStripMenuItem("Test 1"), New ToolStripMenuItem("Test 2") })
GcMultiRow1.Rows(0).Cells(0).ContextMenuStrip = menu
```

**[CS]**

```csharp
ContextMenuStrip menu = new ContextMenuStrip();
menu.Items.AddRange(new ToolStripItem[] { new ToolStripMenuItem("Test 1"), new ToolStripMenuItem("Test 2") });
gcMultiRow1.Rows[0].Cells[0].ContextMenuStrip = menu;
```

### Cell Editing Control Context Menu

The context menu can be displayed while editing the cell if the cell editing control supports the context menu.

#### Using Code

This example illustrates the cell displaying the context menu while being edited.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow
Private Sub GcMultiRow1_EditingControlShowing(ByVal sender As System.Object, ByVal e As GrapeCity.Win.MultiRow.EditingControlShowingEventArgs) Handles GcMultiRow1.EditingControlShowing
    Dim textBoxEditingControl As TextBoxEditingControl = DirectCast(e.Control, TextBoxEditingControl)
    If textBoxEditingControl IsNot Nothing Then
        Dim menu As ContextMenuStrip = New ContextMenuStrip()
        menu.Items.AddRange(New ToolStripItem() { New ToolStripMenuItem("Test 1"), New ToolStripMenuItem("Test 2") })
        textBoxEditingControl.ContextMenuStrip = menu
    End If
End Sub
```

**[CS]**

```csharp
using GrapeCity.WinForms;
private void gcMultiRow1_EditingControlShowing(
	object sender, EditingControlShowingEventArgs e)
{
    TextBoxEditingControl textBoxEditingControl = e.Control as TextBoxEditingControl;
    ContextMenuStrip menu = new ContextMenuStrip();
    menu.Items.AddRange(new ToolStripItem[] { new ToolStripMenuItem("Test 1"), new ToolStripMenuItem("Test 2") });
    textBoxEditingControl.ContextMenuStrip = menu;
}
```
Shortcut Keys

You can use shortcut keys to work with the GcMultiRow control using the keyboard.

Default Shortcut Keys

The default shortcut keys for the grid vary depending upon the view mode. The following table shows the actions versus the allocation of keys in various modes. Please refer to View Modes for details on the grid view modes.

<table>
<thead>
<tr>
<th>Group or Actions</th>
<th>Description</th>
<th>Default Mode</th>
<th>Row Mode</th>
<th>Display Mode</th>
<th>Listbox Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SelectionActions</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MoveDown</td>
<td>Move to the cell below.</td>
<td>Keys.Down</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MoveUp</td>
<td>Move to the cell above.</td>
<td>Keys.Up</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MoveLeft</td>
<td>Move to the cell on the left.</td>
<td>Keys.Left</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MoveRight</td>
<td>Move to the cell on the right.</td>
<td>Keys.Right</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MoveToFirstCell</td>
<td>Move to the first cell of the first row.</td>
<td>Keys.Control + Keys.Home</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MoveToLastCell</td>
<td>Move to the last cell of the last row.</td>
<td>Keys.Control + Keys.End</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MoveToNextCell</td>
<td>Move to the next cell (regardless of the row).</td>
<td>Keys.Tab OR Keys.Control + Keys.Tab</td>
<td>Keys.Tab OR Keys.Control + Keys.Tab</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MoveToFirstChildInRowByTabOrder</td>
<td>Move to the first cell in the tab order.</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MoveToFirstChildInRowByTabOrder</td>
<td>Move to the first cell in the tab order of the current row.</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MoveToLastCellInRow</td>
<td>Move to the last cell of the current row.</td>
<td>Keys.End OR Keys.Control + Keys.Right</td>
<td>Keys.End OR Keys.Control + Keys.Right</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MoveToLastCellInRowByTabOrder</td>
<td>Move to the last cell in the tab order.</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MoveToPreviousCellInRow</td>
<td>Move to the next cell in the tab order of the current row.</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MoveToNextCellInRow</td>
<td>Move to the next cell in the current row.</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MoveToPreviousRow</td>
<td>Move to the previous row.</td>
<td>-</td>
<td>Keys.Up OR Keys.Left</td>
<td>Keys.Up</td>
<td>-</td>
</tr>
<tr>
<td>MoveToNextRow</td>
<td>Move to the next row.</td>
<td>-</td>
<td>Keys.Down OR Keys.Left</td>
<td>Keys.Down</td>
<td>-</td>
</tr>
<tr>
<td>MoveToPreviousPage</td>
<td>Move to the previous page.</td>
<td>Keys.PageUp</td>
<td>Keys.PageUp</td>
<td>Keys.PageUp</td>
<td>-</td>
</tr>
<tr>
<td>ReverseSelectCurrentRow</td>
<td>Reverse the selection of the present row.</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Keys.Space</td>
</tr>
<tr>
<td>SelectRow</td>
<td>Select the current row.</td>
<td>Keys.Shift + Keys.Space</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ShiftDown</td>
<td>Expand or shorten the selected range into the closest cells below.</td>
<td>Keys.Shift + Keys.Down</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ShiftUp</td>
<td>Expand or shorten the selected range into the closest cells above.</td>
<td>Keys.Shift + Keys.Up</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ShiftLeft</td>
<td>Expand or shorten the selected range into the closest cells on the left.</td>
<td>Keys.Shift + Keys.Left</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
ShiftRight  Expand or shorten the selected range into the closest cells on the right.  Keys.Shift + Keys.Right  -  -  -  
ShiftToFirstCell  Change the currently selected range from the present cells to the first cell of the first active row.  Keys.Shift + Keys.Control + Keys.Home  -  -  -  
ShiftToLastCell  Change the currently selected range from the present cells to the last cell of the last active row.  Keys.Shift + Keys.Control + Keys.End  -  -  -  
ShiftToFirstCellInRow  Change the currently selected range from the present cells to the first cell of the present row.  Keys.Shift + Keys.Home OR Keys.Shift + Keys.Control + Keys.Left  -  -  -  
ShiftToLastCellInRow  Change the currently selected range from the present cells to the last cell of the present row.  Keys.Shift + Keys.End OR Keys.Shift + Keys.Control + Keys.Right  -  -  -  
ShiftToFirstRow  Change the currently selected range from the present cells to the cell of the first row having the same CellIndex.  Keys.Shift + Keys.Control + Keys.Up OR Keys.Shift + Keys.Home  -  -  -  
ShiftToLastRow  Change the currently selected range from the present cells to the cell of the last row having the same CellIndex.  Keys.Shift + Keys.End OR Keys.Shift + Keys.Control + Keys.Down  -  -  -  
ShiftToPreviousRow  Change the currently selected range from the present cells to the cell of the previous row having the same CellIndex.  -  Keys.Shift + Keys.Up OR Keys.Shift + Keys.Left  -  -  
ShiftToNextRow  Change the currently selected range from the present cells to the cell of the next row having the same CellIndex.  -  Keys.Shift + Keys.Down OR Keys.Shift + Keys.Right  -  -  -  
ShiftPageDown  Expand or shorten the selected range to the next page.  Keys.Shift + Keys.PageDown OR Keys.Shift + Keys.Up  -  -  -  
ShiftPageUp  Expand or shorten the selected range to the previous page.  Keys.Shift + Keys.PageUp OR Keys.Shift + Keys.Down  -  -  -  
ComponentActions  
SelectNextControl  Shift the focus to the next control.  -  -  -  -  
SelectPreviousControl  Shift the focus to the previous control.  -  -  -  -  
EditingActions  
BeginEdit  Start editing of the cells.  Keys.F2 OR Keys.Enter  Keys.F2 OR Keys.Enter  -  -  
EndEdit  Confirm the editing.  Keys.Enter  Keys.Enter  -  -  
CancelCellEdit  Cancel input and exit edit mode when the current cell is in edit mode.  -  -  -  -  
CancelEdit  Cancel the cell editing.  Keys.Escape  Keys.Escape  -  -  
CancelRowEdit  Cancel Row level editing if current cell is not in edit mode, or if the Dirty state of the cell is set to false.  -  -  -  -  
Clear  Delete the values of the selected cells.  Keys.Delete  Keys.Delete  -  -  
CommitRow  Commit the row changes.  Keys.Control + Keys.Enter  Keys.Control + Keys.Enter  -  -  
ScrollActions  
VerticalScrollToFirstPage  Move to the first page in the vertical direction.  -  -  -  Keys.Control + Keys.Up
VerticalScrollToLastPage Move to the last page in the vertical direction. - - Keys.Control + Keys.Down
VerticalScrollToPreviousPage Move to the previous page in the vertical direction. - - Keys.PageUp OR Keys.Shift + Keys.Space
VerticalScrollToNextPage Move to the next page in the vertical direction. - - Keys.PageDown OR Keys.Space
HorizontalScrollToFirstPage Move to the first page in the horizontal direction. - - Keys.Home OR Keys.Control + Keys.Left
HorizontalScrollToLastPage Move to the last page in the horizontal direction. - - Keys.End OR Keys.Control + Keys.Right
HorizontalScrollToPreviousPage Move to the previous page in the horizontal direction. - - Keys.Left
HorizontalScrollToNextPage Move to the next page in the horizontal direction. - - Keys.Right
ScrollUp Move one step up in the vertical direction. - - Keys.Up
ScrollDown Move one step down in the vertical direction. - - Keys.Down
ScrollLeft Move one step left in the horizontal direction. - - Keys.Left
ScrollRight Move one step right in the horizontal direction. - - Keys.Right

The list of default shortcut keys can be retrieved using the following properties of the ShortcutKeyManager.

Using Code

The following code gets the shortcut keys in default mode using the ShortcutKeyManager.DefaultModeList ('DefaultModeList Property' in the on-line documentation) property.

[VB]

Imports GrapeCity.Win.MultiRow

' Place the ListView control onto the form
ListView1.SuspendLayout()
ListView1.View = View.Details
ListView1.Columns.Add("Action")
ListView1.Columns.Add("Key")
For Each shortcutKey As ShortcutKey In GcMultiRow1.ShortcutKeyManager.DefaultModeList
    Dim item As ListViewItem = New ListViewItem()
    item.Text = shortcutKey.Action.DisplayName
    item.SubItems.Add(shortcutKey.Key.ToString())
    ListView1.Items.Add(item)
Next
ListView1.ResumeLayout()

[CS]

using GrapeCity.Win.MultiRow;

// Place the ListView control onto the form
listView1.SuspendLayout();
listView1.View = View.Details;
listView1.Columns.Add("Action");
listView1.Columns.Add("Key");
foreach (ShortcutKey shortcutKey in gcMultiRow1ShortcutKeyManager.DefaultModeList)
{
    ListViewItem item = new ListViewItem();
    item.Text = shortcutKey.Action.DisplayName;
    item.SubItems.Add(shortcutKey.Key.ToString());
    listView1.Items.Add(item);
}
listView1.ResumeLayout();

The shortcut keys in row mode can be retrieved using the ShortcutKeyManager.RowModeList ('RowModeList Property' in the on-line documentation) property and those in display mode can be retrieved using the ShortcutKeyManager.DisplayModeList ('DisplayModeList Property' in the on-line documentation) property.

Registering Shortcut Keys

Use the ShortcutKeyManager.Register ('Register Method' in the on-line documentation) method to register the shortcut keys.
Using Code
The following code starts the editing of the current cell when the user presses the F3 key.

[VB]

[CS]
gcMultiRow1ShortcutKeyManager.Register(GrapeCity.Win.MultiRow.EditingActions.BeginEdit, Keys.F3);

Un-register the Shortcut Keys
Use the ShortcutKeyManager.Unregister ('Unregister Method' in the on-line documentation) method to unregister the already registered keys. The shortcut keys can be deleted by deleting the action or by deleting a key.

Using Code
The following code deletes the shortcut key (Ctrl+A) which by default is assigned to the SelectAll action.

[VB]
GcMultiRow1ShortcutKeyManager.Unregister(GrapeCity.Win.MultiRow.SelectionActions.SelectAll)

[CS]
gcMultiRow1ShortcutKeyManager.Unregister(GrapeCity.Win.MultiRow.SelectionActions.SelectAll);

The following code deletes all the actions assigned to the Tab key.

[VB]
GcMultiRow1.ShortcutKeyManager.Unregister(Keys.Tab)

[CS]
gcMultiRow1.ShortcutKeyManager.Unregister(Keys.Tab);

Creating Actions
You can create actions and assign them to the shortcut keys. To create actions for shortcut keys, create a class that implements the IAction interface.

Using Code
The following code shows the current cell's information when the user presses the F3 key.

[VB]
Imports GrapeCity.Win.MultiRow
Public Class MyAction
    Implements IAction
    Public Function CanExecute(ByVal target As GcMultiRow) As Boolean Implements IAction.CanExecute
        Return True
    End Function
    Public ReadOnly Property DisplayName() As String Implements IAction.DisplayName
        Get
            Return Me.ToString()
        End Get
    End Property
    Public Sub Execute(ByVal target As GcMultiRow) Implements IAction.Execute
        MessageBox.Show(target.CurrentCellPosition.ToString())
    End Sub
    End Class
GcMultiRow1.ShortcutKeyManager.Register(New MyAction(), Keys.F3)

[CS]
using GrapeCity.Win.MultiRow;
public class MyAction : IAction
{    
    public bool CanExecute(GcMultiRow target)
    {
        return true;
    }

    public void Execute(GcMultiRow target)
    {
        MessageBox.Show(target.CurrentCellPosition.ToString());
    }
}
public string DisplayName
{
    get { return this.ToString(); }
}

public void Execute(GcMultiRow target)
{
    MessageBox.Show(target.CurrentCellPosition.ToString());
}

goMultiRow1ShortcutKeyManager.Register(new MyAction(), Keys.F3);

Executing Actions
You can also directly execute the actions of the shortcut keys. The execution of the action is treated as a user operation.

Using Code
The following code selects all the cells of the GcMultiRow control.

[VB]
GrapeCity.Win.MultiRow.SelectionActions.SelectAll.Execute(GcMultiRow1)

[CS]
GrapeCity.Win.MultiRow.SelectionActions.SelectAll.Execute(gcMultiRow1);

Shortcut Key Settings Dialog
You can change the settings for the shortcut keys for each mode in the dialog shown below, by pressing the ellipsis in the GcMultiRow.ShortcutKeyManager ('ShortcutKeyManager Property' in the on-line documentation) property in the Properties window.

You can set user-defined custom actions using the Shortcut Keys setting dialog. Use the following steps to set user-defined custom actions.

1. Create a class that implements the IAction interface and add it to the project.
2. Build the project.
3. Select the GcMultiRow.ShortcutKeyManager property from the Properties window, and click the ... button.
4. Select the key to which you wish to add the user-defined custom action, from the Shortcut Keys Setting dialog being displayed.
5. Select Custom Actions from the Types: Action combobox.
6. Select the Action from the Action list, and click the Add button.
7. Click the OK button and close the Shortcut Keys Setting dialog.

Setting a Default Shortcut Key to the User-defined GcMultiRow Control
You can assign your own default shortcut keys for the user-defined default GcMultiRow control using the InitialShortcutKeyManager ('InitialShortcutKeyManager Property' in the on-line documentation) property.

Attention
- If a shortcut is set using the InitialShortcutKeyManager property, then even if you set the shortcut key using the GcMultiRow.ShortcutKeyManager property later, that setting will not be valid.
- If the GcMultiRowShortcutKeyManager property is set prior to the setting of the InitialShortcutKeyManager property, the InitialShortcutKeyManager property setting will not be valid.

Using Code
The following code changes the default shortcut key for the right key of the user-defined GcMultiRow control.
Imports System.Windows.Forms
Imports GrapeCity.Win.MultiRow
Public Class MyGcMultiRow
    Inherits GcMultiRow
    Public Sub New()
        InitializeCustomGrid()
    End Sub
    Private Sub InitializeCustomGrid()
    End Sub
End Class
GcMultiRow1.CurrentCell.HorizontalResize(10)
GcMultiRow1.CurrentCell.VerticalResize(10)
End If

[CS]

using GrapeCity.Win.MultiRow;

if (gcMultiRow1.CurrentCell != null)
{
    gcMultiRow1.CurrentCell.HorizontalResize(10);
    gcMultiRow1.CurrentCell.VerticalResize(10);
}

Use the return values of the Cell.Width ('Width Property' in the on-line documentation) property and the Cell.Height ('Height Property' in the on-line documentation) property to specify the size of the cell absolutely.

Using Code

The following sample creates an offset based on the width of the cell.

[VB]

Imports GrapeCity.Win.MultiRow

Dim targetCell As Cell = GcMultiRow1.CurrentCell
If targetCell IsNot Nothing Then
    Dim newWidth As Integer = 300
    Dim offset As Integer = newWidth - targetCell.Width
    targetCell.HorizontalResize(offset)
End If

[CS]

using GrapeCity.Win.MultiRow;

Cell targetCell = gcMultiRow1.CurrentCell;
if (targetCell != null)
{
    int newWidth = 300;
    int offset = newWidth - targetCell.Width;
    targetCell.HorizontalResize(offset);
}

Retrieving the Method Used to Move Cells

In the GcMultiRow control, use the MoveStatus property to get the action used to move cells.

Target Events

The MoveStatus property can be used in the following events.

- GcMultiRow.CellEnter ('CellEnter Event' in the on-line documentation) Event
- GcMultiRow.CellLeave ('CellLeave Event' in the on-line documentation) Event
- GcMultiRow.RowEnter ('RowEnter Event' in the on-line documentation) Event
- GcMultiRow.RowLeave ('RowLeave Event' in the on-line documentation) Event

Values that can be Retrieved

Values that can be retrieved using the MoveStatus property are members of the MoveStatus ('MoveStatus Enumeration' in the on-line documentation) enumeration.

If you use the SelectionActions ('SelectionActions Class' in the on-line documentation) action to move the current cell, the value corresponding to the action is retrieved. Also, if you use the mouse to move the current cell, the MoveStatus property retrieves MouseClick. If you move the current cell by some other operation, the MoveStatus property retrieves NoAction.
Usage in the CellEnter, CellLeave, RowEnter, and RowLeave Events

You need to use the `CellMoveEventArgs` class in order to use `MoveStatus` in the CellEnter, CellLeave, RowEnter, and RowLeave events.

**Using Code**

The following code uses the CellEnter event.

**[VB]**
```vbnet
Imports GrapeCity.Win.MultiRow
Private Sub GcMultiRow1_CellEnter(sender As Object, e As CellEventArgs) Handles GcMultiRow1.CellEnter
    Dim newArgs As CellMoveEventArgs = DirectCast(e, CellMoveEventArgs)
    If newArgs.MoveStatus = MoveStatus.MouseClick Then
        Console.WriteLine(GcMultiRow1.CurrentCellPosition)
    End If
End Sub
```

**[CS]**
```csharp
using GrapeCity.Win.MultiRow;
private void gcMultiRow1_CellEnter(object sender, CellEventArgs e)
{
    CellMoveEventArgs newArgs = e as CellMoveEventArgs;
    if (newArgs.MoveStatus == MoveStatus.MouseClick)
    {
        Console.WriteLine(gcMultiRow1.CurrentCellPosition);
    }
}
```

Usage in the CellBeginEdit Event

In the CellBeginEdit event, use the `CellBeginEditEventArgs` class to get the `MoveStatus` property.

You need to set the GcMultiRow.EditMode property to EditOnEnter, in order to use MoveStatus in the CellBeginEdit event. If the GcMultiRowEditMode property is not set to EditOnEnter, MoveStatus always gets NoAction.

Usage in the CellEndEdit Event

In the CellEndEdit event, use the `CellEndEditEventArgs` class to get the `MoveStatus` property. If you want to use MoveStatus in the CellEndEdit event, the current cell must be in edit mode.

Usage in the NewCellPositionNeeded Event

In the NewCellPositionNeeded event, use the NewCellPositionNeededEventArgs class to get the MoveStatus property.

In the NewCellPositionNeeded event, you can use NewCellPosition property in combination with another operation to change the position when moving the current cell.

**Using Code**

The following code prevents moving a cell with a mouse click.

**[VB]**
```vbnet
Imports GrapeCity.Win.MultiRow
    If e.MoveStatus = MoveStatus.MouseClick Then
    End If
End Sub
```

**[CS]**
```csharp
using GrapeCity.Win.MultiRow;
private void gcMultiRow1_NewCellPositionNeeded(object sender, NewCellPositionNeededEventArgs e)
{
    if (e.MoveStatus == MoveStatus.MouseClick)
    {
    }
}
```

Automatic Merging of Cells

The GcMultiRow control provides a feature to perform automatic merging of cells, when the values of the cells placed in the row are the same in vertically adjacent rows.

**Conditions for Merging**

The following conditions must be met for the cells to be merged.

- Cells are placed in the Row section of the template.
- The upper and lower ends of the cells are in contact, in the Row section.
- The Cell.Mergeable ('Mergeable Property' in the on-line documentation) property is set to True.
- The cell values are not empty (Is not String.Empty or Null).
- The Value ('Value Property' in the on-line documentation) property of the vertically adjacent cell is set to the same value.
- The Visible ('Visible Property' in the on-line documentation) property of the cell is set to True.
In the following figure, the cells other than Cell1, do not meet the conditions of merging.

Cell Style
The style that is set for the top-most cell, is applied to the cells that have been merged.

Selection and Editing of Cells
If the GcMultiRow.MergedCellsSelectionMode ("MergedCellsSelectionMode Property" in the on-line documentation) property is set to All, all the merged cells are selected. In this case, if you edit the cells, all the merged cells are changed to the same value. If the GcMultiRow.MergedCellsSelectionMode property is set to Individually, the merged cells are selected one by one. In this case, you can change the value of each cell individually.

Validation of Cells
If a validator is set in the merged cell, then the validator and the validation action is performed for the top-most cell, but the GcMultiRow.CellValidating ("CellValidating Event" in the on-line documentation) event is fired for the current cell; therefore, if you want to use the automatic merge feature of the cells, it is recommended that you do not set different validators for the cells.

Specifying the Vertical Relationship
If cells that meet the merging conditions are present in the GcMultiRow control, the GcMultiRow.QueryCellMergeState ("QueryCellMergeState Event" in the on-line documentation) event is fired. You can use this event to specify the vertical relationship for multiple cells, and restrict the merging of cells if the value of the cell above is different from that of the cell below.

Using Code
This example merges cells.

[VB]
Imports GrapeCity.WinForms.MultiRow
Private Sub GcMultiRow1_QueryCellMergeState(sender As Object, e As QueryCellMergeStateEventArgs) Handles GcMultiRow1.QueryCellMergeState

Copyright © GrapeCity, inc. All rights reserved.
If e.ShouldMerge = True Then
If Not e.QueryCell.CellIndex = 0 Then
    Dim newQueryCell As CellPosition = New CellPosition(e.QueryCell.RowIndex, e.QueryCell.CellIndex - 1)
    Dim newTargetCell As CellPosition = New CellPosition(e.TargetCell.RowIndex, e.TargetCell.CellIndex - 1)
    e.ShouldMerge = Me.GcMultiRow1.IsMerged(newQueryCell, newTargetCell)
End If
End If
End Sub

**Limitations**

- If the GcMultiRow.MergedCellsSelectionMode property is set to All, the behavior is as follows.
  - Even if a merged cell is selected, only one cell is included in the GcMultiRow.SelectedCells (SelectedCells Property in the on-line documentation) property.
  - Events related to the cell occur for the currently selected cell.
- If the GcMultiRow.AllowUserToReverseSelect (AllowUserToReverseSelect Property in the on-line documentation) property is set to True, there are times when the behavior where the selection state of already selected cells is reversed by the Ctrl key + mouse click, and does not work correctly. For example, if cells of the first and second row are merged, after you have selected a cell in the first row using a mouse click, the selection state of the cell is not canceled, even if you press Ctrl key + mouse click on a cell in the second row.
  - Gradation of the background color, and the top and bottom borders set in the Row section, is not applied to the merged cells.
  - In case of resizing the height of the Row section of the merged cells, it is not possible to resize multiple Row sections together.

**Displaying a New Row at the Top**

In the GcMultiRow control, you can specify whether to display a new row at the top, or at the bottom of the grid.

**Specifying the Display Position**

Use the GcMultiRow.NewRowPosition (NewRowPosition Property in the on-line documentation) property to specify the display position of the new row. To display the new row at the top of the grid, set the GcMultiRow.NewRowPosition property to Top.

**Using Code**

This example sets the NewRowPosition property.

**[VB]**


**[CS]**


**New Row Separator**

A separator is displayed to separate the new row from the other rows, when you display the new row at the top of the grid. Use the GcMultiRow.NewRowSeparatorStyle (NewRowSeparatorStyle Property in the on-line documentation) property to set the style of the separator.

![Separator](image)

The Separator feature is only available to display the new row at the top of the grid.

**Using Code**

This example sets the new row separator.

**[VB]**


**[CS]**


**Alternating Display of a New Row**

You can display alternate text instead of a new row, by using the GcMultiRow.NewRowAlternateText (NewRowAlternateText Property in the on-line documentation) property. Set the GcMultiRow.NewRowAlternateText.Visible (Visible Property in the on-line documentation) property to True, to display alternate text.

**Using Code**

This example displays alternate text.

**[VB]**

GcMultiRow1.NewRowAlternateText.Visible = True
Expanding Or Collapsing Columns

The GcMultiRow control provides the functionality to expand or collapse columns.

Using Code

You can expand or collapse columns using the Expand ('Expand Method' in the on-line documentation) method or the Collapse ('Collapse Method' in the on-line documentation) method. In addition, it is possible to get the expanded or collapsed state of the column, by using the Iscollapsed ('IsCollapsed Property' in the on-line documentation) property.
private void Form1_Load(object sender, EventArgs e)
{
    TextBoxCell textBoxCell1 = new TextBoxCell();
textBoxCell1.Name = "textBoxCell1";
TextBoxCell textBoxCell2 = new TextBoxCell();
textBoxCell2.Name = "textBoxCell2";
TextBoxCell textBoxCell3 = new TextBoxCell();
textBoxCell3.Name = "textBoxCell3";
gcMultiRow1.Template = Template.CreateGridTemplate(new Cell[] { textBoxCell1, textBoxCell2, textBoxCell3 });
}
private void button1_Click(object sender, EventArgs e)
{
    if (gcMultiRow1.Columns[1].IsCollapsed == true)
    {
        gcMultiRow1.Columns[1].Expand();
    } else {
        gcMultiRow1.Columns[1].Collapse();
    }
}

If the Collapse method is executed, it works as follows.

- A column that is collapsed using the Collapse method cannot be resized with the mouse. You need to use the Expand method to expand a column.
- The Cell.Visible (Visible Property in the on-line documentation) property of the cell for which the Collapse method is executed, is set to False.
- The cell for which the Collapse method is executed, cannot be selected by mouse operation or shortcut keys.
- Clipboard operations (cut, copy, paste) cannot be performed for the value of the cell for which the Collapse method is executed.

Automatic Adjustment of the Cell Width

In the GcMultiRow control, you can adjust the width of the cell to fit the cell contents.

Automatic Adjustment based on the Cell Contents

You can use the GcMultiRow.HoriztonalAutoSizeMode (HorizontalAutoSizeMode Property in the on-line documentation) property to automatically adjust the width of the cells placed in the column header section, column footer section, or row, to fit the cell contents.

- Automatic adjustment of the column width, by double-clicking the right edge of the header, does not work if the GcMultiRow.HoriztonalAutoSizeMode (HorizontalAutoSizeMode Property in the on-line documentation) property is set to a value other than None.
- If you run the Cell.PerformHorizontalAutoFit (PerformHorizontalAutoFit Method in the on-line documentation) method or the Cell.HoriztonalResize (HorizontalResize Method in the on-line documentation) method, in a state when the GcMultiRow.HoriztonalAutoSizeMode property is set to a value other than None, the System.InvalidOperationException exception occurs.

Using Code

The following code automatically adjusts the width of the cells placed in the column header section, to fit the cell contents.

[VB]
Imports GrapeCity.Win.MultiRow
Dim textBoxCell1 As New TextBoxCell()
textBoxCell1.Name = "textBoxCell1"
Dim textBoxCell2 As New TextBoxCell()
textBoxCell2.Name = "textBoxCell2"
Dim template1 As Template = Template.CreateGridTemplate(New Cell() { textBoxCell1, textBoxCell2 })
template1.ColumnHeaders(0).Cells("columnHeaderCell1").Value = "Column1"
template1.ColumnHeaders(0).Cells("columnHeaderCell2").Value = "Column header Cell 2"
gcMultiRow1.Template = template1
gcMultiRow1.RowCount = 3
gcMultiRow1.SetValue(0, "textBoxCell1", "ABC")

cgMultiRow1.SetValue(0, "textBoxCell1", "MultiRow for Windows Forms 7.0")

[CS]
using GrapeCity.Win.MultiRow;
TextBoxCell textBoxCell1 = new TextBoxCell();
textBoxCell1.Name = "textBoxCell1";
TextBoxCell textBoxCell2 = new TextBoxCell();
textBoxCell2.Name = "textBoxCell2";
Template template1 = Template.CreateGridTemplate(new Cell[] { textBoxCell1, textBoxCell2 });
template1.ColumnHeaders(0).Cells("columnHeaderCell1").Value = "Column1"
template1.ColumnHeaders(0).Cells("columnHeaderCell2").Value = "Column header Cell 2";
gcMultiRow1.Template = template1;
gcMultiRow1.RowCount = 3;
gcMultiRow1.SetValue(0, "textBoxCell1", "ABC");
gcMultiRow1.SetValue(0, "textBoxCell1", "MultiRow for Windows Forms 7.0");
gcMultiRow1.HorizontalAlignmentMode = HorizontalAutoSizeMode.CellsInColumnHeader;

Automatically Adjusting the Column Width of Multiple Columns

If you double-click the right edge of the header, when multiple columns are selected, you can automatically adjust the width of each cell to fit the respective cell contents. There are no additional settings required in order to achieve this behavior.

If you set the Template.LayoutMode (LayoutMode Property in the on-line documentation) property to LeftToRight, and use the Column mode, the width of the row is automatically adjusted, so the behavior, in appearance, seems to be the same as when the Template.LayoutMode property is set to TopToBottom.
Disabling Automatic Adjustment

Set the AllowUserToAutoFitColumns (‘AllowUserToAutoFitColumns Property’ in the on-line documentation) property to False to disable the automatic adjustment of the column width, when you double-click the right edge of the header.

Using Code

This example sets the AllowUserToAutoFitColumns property.

[VB]
GcMultiRow1.AllowUserToAutoFitColumns = False

[CS]
gcMultiRow1.AllowUserToAddRows = false;

Customization of Automatic Adjustment of Width

Use the GcMultiRow.CellAutoFitPreferredSizeNeeded (‘CellAutoFitPreferredSizeNeeded Event’ in the on-line documentation) event to customize the cell width that is automatically adjusted, when you double-click the right edge of the header.

Using Code

The following code automatically adjusts the column width to 100, when you double-click the right edge of the header.

[VB]
Imports GrapeCity.Win.MultiRow
Private Sub Form1_Load(sender As System.Object, e As System.EventArgs) Handles MyBase.Load
Dim textBoxCell1 As New TextBoxCell()
textBoxCell1.Name = "textBoxCell1"
Dim template1 As Template = Template.CreateGridTemplate(New Cell() {textBoxCell1})
GcMultiRow1.Template = template1
GcMultiRow1.RowCount = 3
GcMultiRow1.SetValue(0, "textBoxCell1", "ABC")
End Sub

Private Sub GcMultiRow1_CellAutoFitPreferredSizeNeeded(sender As Object, e As CellAutoFitPreferredSizeNeededEventArgs) Handles GcMultiRow1.CellAutoFitPreferredSizeNeeded
If e.Scope = CellScope.Row Then
Dim textBoxCell = DirectCast(GcMultiRow1.Rows(e.SectionIndex).Cells(e.CellIndex), TextBoxCell)
If textBoxCell IsNot Nothing AndAlso e.Direction = Orientation.Horizontal Then
e.PreferredSize = New Size(100, e.PreferredSize.Height)
End If
End If
End Sub

[CS]
using GrapeCity.Win.MultiRow;
private void Form1_Load(object sender, EventArgs e)
{
    TextBoxCell textBoxCell1 = new TextBoxCell();
textBoxCell1.Name = "textBoxCell1";
Template template1 = Template.CreateGridTemplate(new Cell[] { textBoxCell1 });
gcMultiRow1.Template = template1;
gcMultiRow1.RowCount = 3;
gcMultiRow1.SetValue(0, "textBoxCell1", "ABC");
}

private void gcMultiRow1_CellAutoFitPreferredSizeNeeded(object sender, CellAutoFitPreferredSizeNeededEventArgs e)
{
    if (e.Scope == CellScope.Row)
    {
        TextBoxCell textBoxCell = gcMultiRow1[e.SectionIndex, e.CellIndex] as TextBoxCell;
        if (textBoxCell != null && e.Direction == Orientation.Horizontal)
        {
            e.PreferredSize = new Size(100, e.PreferredSize.Height);
        }
    }
}

Points to Note with Default Settings

By default, GcMultiRow provides all the features that are normally provided by a grid control and other general applications. The following topics discuss how to limit the grid’s functions.

Adding Rows

Normally, the user can add rows by entering a new row. To disable this feature, set the GcMultiRow.AllowUserToAddRows (‘AllowUserToAddRows Property’ in the on-line documentation) property to False.

Using Code

This example shows how to prevent users from adding new rows when entering a new row.

[VB]
GcMultiRow1.AllowUserToAddRows = False

[CS]
gcMultiRow1.AllowUserToAddRows = false;

Deleting Rows


By default, the user can delete rows using the Ctrl + Delete keys. To disable this feature, set the GcMultiRow.AllowUserToDeleteRows ('AllowUserToDeleteRows Property' in the on-line documentation) property to False.

Using Code
This example prevents users from deleting rows.

[VB]
GcMultiRow1.AllowUserToDeleteRows = False

[CS]
gcMultiRow1.AllowUserToDeleteRows = false;

For details on the default shortcut keys, refer to Shortcut Keys.

Zoom
By default, the user can zoom in or out of the grid using the mouse or keyboard. To disable this feature, set the GcMultiRow.AllowUserToZoom ('AllowUserToZoom Property' in the on-line documentation) property to False.

Using Code
This example shows how to not allow users to zoom the grid.

[VB]
GcMultiRow1.AllowUserToZoom = False

[CS]
gcMultiRow1.AllowUserToZoom = false;

Clipboard Operations (Grid)
By default, the user can copy the cell values using the keyboard. To disable this feature, set the GcMultiRow.AllowClipboard ('AllowClipboard Property' in the on-line documentation) property to False.

Using Code
This example prevents users from copying cell values using the keyboard.

[VB]
GcMultiRow1.AllowClipboard = False

[CS]
gcMultiRow1.AllowUserToClipboard = false;

Clipboard Operations (Cell Editing Control)
By default, the cell editing control of each cell is provided with clipboard functionality. For example, clipboard operations are enabled while editing the text box cell, since it uses the TextBoxEditingControl. This is the same as the standard TextBox controls.

Using Code
This example disables this clipboard functionality.

[VB]
Imports GrapeCity.Win.MultiRow
Private Sub GcMultiRow1_EditingControlShowing(ByVal sender As System.Object, ByVal e As EditingControlShowingEventArgs) Handles GcMultiRow1.EditingControlShowing
  If TypeOf e.Control Is TextBoxEditingControl Then
    Dim textBoxEditingControl As TextBoxEditingControl = DirectCast(e.Control, TextBoxEditingControl)
    ' Disable the context menu by assigning an empty context menu
    textBoxEditingControl.ContextMenu = New ContextMenu()
    ' Disable the clipboard operations through keyboard
    AddHandler textBoxEditingControl.KeyDown, AddressOf Me.textBoxEditingControl_KeyDown
    RemoveHandler textBoxEditingControl.KeyDown, AddressOf Me.textBoxEditingControl_KeyDown
  End If
End Sub

Private Sub textBoxEditingControl_KeyDown(ByVal sender As System.Object, ByVal e As KeyEventArgs)
  Dim textBox As TextBox = DirectCast(sender, TextBox)
  If e.Modifiers = Keys.Control Then
    e.Handled = True
    textBox.SelectionLength = 0
  End If
End Sub

[CS]
using GrapeCity.Win.MultiRow;

private void Form1_Load(object sender, EventArgs e)
{
  gcMultiRow1.EditingControlShowing += new EditingControlShowingEventArgsDelegate(gcMultiRow1_EditingControlShowing);
}

private void gcMultiRow1_EditingControlShowing(object sender, EditingControlShowingEventArgs e)
{
  if (e.Control is TextBoxEditingControl)
  {
    TextBoxEditingControl textBoxEditingControl = e.Control as TextBoxEditingControl;
    // Disable the context menu by assigning an empty context menu
    textBoxEditingControl.ContextMenu = new ContextMenu();
    // Disable the clipboard operations through keyboard
    textBoxEditingControl.KeyDown += new KeyEventHandler(textBoxEditingControl_KeyDown);
    textBoxEditingControl.KeyDown -= new KeyEventHandler(textBoxEditingControl_KeyDown);
  }
}
private void textBoxEditingControl_KeyDown(object sender, KeyEventArgs e)
{
    TextBox textBox = sender as TextBox;
    if (e.Modifiers == Keys.Control)
    {
        e.Handled = true;
        textBox.SelectionLength = 0;
    }
}

Editing Cell values
User input is allowed by default in the editable cell types, for example, TextBoxCell ('TextBoxCell Class' in the on-line documentation) or DateTimePickerCell ('DateTimePickerCell Class' in the on-line documentation). To disable this feature, use either of the following.
    - Set the GcMultiRow.EditMode ('EditMode Property' in the on-line documentation) property to EditProgrammatically.
    - Set the Cell.ReadOnly ('ReadOnly Property' in the on-line documentation) property to True.

Cell Selection
By default, it is possible to select all the cells except the header cell and its inherited classes. To disable this functionality, you can use either of the following.
    - Set the GcMultiRow.ViewMode ('ViewMode Property' in the on-line documentation) property to Display.
    - Set the Cell.Selectable ('Selectable Property' in the on-line documentation) property to False.

Resizing Cells
By default, cells can be resized using the header cell and its inherited classes. To disable resizing by the user, set the GcMultiRow.AllowUserToResize ('AllowUserToResize Property' in the on-line documentation) property to False. To disable the resizing in individual cells, use the Cell.ResizeMode ('ResizeMode Property' in the on-line documentation) property. By default, resizing is allowed for each of the following cell types:
    - ColumnHeaderCell ('ColumnHeaderCell Class' in the on-line documentation)
    - CornerHeaderCell ('CornerHeaderCell Class' in the on-line documentation)
    - RowHeaderCell ('RowHeaderCell Class' in the on-line documentation)
    - HeaderCell ('HeaderCell Class' in the on-line documentation)

Input Null Values
By default, the user can input null (Nothing in Visual Basic) using Ctrl + 0. You can delete the corresponding shortcut keys to disable this feature.

Using Code
The following example prevents the user from inputting a null value using the Ctrl + 0 keys.

[VB]

[CS]
gcMultiRow1.ShortcutKeyManager.Unregister(GrapeCity.Win.MultiRow.EditingActions.InputNullValue);

Cell Autosize
The cell width is auto-adjusted based on its contents when you double-click the border of the column header cell. By default, the target cells are limited to the area that is displayed on the screen. To make all cells the target cells, change the GcMultiRow.AutoFitContent ('AutoFitContent Property' in the on-line documentation) property to All.

Using Code
The following example makes all the cells auto-adjust when the user double-clicks the border of the column header cell.

[VB]
GcMultiRow1.AutoFitContent = GrapeCity.Win.MultiRow.AutoFitContent.All

[CS]
gcMultiRow1.AutoFitContent = GrapeCity.Win.MultiRow.AutoFitContent.All;

When the control has a lot of data, the processing time increases depending on the amount of data.

New Features of 7.0
The new features of 7.0 that affect the compatibility with version 6.0, are disabled by default. These features include those which are generally expected as default behavior.
    - Remove the selection from the already selected cells using the Ctrl key and mouse click.
    - Continuously select cells using the Shift key, cursor key, and mouse operations.

Using Code
The following code enables these features.

[VB]
GcMultiRow1.AllowUserToReverseSelect = True
GcMultiRow1.AllowUserToShiftSelect = True

[CS]
gcMultiRow1.AllowUserToReverseSelect = true;
gcMultiRow1.AllowUserToShiftSelect = true;

For information about other new features of 7.0, refer to Feature Comparison.

Tips on Improving the Performance
This section introduces code examples to improve the speed and memory usage.
Suppress the Rendering of the Control

You can improve the performance by first suppressing the rendering before making any changes, and then by resuming the rendering after the changes are done. Use the following methods to suppress and resume the rendering.

- GcMultiRow.SuspendLayout Method (Control.SuspendLayout method)
- GcMultiRow.ResumeLayout Method (Control.ResumeLayout method)

These methods are the same as for the System.Windows.Forms.ListBox or System.Windows.Forms.DataGridView.

Using Code

The following code suppresses the rendering of the control.

[VB]

' Suppress the rendering of the control.
GcMultiRow1.SuspendLayout()

' TODO: Changes on GcMultiRow control

' Resume the rendering of the control
GcMultiRow1.ResumeLayout()

[CS]

// Suppress the rendering of the control
gcMultiRow1.SuspendLayout();

// TODO: Changes on GcMultiRow control

// Resume the rendering of the control
gcMultiRow1.ResumeLayout();

Initialization

To improve the performance, the control is placed without a default template. If a default template is required, you can select "Default Template" from the smart tag.

Access to the Cell Values

The GcMultiRow.GetValue ('GetValue Method' in the on-line documentation) method and the GcMultiRow.SetValue ('SetValue Method' in the on-line documentation) method provide an even more efficient way to access cell values than the Cell.Value ('Value Property' in the on-line documentation) property.

Using Code

The following code indexes the row and cell.

[VB]

GcMultiRow1.Rows(0).Cells(1).Value = 1234

[CS]

gcMultiRow1.Rows[0].Cells[1].Value = 1234;

Using Code

The following code provides direct access to the values inside the control.
The following code sets the value of all the cells to null (Nothing in Visual Basic), except for the header cells.

**VB**

```vbnet
Imports GrapeCity.Win.MultiRow

' When accessing different rows individually
Dim gcMultiRow As GcMultiRow = Me.GcMultiRow1
For rowIndex As Integer = 0 To gcMultiRow.RowCount - 1
    For cellIndex As Integer = 0 To gcMultiRow.Rows(rowIndex).Cells.Count - 1
        If Not TypeOf gcMultiRow(rowIndex, cellIndex) Is HeaderCell Then
            gcMultiRow.SetValue(rowIndex, cellIndex, Nothing)
        End If
    Next
Next

' When accessing the template rows
Dim gcMultiRow As GcMultiRow = Me.GcMultiRow1
For rowIndex As Integer = 0 To gcMultiRow.RowCount - 1
    For cellIndex As Integer = 0 To gcMultiRow.Template.Row.Cells.Count - 1
        If Not TypeOf gcMultiRow.Template.Row.Cells(cellIndex) Is HeaderCell Then
            gcMultiRow.SetValue(rowIndex, cellIndex, Nothing)
        End If
    Next
Next
```

**CS**

```csharp
using GrapeCity.Win.MultiRow;

// When accessing different rows individually
GcMultiRow gcMultiRow = this.gcMultiRow1;
```
Selection of Data Source
When you are handling large amounts of data, it is more efficient to use bound mode instead of unbound mode.

Cell Styles
The GcMultiRow.DefaultCellStyle ('DefaultCellStyle Property' in the on-line documentation) property is more efficient than the GcMultiRow.Rows[0].Cells[0].Style property when you have to apply the same style to all the cells.

Using Code
The following example illustrates using the DefaultCellStyle property.

[VB]
Imports GrapeCity.Win.MultiRow

' Change the back color of all the cells(1)
Dim gcMultiRow As GcMultiRow = Me.GcMultiRow1
For rowIndex As Integer = 0 To gcMultiRow.RowCount - 1
    For cellIndex As Integer = 0 To gcMultiRow.Rows(rowIndex).Cells.Count - 1
        gcMultiRow(rowIndex, cellIndex).Style.BackColor = Color.Azure
    Next
Next

' Change the back color of all the cells(2)

[CS]
using GrapeCity.Win.MultiRow;

// Change the back color of all the cells(1)
GcMultiRow gcMultiRow = this.gcMultiRow1;
for (int rowIndex = 0; rowIndex &lt; gcMultiRow.RowCount; rowIndex++)
{
    for (int cellIndex = 0; cellIndex &lt; gcMultiRow.Rows[rowIndex].Cells.Count; cellIndex++)
    {
        gcMultiRow[rowIndex, cellIndex].Style.BackColor = Color.Azure;
    }
}
Headers

This section provides information about using headers at runtime.

- Selecting Cells using Headers
- Filtering Rows using the Column Headers
- Sorting Rows using Column Headers
- Changing the Header Appearance
- Using Row Headers

Selecting Cells using Headers

Use the `HeaderCell.SelectionMode` property to select cells using headers. This property sets the area or range that is selected when the header is clicked. This property can be used in the `ColumnHeaderCell`, `RowHeaderCell`, or `CornerHeaderCell` sections which inherit from header cell.

If sorting using a header has been enabled for a header, you cannot select cells using this same header.

Select only Cells Inside the Header Area

You can specify that only cells within the header area are selected when the user clicks on the header if the `HeaderCell.SelectionMode` property has been set to `ContainedCells`.

The following image shows what happens when the user clicks on a column header, how the column header area is calculated, and how the cells are selected. Note that the cells lying partially inside the column header area are not the target for selection.

The calculation for the header area uses the cell width for column headers and cell height for row headers.

Select All Cells in the Header Area

All the cells included in the header area are selected when the user clicks on a header if the `HeaderCell.SelectionMode` property has been set to `IntersectedCells`.

The following image shows what happens when the user clicks on a column header, how the column header area is calculated, and how the cells are selected. Note that the cells lying partially inside the column header area are also a target for selection.
The calculation of the header area uses the cell width for column headers and cell height for row headers.

Select Rows

The entire row is selected when the user clicks on the header if the HeaderCell.SelectionMode property has been set to Row. You cannot specify this value in the column header cell.

Select All Rows

All the rows in the grid are selected when the user clicks on the header if the HeaderCell.SelectionMode property has been set to AllRows. You cannot specify this value in the row header cell. The default value of the SelectionMode property in the corner header cell is AllRows.

Disable Selection

The cells are not selected if the user clicks on the header when the HeaderCell.SelectionMode property has been set to None. This setting can be used when you want to sort using the column headers, disable the selection operation, or any other customizations.

Filtering Rows using the Column Headers

You can use the column header cell and commands in the drop-down list to filter rows.

Filter cannot be used in virtual mode.
New rows are not considered while filtering.

Setting the Drop-down list

You need to set the HeaderDropDownList class instance into the ColumnHeaderCell. DropDownList (‘DropDownList Property’ in the online documentation) property to set a drop-down list. Additionally, to enable the filter commands, you need to set the second argument of the constructor to True.
Using Code

This example creates a filter.

[VB]

Imports GrapeCity.Win.MultiRow
Dim template As Template = template.Default
Dim columnHeaderCell As ColumnHeaderCell = DirectCast(template.ColumnHeaders(0).Cells(0), ColumnHeaderCell)
columnHeaderCell.DropDownList = New HeaderDropDownList("textBoxCell1", True, False)
GcMultiRow1.Template = template

[CS]

using GrapeCity.Win.MultiRow;
Template template = Template.Default;
ColumnHeaderCell columnHeaderCell = template.ColumnHeaders[0].Cells[0] as ColumnHeaderCell;
columnHeaderCell.DropDownList = new HeaderDropDownList("textBoxCell1", true, false);
mcMultiRow1.Template = template;

Specifying the Filter Range

You can specify the range of rows to be filtered using the HeaderDropDownList.StartRow ("StartRow Property' in the on-line documentation) and HeaderDropDownList.EndRow ("EndRow Property' in the on-line documentation) properties.

Counting the Values

The first 1000 different values in the row filter range are counted by default. You can check this value using the DropDownAutoFilterItem.MaxCount ("MaxCount Property' in the on-line documentation) property. The DropDownAutoFilterItem ("DropDownAutoFilterItem Class' in the on-line documentation) class creates the list of filter candidates automatically, depending on the different values. You can improve the performance while displaying the drop-down, by reducing the number of values counted for filtering.

Using Code

The code shown below counts the first 100 different values.

[VB]

Imports GrapeCity.Win.MultiRow
Dim template As Template = template.Default
Dim columnHeaderCell As ColumnHeaderCell = DirectCast(template.ColumnHeaders(0).Cells(0), ColumnHeaderCell)
Dim headerDropDownList As HeaderDropDownList = New HeaderDropDownList("textBoxCell1", True, False)
For Each item As DropDownItem In headerDropDownList.Items
    If TypeOf item Is DropDownAutoFilterItem Then
        Dim dropDownAutoFilterItem As DropDownAutoFilterItem = DirectCast(item, DropDownAutoFilterItem)
dropDownAutoFilterItem.MaxCount = 100
    End If
Next
columnHeaderCell.DropDownList = headerDropDownList
GcMultiRow1.Template = template

' Set the sample data
GcMultiRow1.AllowUserToAddRows = False
GcMultiRow1.RowCount = 200
For i As Integer = 0 To GcMultiRow1.RowCount - 1
    GcMultiRow1(i, "textBoxCell1").Value = i.ToString()
using GrapeCity.Win.MultiRow;

Template template = Template.Default;
ColumnHeaderCell columnHeaderCell = template.ColumnHeaders[0].Cells[0] as ColumnHeaderCell;
HeaderDropDownList headerDropDownList = new HeaderDropDownList("textBoxCell1", true, false);
foreach (DropDownItem item in headerDropDownList.Items)
{
    if (item is DropDownAutoFilterItem)
    {
        DropDownAutoFilterItem dropDownAutoFilterItem = item as DropDownAutoFilterItem;
        dropDownAutoFilterItem.MaxCount = 100;
    }
}
columnHeaderCell DropDownList = headerDropDownList;
gcMultiRow1.Template = template;

// Set the sample data
gcMultiRow1.AllowUserToAddRows = false;
gcMultiRow1.RowCount = 200;
for (int i = 0; i < gcMultiRow1.RowCount; i++)
{
    gcMultiRow1[i, "textBoxCell1"].Value = i.ToString();
}

Clearing the Filter

You can use the GcMultiRow.ClearAllFilters (‘ClearAllFilters Method’ in the on-line documentation) method to collectively clear all the filters that are set.

Using Code

The following code uses the ClearAllFilters method.

[VB]

GcMultiRow1.ClearAllFilters()

[CS]

gcMultiRow1.ClearAllFilters();

Filter Indicator

You can use the ColumnHeaderCell.DropDownButtonImages (‘DropDownButtonImages Property’ in the on-line documentation) property to set an indicator image in the drop button of the filter. Set the ColumnHeaderCell.ShowDropDownButtonImages (‘ShowDropDownButtonImages Property’ in the on-line documentation) property to True to enable the ColumnHeaderCell.DropDownButtonImages property settings.

The filter indicator does not support zoom.

Using Code

The following code sets the image of an indicator which is displayed when the filter is executed.

[VB]

Imports GrapeCity.Win.MultiRow

Dim template As Template = template.Default
Dim columnHeaderCell As ColumnHeaderCell = DirectCast(template.ColumnHeaders(0).Cells(0), ColumnHeaderCell)
Dim headerDropDownList1 As New HeaderDropDownList()
headerDropDownList1.Items.Add(New DropDownAutoFilterItem())

columnHeaderCell.FilterCellName = "textBoxCell1"
columnHeaderCell.DropDownList = headerDropDownList1

columnHeaderCell.DropDownButtonImages.Filtered = New Bitmap("test.bmp")
columnHeaderCell.ShowDropDownButtonImages = True

gcMultiRow1.Template = template

[CS]

using GrapeCity.Win.MultiRow;

Template template = Template.Default;
ColumnHeaderCell columnHeaderCell = template.ColumnHeaders[0].Cells[0] as ColumnHeaderCell;
HeaderDropDownList headerDropDownList1 = new HeaderDropDownList();
headerDropDownList1.Items.Add(new DropDownAutoFilterItem());

columnHeaderCell.FilterCellName = "textBoxCell1";
columnHeaderCell.DropDownList = headerDropDownList1;

columnHeaderCell.DropDownButtonImages.Filtered = new Bitmap("test.bmp");
columnHeaderCell.ShowDropDownButtonImages = true;
gcMultiRow1.Template = template;

Excluding Frozen Rows from Filtering

You can exclude frozen rows from filtering by setting the GcMultiRow.ExcludeFreezeRowsWhenFilter (‘ExcludeFreezeRowsWhenFilter Property’ in the on-line documentation) property to True.

If the GcMultiRow.ExcludeFreezeRowsWhenFilter property is set to True, the values of the frozen rows are not displayed in the filter’s drop-down list.

Using Code

The following code excludes 10 rows from the bottom of the grid, from filtering.

[VB]

GcMultiRow1.FreezeBottomRowCount = 10
GcMultiRow1.ExcludeFreezeRowsWhenFilter = True

[CS]

gcMultiRow1.FreezeBottomRowCount = 10;
gcMultiRow1.ExcludeFreezeRowsWhenFilter = true;

Multiple Selection Filters

You can use the ColumnHeaderCell.DropDownContextMenuStrip (‘DropDownContextMenuStrip Property’ in the on-line documentation) property to set multiple selection filters. Set the HeaderDropDownContextMenuStrip.Items property to an instance of the AutoFilterToolStripItem (‘AutoFilterToolStripItem Class’ in the on-line documentation) class, and set it to the ColumnHeaderCell.DropDownContextMenuStrip property to use the multiple selection filters.

If the DropDownContextMenuStrip property is not empty, the filter settings of the ColumnHeaderCell.DropDownList (‘DropDownList Property’ in the on-line documentation) property become invalid.

The DropDownContextMenuStrip property does not support the display of the designer in the Runtime tab.

Using Code

This example creates multiple selection filters.

[VB]
Imports GrapeCity.Win.MultiRow

Dim textBoxCell1 As New TextBoxCell()
textBoxCell1.Name = "textBoxCell1"

Dim template As Template = template.CreateGridTemplate([New Cell() {textBoxCell1}])

Dim columnHeaderCell = DirectCast(template.ColumnHeaders(0).Cells(0), ColumnHeaderCell)
columnHeaderCell.DropDownList = New HeaderDropDownList("textBoxCell1", false, false)

Dim headerDropDownContextMenu1 As New HeaderDropDownContextMenu()
headerDropDownContextMenu1.Items.Add(New AutoFilterToolStripItem())

columnHeaderCell.DropDownContextMenuStrip = headerDropDownContextMenu1
GcMultiRow1.Template = template

[CS]

using GrapeCity.Win.MultiRow;

TextBoxCell textBoxCell1 = new TextBoxCell();
textBoxCell1.Name = "textBoxCell1";

Template template = Template.CreateGridTemplate(new Cell[] { textBoxCell1 });

ColumnHeaderCell columnHeaderCell = template.ColumnHeaders[0].Cells[0] as ColumnHeaderCell;
columnHeaderCell.DropDownList = new HeaderDropDownList("textBoxCell1", false, false);

HeaderDropDownContextMenu headerDropDownContextMenu1 = new HeaderDropDownContextMenu();
headerDropDownContextMenu1.Items.Add(new AutoFilterToolstripItem());

columnHeaderCell.DropDownContextMenuStrip = headerDropDownContextMenu1;
gcMultiRow1.Template = template;

Use the following steps to set multiple selection filters in the designer.

1. Select the cell on which you wish to set the multiple selection filter (for example: columnHeaderCell1).
2. Select the DropDownList property from the Properties window, and select FilterDropDownList.
3. Select the DropDownList.CellName property from the Properties window, and select the cell on which you want to perform filtering (for example: textBoxCell1).
4. Select the DropDownListContextMenuStrip property from the Properties window, and select Add New to add a HeaderDropDownContextMenu on the template (for example: headerDropDownContextMenu1).
5. Select the HeaderDropDownContextMenu, then select the Items property and click the ... button.
6. Check that showAllToolStripItem1 is set in the Members list of the Item Collection Editor that is displayed.
7. Click the OK button and close the window.
8. Run the project.
9. Display the drop-down list of columnHeaderCell1, and confirm that the multiple selection filter is displayed.

Sorting Rows using Column Headers

You can use the column header cell or the commands in the header drop-down list to sort rows.

- Sorting cannot be used with virtual mode.
- New rows are not considered while sorting.

Clicking Column Headers

To enable sorting when clicking on the column header, set the ColumnHeaderCell.SortMode ('SortMode Property' in the on-line MultiRow Windows Forms Developer’s Guide 194

Copyright © GrapeCity, inc. All rights reserved.
Set the ColumnHeaderCell.SelectionMode property (HeaderCell.SelectionMode 'in the on-line documentation) property) to None.

Using Code

This example enables sorting.

[VB]

```vbnet
Imports GrapeCity.Win.MultiRow

Dim template As Template = template.Default

Dim columnHeaderCell As ColumnHeaderCell = DirectCast(template.ColumnHeaders(0).Cells(0), ColumnHeaderCell)
columnHeaderCell.SelectionMode = MultiRowSelectionMode.None
columnHeaderCell.SortMode = SortMode.Automatic
columnHeaderCell.Value = "(Click)"

GcMultiRow1.Template = template
GcMultiRow1.AllowUserToAddRows = False

' Input sample data
GcMultiRow1.RowCount = 3
GcMultiRow1.Rows(0)("textBoxCell1").Value = 2
GcMultiRow1.Rows(1)("textBoxCell1").Value = 1
GcMultiRow1.Rows(2)("textBoxCell1").Value = 3
```

[CS]

```csharp
using GrapeCity.Win.MultiRow;

Template template = Template.Default;

ColumnHeaderCell columnHeaderCell = (ColumnHeaderCell)template.ColumnHeaders[0].Cells[0];
columnHeaderCell.SelectionMode = MultiRowSelectionMode.None;
columnHeaderCell.SortMode = SortMode.Automatic;
columnHeaderCell.Value = "(Click)";

gcMultiRow1.Template = template;
gcMultiRow1.AllowUserToAddRows = false;

// Input sample data
gcMultiRow1.RowCount = 3;
gcMultiRow1.Rows[0]"textBoxCell1".Value = 2;
gcMultiRow1.Rows[1]"textBoxCell1".Value = 1;
gcMultiRow1.Rows[2]"textBoxCell1".Value = 3;
```

When sorting using column headers, the sort state is displayed on the column header as "Sort Glyph." You can get or set the state of the sort glyph using the ColumnHeaderCell.SortGlyphDirection (SortGlyphDirection 'in the on-line documentation) property. When you set the ColumnHeaderCell.SortMode (SortMode 'in the on-line documentation) property to Automatic, the sort direction becomes fixed in the ascending or descending order. To implement custom sorting, set the ColumnHeaderCell.SortMode (SortMode 'in the on-line documentation) property to Programmatic and use the SortCompare (SortCompare Event 'in the on-line documentation) event.

Setting the Drop-down list

In order to implement sorting using the drop-down list, set the instance of the HeaderDropdownList (HeaderDropdownList Class 'in the on-line documentation) class into the ColumnHeaderCell.DropdownList (DropdownList 'in the on-line documentation) property. Additionally, in order to enable the sorting commands, set the third argument of the constructor to True.

Using Code

This example implements sorting using the drop-down list.

[VB]

```vbnet
Imports GrapeCity.Win.MultiRow
```
Dim template As Template = Template.Default
Dim columnHeaderCell As ColumnHeaderCell = DirectCast(template.ColumnHeaders(0).Cells(0), ColumnHeaderCell)
columnHeaderCell.DropDownList = New HeaderDropDownList("textBoxCell1", False, True)
GcMultiRow1.Template = template
GcMultiRow1.AllowUserToAddRows = False
' Input Sample data
GcMultiRow1.RowCount = 3
GcMultiRow1.Rows(0)("textBoxCell1").Value = 2
GcMultiRow1.Rows(1)("textBoxCell1").Value = 1
GcMultiRow1.Rows(2)("textBoxCell1").Value = 3

The range of rows to be sorted can be set with the HeaderDropDownList.StartRow ('StartRow Property' in the on-line documentation) and the HeaderDropDownList.EndRow ('EndRow Property' in the on-line documentation) properties.

Sort Indicator
You can set the indicator image during sorting using the ColumnHeaderCell.SortGlyphAscendingImage ('SortGlyphAscendingImage Property' in the on-line documentation) and ColumnHeaderCell.SortGlyphDescendingImage ('SortGlyphDescendingImage Property' in the on-line documentation) properties.

The Sort indicator does not support zoom.

Using Code
The following code sets an indicator image which is displayed when sorting is executed.

[VB]
Imports GrapeCity.Win.MultiRow
Dim template As Template = Template.Default
Dim columnHeaderCell As ColumnHeaderCell = DirectCast(template.ColumnHeaders(0).Cells(0), ColumnHeaderCell)
columnHeaderCell.SelectionMode = MultiRowSelectionMode.None
columnHeaderCell.SortMode = SortMode.Automatic
columnHeaderCell.Value = "[クリック]"
columnHeaderCell.SortGlyphAscendingImage = New Bitmap("test1.bmp")
columnHeaderCell.SortGlyphDescendingImage = New Bitmap("test2.bmp")
GcMultiRow1.Template = template
GcMultiRow1.AllowUserToAddRows = False
' Input Sample data
GcMultiRow1.RowCount = 3
GcMultiRow1.Rows(0)("textBoxCell1").Value = 2
GcMultiRow1.Rows(1)("textBoxCell1").Value = 1
GcMultiRow1.Rows(2)("textBoxCell1").Value = 3

Copyright © GrapeCity, inc. All rights reserved.
[CS]

```c
using GrapeCity.Win.MultiRow;

Template template = Template.Default;

ColumnHeaderCell columnHeaderCell = (ColumnHeaderCell)template.ColumnHeaders[0].Cells[0];
columnHeaderCell.SelectionMode = MultiRowSelectionMode.None;
columnHeaderCell.SortMode = SortMode.Automatic;
columnHeaderCell.Value = "(クリック)";
columnHeaderCell.SortGlyphAscendingImage = new Bitmap("test1.bmp");
columnHeaderCell.SortGlyphDescendingImage = new Bitmap("test2.bmp");

gcMultiRow1.Template = template;
gcMultiRow1.AllowUserToAddRows = false;

// Input sample data
gcMultiRow1.RowCount = 3;
gcMultiRow1.Rows[0]["textBoxCell1"].Value = 2;
gcMultiRow1.Rows[1]["textBoxCell1"].Value = 1;
gcMultiRow1.Rows[2]["textBoxCell1"].Value = 3;
```

Changing the Header Appearance

The visual style is enabled in the GcMultiRow control headers by default. The effect of the visual style depends on the OS where the application is running. The following information is common for **HeaderCell ('HeaderCell Class' in the on-line documentation)**, **ColumnHeaderCell ('ColumnHeaderCell Class' in the on-line documentation)**, **RowHeaderCell ('RowHeaderCell Class' in the on-line documentation)**, and **CornerHeaderCell ('CornerHeaderCell Class' in the on-line documentation)** sections.

**Flat Style**

You can use the flat style by setting the **HeaderCell.FlatStyle ('FlatStyle Property' in the on-line documentation)** property. The appearance when hovering with the mouse is set using the **HeaderCell.FlatAppearance ('FlatAppearance Property' in the on-line documentation)** property.

**Office 2007 Style**

You can use the Office 2007-like style by setting the **HeaderCell.Office2007Style ('Office2007Style Property' in the on-line documentation)** property.

<table>
<thead>
<tr>
<th>Office2007Style</th>
<th>Appearance</th>
<th>Mouse Over</th>
<th>Line Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Silver</td>
<td>[Silver]</td>
<td>[Silver]</td>
<td>Color.FromArgb(144, 145, 146)</td>
</tr>
<tr>
<td>Blue</td>
<td>[Blue]</td>
<td>[Blue]</td>
<td>Color.FromArgb(158, 182, 206)</td>
</tr>
<tr>
<td>Black</td>
<td>[Black]</td>
<td>[Black]</td>
<td>Color.FromArgb(182, 182, 182)</td>
</tr>
</tbody>
</table>

**Using Code**

This example sets a style and border.

[VB]
Imports GrapeCity.Win.MultiRow

GcMultiRow1.Template = Template.Default
Dim template1 As Template = GcMultiRow1.Template
For Each cell As Cell In template1.ColumnHeader.Cells
    If TypeOf cell Is HeaderCell Then
        Dim headerCell As HeaderCell = TryCast(cell, HeaderCell)
        headerCell.Style.Border = New Border(LineStyle.Thin, Color.FromArgb(158, 182, 206))
    End If
Next
GcMultiRow1.Template = template1

[CS]

using GrapeCity.Win.MultiRow;

gcMultiRow1.Template = Template.Default;
Template template1 = gcMultiRow1.Template;
foreach (Cell cell in template1.ColumnHeader.Cells)
{
    if (cell is HeaderCell)
    {
        HeaderCell headerCell = cell as HeaderCell;
        headerCell.Style.Border = new Border(LineStyle.Thin, Color.FromArgb(158, 182, 206));
    }
}
gcMultiRow1.Template = template1;

Header Borders

You can display the gutter specifying the boundaries in the header cell by using the HeaderCell.GutterStyles ('GutterStyles Property' in the on-line documentation) property. The gutter indicating the boundaries can be displayed on either or all of the top, bottom, left, or right sides. The header boundaries are displayed only when the visual style is enabled. In other cases, the boundaries are displayed using the cell Borders.

Effect Direction

You can specify the position of the highlight effect when the mouse pointer is on the header cell, using the HeaderCell.HoverDirection ('HoverDirection Property' in the on-line documentation) property. The highlight effect can be applied to either the top or bottom or right or left. You may also choose not to display it.

Using Row Headers

You can use the row header cell to let the users select the rows, specify the indicator for the present row, display an icon to indicate the row that is being edited, and show any errors in the row.

Indicator

You can indicate the current row to the user by displaying an icon. To display an indicator in the row header cell, set the RowHeaderCell.ShowIndicator ('ShowIndicator Property' in the on-line documentation) property to True.
A pen like icon is shown when the user is editing the current row. Set the GcMultiRow.ShowEditingIcon ('ShowEditingIcon Property' in the on-line documentation) property to False to hide the icon.

Selecting Rows

The row can be selected using the row header cell by default. Use the RowHeaderCellSelectionMode ('SelectionMode Property' in the on-line documentation) property to change this behavior.

Fixing the Scrolling

By default, the row header cell in the row can be scrolled horizontally in the same way as the other cells. This horizontal scrolling can be prevented and the row header cell can always be shown on the left edge as the row header by implementing Freezing Rows.

Resize

Resizing vertically using the row header cell works by default. In order to change this behavior, you can change the RowHeaderCell.ResizeMode ('ResizeMode Property' in the on-line documentation) property value.

Row Errors

An error icon is displayed in the row header cell when the Row.ErrorText ('ErrorText Property' in the on-line documentation) property is not null. For details, refer to Displaying Error Messages.

Cell Types

MultiRow offers 26 types of cells. The developer can also create a custom cell type.

- Base Cell
- HeaderCell
- ColumnHeaderCell
- RowHeaderCell
- CornerHeaderCell
- TextBoxCell
Base Cell

The base cell is the base for all MultiRow cell types. You can use a base cell to create editable cells or headers.

Features

You can use the following features in a base cell.

- Position
- Size
- Style
- Allow Editing
- Whether Events can be displayed
- Show or Hide
- Get display state
- Allow Selection
- Get selection state
- Specify field for data binding
- Status of data binding
- Tab Order and Tab Stop
- Changing Cell Size
- Maximum and Minimum Cell Size
- Context Menu
Error Icon Display
Error Icon Placement
ToolTip

For more details on each feature, refer to the Cell ('Cell Class' in the on-line documentation) class. For more information about the cell style, refer to Cell Styles and Conditional Cell Styles.

**DataType**

The base cell stores the System.Object value type. This type can be confirmed with the Cell.ValueType (Value Type Property' in the on-line documentation) property. The input, display, and edit features are not implemented here.

**Cell Edit Control**

The base cell does not provide a cell edit control. The Cell.EditType (EditType Property' in the on-line documentation) property returns a null reference (Nothing in Visual Basic).

**Style**

The base cell does not provide any styles.

**Shortcut Key**

The base cell does not provide any shortcut keys.

**Event**

The base cell does not provide any events.

**HeaderCell**

The header cell offers features such as cell selection or resizing, caption display, and visual styles. You can use ColumnHeaderCell, RowHeaderCell, and CornerHeaderCell inherited from header cell to create a column header, row header, or corner header respectively.

**Features**

In addition to the Base Cell features, the following features can also be used in a header cell.

- Selection Direction for Cell
- Display Header Separator Line
- Visual Style

For more details on each feature, refer to the HeaderCell ('HeaderCell Class' in the on-line documentation) class.

**Data Types**

The header cell uses the Object value type and this type can be checked with the HeaderCell.ValueType (Value Type Property' in the on-line documentation) property. The header caption uses a value that has been type cast to String type. This type can be checked by the HeaderCell.FormattedValueType (FormattedValueType Property' in the on-line documentation) property. You can modify the behavior at the time of type casting by overriding the ToString method of Object type. Use the HeaderCell.OnCellFormatting (OnCellFormatting Method' in the on-line documentation) method to change behavior when a value is read into the cell. Override the HeaderCell.OnCellParsing (OnCellParsing Method' in the on-line documentation) method to change
behavior when the value is written back from the cell.

**Cell Edit Control**

The header cell does not provide a cell edit control. The HeaderCell.**EditType** (**EditType Property** in the on-line documentation) property always returns a null reference (Nothing in Visual Basic).

**Styles**

The header cell supports following the members of the **CellStyle** (**CellStyle Class** in the on-line documentation) class. You can set cell styles with the HeaderCell.**Style** (**Style Property** in the on-line documentation) property.

### CellStyle Members

<table>
<thead>
<tr>
<th>Member</th>
<th>enabled or disabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>Enabled only when ModeHeaderCell.FlatStyle is other than System and HeaderCell.UseVisualStyleBackColor (UseVisualStyleBackColor Property in the on-line documentation) is False</td>
</tr>
<tr>
<td>BackgroundGradientEffect</td>
<td>Enabled only when HeaderCell.FlatStyle (FlatStyle Property in the on-line documentation) is other than System and HeaderCell.UseVisualStyleBackColor is False.</td>
</tr>
<tr>
<td>Border</td>
<td>Enabled</td>
</tr>
<tr>
<td>DataSourceNullValue</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledBackColor</td>
<td>Enabled only when ModeHeaderCell.FlatStyle is other than System and HeaderCell.UseVisualStyleBackColor is False</td>
</tr>
<tr>
<td>DisabledForeColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledGradientEffect</td>
<td>Enabled only when HeaderCell.FlatStyle is other than System and HeaderCell.UseVisualStyleBackColor is False.</td>
</tr>
<tr>
<td>EditingBackColor</td>
<td>-</td>
</tr>
<tr>
<td>EditingForeColor</td>
<td>-</td>
</tr>
<tr>
<td>Font</td>
<td>Enabled</td>
</tr>
<tr>
<td>ForeColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>Format</td>
<td>Enabled</td>
</tr>
<tr>
<td>FormatProvider</td>
<td>Enabled</td>
</tr>
<tr>
<td>Image</td>
<td>Enabled</td>
</tr>
<tr>
<td>ImageAlign</td>
<td>Enabled</td>
</tr>
<tr>
<td>ImeMode</td>
<td>-</td>
</tr>
<tr>
<td>ImeSentenceMode</td>
<td>-</td>
</tr>
<tr>
<td>InputScope</td>
<td>-</td>
</tr>
<tr>
<td>LineAdjustment</td>
<td>Enabled only for GDI+Compatible Mode</td>
</tr>
<tr>
<td>Margin</td>
<td>Enabled</td>
</tr>
<tr>
<td>MouseOverBackColor</td>
<td>Enabled only when HeaderCell.FlatStyle is other than System and HeaderCell.UseVisualStyleBackColor is False.</td>
</tr>
<tr>
<td>MouseOverForeColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>MouseOverGradientEffect</td>
<td>Enabled only when HeaderCell.FlatStyle is other than System and HeaderCell.UseVisualStyleBackColor is False.</td>
</tr>
<tr>
<td>Multiline</td>
<td>Enabled</td>
</tr>
</tbody>
</table>
NullValue: Enabled
Padding: Enabled
PatternColor: Enabled only when ModeHeaderCell.FlatStyle is other than System and HeaderCell.UseVisualStyleBackColor is False
PatternStyle: Enabled only when ModeHeaderCell.FlatStyle is other than System and HeaderCell.UseVisualStyleBackColor is False
SelectionBackColor: -
SelectionForeColor: -
SelectionGradientEffect: -
Tag: Enabled
TextAdjustment: Enabled only for GDI+Compatible Mode
TextAlign: Enabled
TextAngle: Enabled only for GDI+Compatible Mode
TextEffect: Enabled
TextImageRelation: Enabled
TextIndent: Enabled
TextVertical: Enabled only for GDI+Compatible Mode
UseCompatibleTextRendering: Enabled
WordWrap: Enabled

To enable **GDI+ Compatibility Mode**, set the UseCompatibleTextRendering property to True.

**Shortcut Keys**

Shortcut keys are not handled in header cells.

**Events**

You can use the GcMultiRow.CellContentClick ('CellContentClick Event' in the on-line documentation) event to get a click in the cell content area. You can use the GcMultiRow.CellContentDoubleClick ('CellContentDoubleClick Event' in the on-line documentation) event for a double-click.

**Change BackColor (HeaderCell)**

You can change the header cell backcolor by setting any color in the HeaderCell.Style.BackColor property and then using any of the following settings.

- Set the HeaderCell.FlatStyle ('FlatStyle Property' in the on-line documentation) property to Flat or Popup.
- Set the HeaderCell.FlatStyle property to Standard and the HeaderCell.UseVisualStyleBackColor ('UseVisualStyleBackColor Property' in the on-line documentation) property to False.

In an environment where a visual style is enabled, the backcolor set using the HeaderCell.Style.BackColor property is not applied to the button if either the HeaderCell.FlatStyle property is set to System or the HeaderCell.UseVisualStyleBackColor property is set to True.
Using the Designer

1. Select a button cell whose back color needs to be changed (For example: headerCell1).
2. From the Properties window, select the HeaderCell.Style.BackColor property and choose any color from the drop-down.
3. From the Properties window, select the HeaderCell.UseVisualStyleBackColor property and set it to False.
4. From the Properties window, select the HeaderCell.FlatStyle property and either set it to Flat, Popup, or Standard.

Using Code

The following code sets the backcolor for the header cell.

[VB]

Imports GrapeCity.Win.MultiRow
Dim template As New Template
Dim headerCell1 As New HeaderCell()
headerCell1.Name = "headerCell1"
headerCell1.Value = "HeaderCell"
headerCell1.UseVisualStyleBackColor = false
headerCell1.FlatStyle = FlatStyle.Standard
GcMultiRow1.Template = Template.CreateGridTemplate(New Cell() { headerCell1 })
GcMultiRow1.RowCount = 10

[CS]

using GrapeCity.Win.MultiRow;

Template template = new Template();
HeaderCell headerCell1 = new HeaderCell();
headerCell1.Name = "headerCell1";
headerCell1.Value = "HeaderCell";
headerCell1.Style.BackColor = Color.FromArgb(192, 255, 192);
headerCell1.Style.Border = new Border(LineStyle.Thin, Color.DarkGray);
headerCell1.UseVisualStyleBackColor = false;
headerCell1.FlatStyle = FlatStyle.Standard;
gcMultiRow1.Template = Template.CreateGridTemplate(new Cell[] { headerCell1 });
gcMultiRow1.RowCount = 10;

ColumnHeaderCell

The column header cell can display column selection or sorting indicators.

Quantity
Features

In addition to the features of the **HeaderCell**, the following features can also be used in ColumnHeaderCell.

- Rearrange Rows (Built-In Sorting)
- Filter Rows (Built-In Filter)
- Drop-Down List

For more details on each feature, refer to the **ColumnHeaderCell** ('ColumnHeaderCell Class' in the on-line documentation) class.

For more information on data types, cell edit control, style, shortcut keys, and events, refer to **HeaderCell**.

Using Code

The code below shows a column header cell that selects a cell in the column header section and a column header cell that sorts a cell.

```vbnet
Imports GrapeCity.Win.MultiRow

Dim template As Template = New Template()
Dim columnHeaderSection As ColumnHeaderSection = New ColumnHeaderSection()
Dim columnHeaderCell1 As ColumnHeaderCell = New ColumnHeaderCell()
Dim columnHeaderCell2 As ColumnHeaderCell = New ColumnHeaderCell()
columnHeaderCell1.Location = New Point(0, 0)
columnHeaderCell1.Value = "Select"
columnHeaderCell2.Location = New Point(columnHeaderCell1.Width, 0)
columnHeaderCell2.Value = "Sort"
columnHeaderCell2.SortCellIndex = 1
columnHeaderCell2.SelectionMode = MultiRowSelectionMode.None
columnHeaderCell2.SortMode = SortMode.Automatic
columnHeaderSection.Cells.Add(columnHeaderCell1)
columnHeaderSection.Cells.Add(columnHeaderCell2)
columnHeaderSection.Height = columnHeaderCell1.Height

Dim textBoxCell1 As TextBoxCell = New TextBoxCell()
Dim textBoxCell2 As TextBoxCell = New TextBoxCell()
textBoxCell1.Location = New Point(0, 0)
textBoxCell2.Location = New Point(textBoxCell1.Width, 0)
template.Row.Cells.Add(textBoxCell1)
template.Row.Cells.Add(textBoxCell2)
template.Row.Height = textBoxCell1.Height
template.Width = textBoxCell1.Size.Width * 2
template.ColumnHeaders.Add(columnHeaderSection)

GcMultiRow1.Template = template
GcMultiRow1.AllowUserToAddRows = False
GcMultiRow1.RowCount = 10

For i As Integer = 0 To GcMultiRow1.RowCount - 1
    GcMultiRow1.Rows(i).Cells(1).Value = i.ToString()
Next
```
using GrapeCity.Win.MultiRow;

Template template = new Template();
ColumnHeaderSection columnHeaderSection = new ColumnHeaderSection();
ColumnHeaderCell columnHeaderCell1 = new ColumnHeaderCell();
ColumnHeaderCell columnHeaderCell2 = new ColumnHeaderCell();
columnHeaderCell1.Location = new Point(0, 0);
columnHeaderCell1.Value = "Select";
columnHeaderCell2.Location = new Point(columnHeaderCell1.Width, 0);
columnHeaderCell2.Value = "Sort";
columnHeaderCell2.SortCellIndex = 1;
columnHeaderCell2.SelectionMode = MultiRowSelectionMode.None;
columnHeaderCell2.SortMode = SortMode.Automatic;
columnHeaderSection.Cells.Add(columnHeaderCell1);
columnHeaderSection.Cells.Add(columnHeaderCell2);
columnHeaderSection.Height = columnHeaderCell1.Height;

TextBoxCell textBoxCell1 = new TextBoxCell();
TextBoxCell textBoxCell2 = new TextBoxCell();
textBoxCell1.Location = new Point(0, 0);
textBoxCell2.Location = new Point(textBoxCell1.Width, 0);
template.Row.Cells.Add(textBoxCell1);
template.Row.Cells.Add(textBoxCell2);

template.Row.Height = textBoxCell1.Height;
template.Width = textBoxCell1.Size.Width * 2;
template.ColumnHeader.Add(columnHeaderSection);

gcMultiRow1.Template = template;
gcMultiRow1.AllowUserToAddRows = false;
gcMultiRow1.RowCount = 10;

for (int i = 0; i < gcMultiRow1.RowCount; i++)
{
    gcMultiRow1.Rows[i].Cells[1].Value = i.ToString();
}

RowHeaderCell

The row header cell can display an indicator and show row selection.

- The row header cell can only be placed in a row.

Features

In addition to the features of the HeaderCell, the following features can also be used in a row header cell.

- Display or Hide Row Indicator
- Display or Hide Row Error
- Display Row Number
Show Row Selection

For more details on each feature, refer to the RowHeaderCell ('RowHeaderCell Class' in the on-line documentation) class.

Data Types

Refer to HeaderCell.

Cell Edit Control

Refer to HeaderCell.

Styles

Refer to HeaderCell.

Shortcut Keys

Refer to HeaderCell.

Events

Refer to HeaderCell.

Using Code

The code below shows a row header cell and a text box cell in a row.

[VB]

Imports GrapeCity.Win.MultiRow

Dim template As New Template()
Dim rowHeaderCell1 As New RowHeaderCell()
Dim textBoxCell1 As New TextBoxCell()

rowHeaderCell1.Location = New Point(0, 0)
rowHeaderCell1.SelectionMode = MultiRowSelectionMode.Row
rowHeaderCell1.HoverDirection = HoverDirection.Right
rowHeaderCell1.ResizeMode = ResizeMode.None
rowHeaderCell1.ValueFormat = "%1%"

textBoxCell1.Location = New Point(rowHeaderCell1.Width, 0)

template.Row.Cells.Add(rowHeaderCell1)
template.Row.Cells.Add(textBoxCell1)
template.Row.Height = textBoxCell1.Height
template.Width = textBoxCell1.Width * 2

GcMultiRow1.Template = template
GcMultiRow1.RowCount = 10

[CS]

using GrapeCity.Win.MultiRow;

Template template = new Template();
RowHeaderCell rowHeaderCell1 = new RowHeaderCell();
TextBoxCell textBoxCell1 = new TextBoxCell();

rowHeaderCell1.Location = new Point(0, 0);
rowHeaderCell1.SelectionMode = MultiRowSelectionMode.Row;
rowHeaderCell1.HoverDirection = HoverDirection.Right;
rowHeaderCell1.ResizeMode = ResizeMode.None;
rowHeaderCell1.ValueFormat = "%1%";

textBoxCell1.Location = new Point(rowHeaderCell1.Width, 0);

template.Row.Cells.Add(rowHeaderCell1);
template.Row.Cells.Add(textBoxCell1);
template.Row.Height = textBoxCell1.Height;
template.Width = textBoxCell1.Width * 2;

gcMultiRow1.Template = template;
gcMultiRow1.RowCount = 10;

CornerHeaderCell

The corner header cell is a header that can be displayed in the upper left corner.

The corner header cell can only be placed in the column header section

Features

In addition to the HeaderCell features, the following features can also be used in the corner header cell.

- Settings applied to the upper left corner
  - Select all cells when clicking
  - Specified direction for mouse over is 'None'
  - Default size is 80x20
- Office 2007 style specific appearance

For more details on each feature, refer to the CornerHeaderCell ('CornerHeaderCell Class' in the on-line documentation) class.

For more information on data types, cell edit control, style, shortcut keys, and events, refer to HeaderCell.

TextBoxCell

The text box cell has features similar to the .NET Framework's TextBox control (System.Windows.Forms.TextBox). The user can use the text box cell to type text in a cell.

Text

Features

In addition to the Base Cell features, the following features can also be used in the text box cell.

- Text Input and Display
- Input Text in Multiline
- Text Word Wrap
- Read-Only Text
- Mask for Password Characters
- Display Scroll bar
- Display Context Menu
- Events

For more details on each feature, refer to the [TextBoxCell (TextBoxCell Class' in the on-line documentation)](http://example.com) class.

**Data Types**

The text box cell uses the Object type value. This type can be checked with the TextBoxCell.ValueType ('ValueType Property' in the on-line documentation) property. The value cast into the String type is used for input and display. This type can be checked with the TextBoxCell.FormattedValueType ('FormattedValueType Property' in the on-line documentation) property. The behavior during casting can be modified by overriding the ToString Object type method. You can override the TextBoxCell.OnCellFormatting ('OnCellFormatting Method' in the on-line documentation) method to modify the behavior when values are read in a cell. Override the TextBoxCell.OnCellParsing ('OnCellParsing Method' in the on-line documentation) method to modify the behavior when a value is written back from the cell.

**Cell Edit Control**

The text box cell value is edited in the TextBoxEditingControl control. This control is inherited from the IEditingControl interface and System.Windows.Forms.TextBox Class. The cell edit control type can be checked with the TextBoxCell.EditType ('EditType Property' in the on-line documentation) property.

**Styles**

The text box cell supports the following CellStyle ('CellStyle Class' in the on-line documentation) class members. The cell style is set with the TextBoxCell.Style property.

<table>
<thead>
<tr>
<th>CellStyle Member</th>
<th>Non-Editable State</th>
<th>Editable State</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>BackgroundGradientEffect</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>Border</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>DataSourceNullValue</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledBackColor</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>DisabledForeColor</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>DisabledGradientEffect</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>EditingBackColor</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td>EditingForeColor</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td>Font</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>ForeColor</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>Format</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>FormatProvider</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>Image</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>ImageAlign</td>
<td>Enabled</td>
<td>-</td>
</tr>
</tbody>
</table>
To enable **GDI+ Compatibility Mode**, set the `TextBoxCell.Style.UseCompatibleTextRendering` property to True.

**Shortcut Keys**

The following table lists keys processed during the editing of the text box cell and lists keys processed by the `GcMultiRow` control.

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Key</th>
<th>TextBoxCell</th>
<th>GcMultiRow</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Keys.PageUp</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.PageDown</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.End</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Home</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Left</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Right</td>
<td>Enabled</td>
<td>-</td>
</tr>
</tbody>
</table>

Copyright © GrapeCity, inc. All rights reserved.
### Keys

<table>
<thead>
<tr>
<th>Key</th>
<th>Enabled</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Keys.Up</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td>Keys.Down</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td>Keys.Insert</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keys.Delete</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td>Keys.BackSpace</td>
<td>Enabled</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Enabled</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Keys.Control</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keys.PageUp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keys.PageDown</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keys.End</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td>Keys.Home</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td>Keys.Left</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td>Keys.Right</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td>Keys.Up</td>
<td></td>
<td>Enabled</td>
</tr>
<tr>
<td>Keys.Down</td>
<td></td>
<td>Enabled</td>
</tr>
<tr>
<td>Keys.A</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td>Keys.C</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td>Keys.V</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td>Keys.X</td>
<td>Enabled</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Enabled</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Keys.Shift</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keys.Left</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td>Keys.Right</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td>Keys.Up</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td>Keys.Down</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td>Keys.Home</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td>Keys.End</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td>Keys.Enter</td>
<td>Enabled only for Multiline</td>
<td></td>
</tr>
<tr>
<td>Keys.F10</td>
<td>Enabled</td>
<td></td>
</tr>
</tbody>
</table>

- **Keys.Control** + **Keys.C** is processed by the GcMultiRow control when multiple cells are selected.
- **Keys.PageDown** and **Keys.PageUp** are processed by the GcMultiRow control when the text box cell value has not been changed.
- **Keys.Left** is processed by the GcMultiRow control when the cursor is before the first character of the text box cell.
- **Keys.Right** is processed by the GcMultiRow control when the cursor is after the last character of the text box cell.
- **Keys.Up** is processed by the GcMultiRow control when the text box cell cursor is in the first row.
- **Keys.Down** is processed by the GcMultiRow control when the text box cell cursor is in the last row.
- **Keys.Up** and **Keys.Down** are processed by the GcMultiRow control when the text box cell does not allow multiline.

### Events

You can use the GcMultiRow.**CellContentClick** (["CellContentClick Event' in the on-line documentation"](https://www.grapeweekly.com)) event to implement processing when clicking in the cell content area. Use the GcMultiRow.**CellContentDoubleClick**
('CellContentDoubleClick Event' in the on-line documentation) event for a double-click. Use the GcMultiRow.CellEditedFormattedValueChanged ('CellEditedFormattedValueChanged Event' in the on-line documentation) event to get cell value changes. The TextBoxEditingControl ('TextBoxEditingControl Class' in the on-line documentation) class events are used to implement processing in the text box cell edit state.

Comparison with Standard Control

The table below shows a comparison between major properties of the text box cell and the System.Windows.Forms.TextBox control (System.Windows.Forms.DataGridViewTextBoxCell class).

<table>
<thead>
<tr>
<th>TextBoxCell</th>
<th>TextBox</th>
<th>DataGridViewTextBoxCell</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>AcceptsReturns</td>
<td>None</td>
</tr>
<tr>
<td>None</td>
<td>AutoCompleteCustomSource</td>
<td>None</td>
</tr>
<tr>
<td>None</td>
<td>AutoCompleteMode</td>
<td>None</td>
</tr>
<tr>
<td>None</td>
<td>AutoCompleteSource</td>
<td>None</td>
</tr>
<tr>
<td>Style.BackColor</td>
<td>BackColor</td>
<td>Style.BackColor</td>
</tr>
<tr>
<td>CharacterCasing</td>
<td>CharacterCasing</td>
<td>None</td>
</tr>
<tr>
<td>Style.ForeColor</td>
<td>ForeColor</td>
<td>Style.ForeColor</td>
</tr>
<tr>
<td>MaxLength</td>
<td>MaxLength</td>
<td>MaxInputLength</td>
</tr>
<tr>
<td>Style.Multiline</td>
<td>MultiLine</td>
<td>None</td>
</tr>
<tr>
<td>PasswordChar</td>
<td>PasswordChar</td>
<td>None</td>
</tr>
<tr>
<td>ScrollBars</td>
<td>ScrollBars</td>
<td>None</td>
</tr>
<tr>
<td>Value</td>
<td>Text</td>
<td>Value</td>
</tr>
<tr>
<td>Style.TextAlign</td>
<td>TextAlign</td>
<td>Style.Alignment</td>
</tr>
<tr>
<td>UseSystemPasswordChar</td>
<td>UseSystemPasswordChar</td>
<td>None</td>
</tr>
<tr>
<td>Style.WordWrap</td>
<td>WordWrap</td>
<td>Style.WrapMode</td>
</tr>
</tbody>
</table>

For more information, see the following topics:

- **Input Line Break (TextBoxCell)**
- **Display Vertical Scroll Bar (TextBoxCell)**
- **How to Use KeyDown Event (TextBoxCell)**

Input Line Break (TextBoxCell)

The user can use the Shift + Enter key to input a line break at runtime. This is the same key used in the System.Windows.Forms.TextBox.

Using the Designer

Use the following steps to allow line breaks in a text box cell.

1. Select the text box cell in which you wish to allow line breaks (for example: textBoxCell1)
2. Drag the cell and make its height about twice the current size.
3. From the Properties window, select the TextBoxCell.Style.Multiline property and choose True from the drop-

Copyright © GrapeCity, inc. All rights reserved.
down list.
4. Change the designer document window tab to Runtime.
5. Input the line break using the Shift + Enter key.

Using Code

The following code enables multiple lines and adds a line break to the cell. This is similar to how you would do this with a TextBox control.

[VB]

Imports GrapeCity.Win.MultiRow

GcMultiRow1.Template = Template.CreateGridTemplate(New Cell() {New TextBoxCell()})
' Enable multiline
GcMultiRow1.Rows(0).Cells(0).Style.Multiline = MultiRowTriState.True
GcMultiRow1.Rows(0).Cells(0).Value = "Hello" & vbCrLf & "World"
' Auto adjust cell height
GcMultiRow1.Rows(0).Cells(0).PerformVerticalAutoFit()

[CS]

using GrapeCity.Win.MultiRow;

gcMultiRow1.Template = Template.CreateGridTemplate(new Cell[] { new TextBoxCell() });
// Enable multiline
gcMultiRow1.Rows[0].Cells[0].Style.Multiline = MultiRowTriState.True;
gcMultiRow1.Rows[0].Cells[0].Value = "Hello\r\nWorld";
// Auto adjust cell height
gcMultiRow1.Rows[0].Cells[0].PerformVerticalAutoFit();

For information about multiple lines and word wrap, refer to Multiline and Word Wrap.

How to Use KeyDown Event (TextBoxCell)

You can perform processing similar to the TextBox.KeyDown event in the text box cell by using the TextBoxEditingControl.KeyDown event.

Using Code

This example uses the KeyDown event.

[VB]

Imports GrapeCity.Win.MultiRow

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    GcMultiRow1.Template = Template.Default
End Sub

Private Sub GcMultiRow1_EditingControlShowing(ByVal sender As System.Object, ByVal e As EditingControlShowingEventArgs) Handles GcMultiRow1.EditingControlShowing
    Dim textBox As TextBoxEditingControl = TryCast(e.Control, TextBoxEditingControl)
    If textBox IsNot Nothing Then
        RemoveHandler textBox.KeyDown, AddressOf Me.textBoxEditingControl_KeyDown
        AddHandler textBox.KeyDown, AddressOf Me.textBoxEditingControl_KeyDown
    End If
End Sub

Private Sub textBoxEditingControl_KeyDown(ByVal sender As System.Object, ByVal e As KeyEventArgs)
    Console.WriteLine(e.KeyCode)
End Sub

[CS]

Copyright © GrapeCity, inc. All rights reserved.
using GrapeCity.Win.MultiRow;

private void Form1_Load(object sender, EventArgs e)
{
    gcMultiRow1.Template = Template.Default;
    gcMultiRow1.EditingControlShowing += new EventHandler<EditingControlShowingEventArgs>(gcMultiRow1_EditingControlShowing);
}

private void gcMultiRow1_EditingControlShowing(object sender, EditingControlShowingEventArgs e)
{
    TextBoxEditingControl textBox = e.Control as TextBoxEditingControl;
    if (textBox != null)
    {
        textBox.KeyDown -= new KeyEventHandler(textBox_KeyDown);
        textBox.KeyDown += new KeyEventHandler(textBox_KeyDown);
    }
}

private void textBox_KeyDown(object sender, KeyEventArgs e)
{
    Console.WriteLine(e.KeyCode);
}

To determine which current cell is using the cell edit control events, you can use the IEditingControl.GcMultiRow property and get the GcMultiRow that hosts the cell edit control.

Using Code
This example gets the current cell in the KeyDown event.

[VB]
Private Sub textBoxEditingControl_KeyDown(ByVal sender As System.Object, ByVal e As KeyEventArgs)
    Dim textBox1 As TextBoxEditingControl = TryCast(sender, TextBoxEditingControl)
End Sub

[CS]
private void textBox_KeyDown(object sender, KeyEventArgs e)
{
    TextBoxEditingControl textBox1 = sender as TextBoxEditingControl;
}

Display Vertical Scroll Bar (TextBoxCell)

You can create a text box cell that allows the input of multiline text and has a vertical scroll bar. The text box cell scroll bar is displayed only when the cell is in an editable state.

Using the Designer

Use the following steps to create a text box cell that allows the input of multiline text and has a vertical scroll bar

1. Select a text box cell where you need to display a vertical scroll bar (for example: textBoxCell1).
2. Drag the cell and make its height about twice its current size.
3. From the Properties window, select the ScrollBars property and choose Vertical from the drop-down list.
4. From the Properties window, select the Style property and expand the properties by clicking the plus sign on the left.
5. Select the Multiline property and choose True from the drop-down list.
6. Select the TextAlign property and choose TopLeft from the drop-down list.
7. Change the designer document window tab to **Runtime**.
8. Input any value based on **Input Line Break (TextBoxCell)**.

**Using Code**

The following code creates a text box cell that allows the input of multiline text and has a vertical scroll bar.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

Dim template As New Template()
Dim textBoxCell1 As New TextBoxCell()

textBoxCell1.Name = "textBoxCell1"
textBoxCell1.ScrollBars = ScrollBars.Vertical
textBoxCell1.Size = New Size(80, 60)
textBoxCell1.Style.Multiline = MultiRowTriState.True
textBoxCell1.Style.TextAlign = MultiRowContentAlignment.TopLeft

GcMultiRow1.Template = Template.CreateGridTemplate(New Cell() { textBoxCell1 })
GcMultiRow1.RowCount = 10
```

**[CS]**

```cs
using GrapeCity.Win.MultiRow;

Template template = new Template();
TextBoxCell textBoxCell1 = new TextBoxCell();

textBoxCell1.Name = "textBoxCell1";
textBoxCell1.ScrollBars = ScrollBars.Vertical;
textBoxCell1.Size = new Size(80, 60);
textBoxCell1.Style.Multiline = MultiRowTriState.True;
textBoxCell1.Style.TextAlign = MultiRowContentAlignment.TopLeft;

gcMultiRow1.Template = Template.CreateGridTemplate(new Cell[] { textBoxCell1 });
gcMultiRow1.RowCount = 10;
```

**ButtonCell**

The button cell offers features equivalent to the Button control of the .NET Framework (System.Windows.Forms.Button). The user can use the button cell to execute a command by clicking. The size of the button is automatically adjusted according to the cell size.

**Features**

In addition to the **Base Cell** features, the following features can also be used in a button cell.

- Display clickable button
- Change button style

For more details on each feature, refer to the **ButtonCell** (*ButtonCell Class* in the on-line documentation)
class.

Data Type

The button cell uses the value Object type and this type can be checked with the ButtonCell.ValueType (Value
type Property' in the on-line documentation) property. The caption displayed on the button uses a value that has been
type cast to a String type. This type can be checked by the ButtonCell.FormattedValueType
(FormattedValueType Property' in the on-line documentation) property. You can modify the behavior when
type casting by overriding the ToString method of the Object type. Override the ButtonCell.OnCellFormatting
(OnCellFormatting Method' in the on-line documentation) method to change behavior when a value is read
into the cell. Then override the ButtonCell.OnCellParsing (OnCellParsing Method' in the on-line
documentation) method to change behavior when the value is written back from the cell.

Cell Edit Control

The button cell does not provide a cell edit control. The ButtonCell.EditType (EditType Property' in the on-line
documentation) property always returns a null reference (Nothing in Visual Basic).

Style

The button cell supports the following members of the CellStyle (CellStyle Class' in the on-line documentation)
class. You can set cell styles by using the ButtonCell.Style (Style Property' in the on-line documentation)
property.

<table>
<thead>
<tr>
<th>CellStyle Members</th>
<th>Enabled or Disabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>BackgroundGradientEffect</td>
<td>Enabled</td>
</tr>
<tr>
<td>Border</td>
<td>Enabled</td>
</tr>
<tr>
<td>DataSourceNullValue</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledBackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledForeColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledGradientEffect</td>
<td>-</td>
</tr>
<tr>
<td>EditingBackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>EditingForeColor</td>
<td>-</td>
</tr>
<tr>
<td>Font</td>
<td>Enabled</td>
</tr>
<tr>
<td>ForeColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>Format</td>
<td>Enabled</td>
</tr>
<tr>
<td>FormatProvider</td>
<td>Enabled</td>
</tr>
<tr>
<td>Image</td>
<td>Enabled</td>
</tr>
<tr>
<td>ImageAlign</td>
<td>Enabled</td>
</tr>
<tr>
<td>ImeMode</td>
<td>-</td>
</tr>
<tr>
<td>ImeSentenceMode</td>
<td>-</td>
</tr>
<tr>
<td>InputScope</td>
<td>-</td>
</tr>
<tr>
<td>LineAdjustment</td>
<td>Enabled only for GDI+Compatible Mode</td>
</tr>
<tr>
<td>Margin</td>
<td>Enabled</td>
</tr>
<tr>
<td>MouseOverBackColor</td>
<td>Enabled</td>
</tr>
</tbody>
</table>
To enable **GDI+ Compatible Mode**, set the `UseCompatibleTextRendering` property to True.

**Shortcut Keys**

In a button cell, Keys.Space is allocated for the button click. The rest of the keys are processed by the GcMultiRow control.

**Events**

You can use the GcMultiRow.`CellContentClick` (*CellContentClick Event* in the on-line documentation) event to implement processing for any click in the cell content area. In case of a double-click, you can use the GcMultiRow.`CellContentDoubleClick` (*CellContentDoubleClick Event* in the on-line documentation) event.

**Standard Control Comparisons**


<table>
<thead>
<tr>
<th>ButtonCell</th>
<th>Button</th>
<th>DataGridViewButtonCell</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ellipsis</td>
<td>AutoEllipsis</td>
<td>None</td>
</tr>
<tr>
<td>None</td>
<td>AutoSize</td>
<td>None</td>
</tr>
<tr>
<td>Style.BackColor</td>
<td>BackColor</td>
<td>Style.BackColor</td>
</tr>
<tr>
<td>FlatStyle</td>
<td>FlatStyle</td>
<td>FlatStyle</td>
</tr>
<tr>
<td>FlatAppearance</td>
<td>FlatAppearance</td>
<td>None</td>
</tr>
</tbody>
</table>
You can change the backcolor of the button cell by setting any color in the ButtonCell.Style.BackColor property and then using any of the following settings:

- Set the ButtonCell.FlatStyle ('FlatStyle Property' in the on-line documentation) property to Flat or Popup.
- Set the ButtonCell.FlatStyle property to Standard and ButtonCell.UseVisualStyleBackColor ('UseVisualStyleBackColor Property' in the on-line documentation) property to False.

In an environment where the visual style is enabled, the backcolor set using the ButtonCell.Style.BackColor property is not applied to the button if either the ButtonCell.FlatStyle property is set as System or the ButtonCell.UseVisualStyleBackColor property is set to True. The effect is only applied to the background of the button. The image below shows an example of a button background set by changing the ButtonCell.Style.Padding property.

**Using the Designer**

1. Select a button cell whose backcolor needs to be changed (for example: buttonCell1).
2. From Properties window, select the ButtonCell.Style.BackColor property and choose any color from the drop-down.
3. From **Properties** window, select the ButtonCell.**UseVisualStyleBackColor** property and set it to False.
4. From **Properties** window, select the ButtonCell.**FlatStyle** property and set it to either Flat, Popup, or Standard.

**Using Code**

This example sets style and colors for the button cell.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

Dim template As New Template
Dim buttonCell1 As New ButtonCell()

buttonCell1.Name = "buttonCell1"
buttonCell1.Value = "Button"
buttonCell1.Style.BackColor = Color.FromArgb(255, 192, 192)
buttonCell1.UseVisualStyleBackColor = False
buttonCell1.FlatStyle = FlatStyle.Standard

GcMultiRow1.Template = Template.CreateGridTemplate(New Cell() { buttonCell1 })
GcMultiRow1.RowCount = 10
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;

Template template = new Template();
ButtonCell buttonCell1 = new ButtonCell();

buttonCell1.Name = "buttonCell1";
buttonCell1.Value = "Button";
buttonCell1.Style.BackColor = Color.FromArgb(255, 192, 192);
buttonCell1.UseVisualStyleBackColor = false;
buttonCell1.FlatStyle = FlatStyle.Standard;

gcMultiRow1.Template = Template.CreateGridTemplate(new Cell[] { buttonCell1 });
gcMultiRow1.RowCount = 10;
```

**Handling User Click (ButtonCell)**

The GcMultiRow.**CellContentButtonClick** ("CellContentButtonClick Event" in the on-line documentation) event is fired when a button cell is clicked. You can use the GcMultiRow.**CellContentButtonClick** event to handle a button cell click.

- **VB**

```vbnet
Imports GrapeCity.Win.MultiRow

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    Dim buttonCell1 As New ButtonCell()
    buttonCell1.Name = "buttonCell1"
    buttonCell1.Value = "buttonCell1"
    Dim template1 As Template = Template.CreateGridTemplate(New Cell() {buttonCell1})
    GcMultiRow1.Template = template1
    GcMultiRow1.RowCount = 10
End Sub
```

**Using Code**

The following code provides information when the button cell placed on the row is clicked.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    Dim buttonCell1 As New ButtonCell()
    buttonCell1.Name = "buttonCell1"
    buttonCell1.Value = "buttonCell1"
    Dim template1 As Template = Template.CreateGridTemplate(New Cell() {buttonCell1})
    GcMultiRow1.Template = template1
    GcMultiRow1.RowCount = 10
End Sub
```
Display String (ButtonCell)

You can display a string in a button cell by setting the string type value in the ButtonCell.Value property (Cell.Value ('Value Property' in the on-line documentation) property).

Using the Designer

1. Add a button cell type to the row (for example: buttonCell1).
2. Select buttonCell1 and set any string in the buttonCell1.Value property through the Properties window.

Using Code

This example creates a button cell with a value.

[VB]

Imports GrapeCity.Win.MultiRow

Dim buttonCell1 As New ButtonCell()
buttonCell1.Name = "buttonCell1"
buttonCell1.Value = "Test"

GcMultiRow1.Template = Template.CreateGridTemplate(New Cell() { buttonCell1 })
GcMultiRow1.RowCount = 10

[CS]

using GrapeCity.Win.MultiRow;

ButtonCell buttonCell1 = new ButtonCell();
buttonCell1.Name = "buttonCell1";
buttonCell1.Value = "Test";
Using Button Commands (ButtonCell)

You can use the ButtonCell.ButtonCommand ('ButtonCommand Property' in the on-line documentation) property to set an action that is executed when the user clicks the button type cell.

- If a command is set in the ButtonCell.ButtonCommand property, the processing is carried out in the following order:
  1. GcMultiRow.CellContentButtonClick ('CellContentButtonClick Event' in the on-line documentation) event processing
  2. GcMultiRow.CellContentClick ('CellContentClick Event' in the on-line documentation) event processing

- If the button command of the button type cell placed in the row header section or the row footer section is run, the action is performed for the current row.

Button Commands

The button commands provide the following 15 types of actions.

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
<th>Condition in which it is not performed</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NavigationButtonCommands</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MoveToNextRow</td>
<td>Move to the row below.</td>
<td>If the current row is the last row</td>
</tr>
<tr>
<td>MoveToPreviousRow</td>
<td>Move to the row above.</td>
<td>If the current row is the first row</td>
</tr>
<tr>
<td>MoveToFirstRow</td>
<td>Move to the first row.</td>
<td>If the current row is the first row</td>
</tr>
<tr>
<td>MoveToLastRow</td>
<td>Move to the last row.</td>
<td>If the current row is the last row</td>
</tr>
<tr>
<td><strong>RowActionButtonCommands</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insert</td>
<td>Add an empty row above the current row.</td>
<td>If the GcMultiRow control is in bound mode</td>
</tr>
<tr>
<td>Remove</td>
<td>Delete the current row.</td>
<td>If the current row is a new row</td>
</tr>
<tr>
<td>Select</td>
<td>Select the current row.</td>
<td>If the current row cannot be selected</td>
</tr>
<tr>
<td>MoveUp</td>
<td>Replace with the row above.</td>
<td>If the current row is the first row</td>
</tr>
<tr>
<td>MoveDown</td>
<td>Replace with the row below.</td>
<td>If the current row is the last row</td>
</tr>
<tr>
<td>InsertCopy</td>
<td>Add a row above the current row. The added row has the same content as that in the current row.</td>
<td>If the current row is the last row</td>
</tr>
<tr>
<td>CopyRowValue</td>
<td>Copy the value of the current row to the clipboard.</td>
<td>-</td>
</tr>
<tr>
<td>ClearValues</td>
<td>Clear the value of the current row.</td>
<td>If the current row cannot be edited</td>
</tr>
<tr>
<td><strong>FormActionButtonCommands</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Print</td>
<td>Print the GcMultiRow.</td>
<td>-</td>
</tr>
</tbody>
</table>
checkBoxCell

The checkBoxCell offers features equivalent to the .NET Framework CheckBox control (System.Windows.Forms.CheckBox). The user can use this cell to enter ON and OFF values. The size of the checkBox depends upon Windows settings.

Features

In addition to the Base Cell features, the following features can also be used in a checkBox cell.

- CheckBox Display and Input
- Get and Set ON or OFF value
- Get and Set ON, OFF, or Indeterminate value
- Set ON, OFF, or Indeterminate value
- Display Text

For more details on each feature, refer to the CheckBoxCell ('CheckBoxCell Class' in the on-line documentation) class.

Data Type

The checkBoxCell uses the value object type and this type can be checked with the CheckBoxCell.ValueType ('Value-Type Property' in the on-line documentation) property. The datatype used for input and display differs based on the value in the CheckBoxCell.ThreeState (ThreeState Property' in the on-line documentation) property. The value cast into the Boolean type is used when the ThreeState property is set to False. When the ThreeState property is True, the value cast into the System.Windows.Forms.CheckState type is used. These types can be checked with the CheckBoxCell.FormattedValueType (FormattedValueType Property' in the on-line documentation) property. The behavior during casting can be modified by overriding the ToString method of the Object type. To modify the behavior when values are read in a cell, you can override the CheckBoxCell.OnCellFormatting ('OnCellFormatting Method' in the on-line documentation) method. The behavior when the value is written back from a cell can be modified by overriding the CheckBoxCell.OnCellParsing (OnCellParsing Method' in the on-line documentation) method.

Cell Edit Control

The checkBoxCell does not provide a cell edit control. The CheckBoxCell.EditType (EditType Property' in the on-line documentation) property always returns a null reference (Nothing in Visual Basic).

Style

The checkBoxCell supports the following members of theCellStyle (CellStyle Class' in the on-line documentation) class. You can set a cell style using the CheckBoxCell.Style (Style Property' in the on-line documentation) property.

CellStyle Members	| Enabled or Disabled
---|---
BackColor	| Enabled
BackgroundGradientEffect	| Enabled
Border	| Enabled
DataSourceNullValue  Enabled
DisabledBackColor  Enabled
DisabledForeColor  Enabled
DisabledGradientEffect  Enabled
EditingBackColor  -
EditingForeColor  -
Font  Enabled
ForeColor  Enabled
Format  Enabled
FormatProvider  Enabled
Image  Enabled
ImageAlign  Enabled
ImeMode  -
ImeSentenceMode  -
InputScope  -
LineAdjustment  Enabled only for GDI+ Compatible Mode
Margin  Enabled
MouseOverBackColor  Enabled
MouseOverForeColor  Enabled
MouseOverGradientEffect  Enabled
Multiline  Enabled
NullValue  Enabled
Padding  Enabled
PatternColor  Enabled
PatternStyle  Enabled
SelectionBackColor  Enabled
SelectionForeColor  Enabled
SelectionGradientEffect  Enabled
Tag  Enabled
TextAdjustment  Enabled only for GDI+ Compatible Mode
 TextAlign  Enabled
TextAngle  Enabled only for GDI+ Compatible Mode
TextEffect  Enabled
TextImageRelation  Enabled
TextIndent  Enabled
TextVertical  Enabled only for GDI+ Compatible Mode
UseCompatibleTextRendering  Enabled
To set **GDI+ Compatibility Mode** as Enabled, set the UseCompatibleTextRendering property to True.

### Shortcut Keys

In a check box cell, Keys.Space is allocated for changing the check state and Keys.Control + Keys.C is allocated for clipboard processing. Keys other than these are processed by GcMultiRow.

### Events

You can use the GcMultiRow.CellContentClick ('CellContentClick Event' in the on-line documentation) event to implement the processing for any click in the cell content area. You can use the GcMultiRow.CellContentDoubleClick ('CellContentDoubleClick Event' in the on-line documentation) event to handle a double-click. Any change in the cell value fires a GcMultiRow.CellEditedFormattedValueChanged ('CellEditedFormattedValueChanged Event' in the on-line documentation) event.

### Standard Control Comparison

The table below shows comparisons between major properties of the check box cell, the System.Windows.Forms.CheckBox control, and the System.Windows.Forms.DataGridViewCheckBoxCell.

<table>
<thead>
<tr>
<th>CheckBoxCell</th>
<th>CheckBox</th>
<th>DataGridViewCheckBoxCell</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appearance</td>
<td>Appearance</td>
<td>None</td>
</tr>
<tr>
<td>CheckAlign</td>
<td>CheckAlign</td>
<td>Style.Alignment</td>
</tr>
<tr>
<td>FalseValue</td>
<td>None</td>
<td>FalseValue</td>
</tr>
<tr>
<td>FlatAppearance</td>
<td>FlatAppearance</td>
<td>None</td>
</tr>
<tr>
<td>FlatStyle</td>
<td>FlatStyle</td>
<td>FlatStyle</td>
</tr>
<tr>
<td>IndeterminateValue</td>
<td>None</td>
<td>IndeterminateValue</td>
</tr>
<tr>
<td>Text</td>
<td>Text</td>
<td>None</td>
</tr>
<tr>
<td>ThreeState</td>
<td>ThreeState</td>
<td>ThreeState</td>
</tr>
<tr>
<td>TrueValue</td>
<td>None</td>
<td>TrueValue</td>
</tr>
<tr>
<td>None</td>
<td>AutoCheck</td>
<td>None</td>
</tr>
<tr>
<td>None</td>
<td>Checked</td>
<td>None</td>
</tr>
<tr>
<td>None</td>
<td>CheckState</td>
<td>None</td>
</tr>
<tr>
<td>Style.TextAlign</td>
<td>TextAlign</td>
<td>None</td>
</tr>
</tbody>
</table>

Refer to the following for more information:

- **Detect Value Change (CheckBoxCell)**
- **Refer to Editing Value when Modifying TrueValue and FalseValue (CheckBoxCell)**

Refer to Editing Value when Modifying TrueValue and FalseValue (CheckBoxCell)

You can use the Cell.EditedFormattedValue ('EditedFormattedValue Property' in the on-line documentation) property for the check box cell and get the cell value being edited; however, if you are editing the TrueValue ('TrueValue Property' in the on-line documentation) and FalseValue ('FalseValue Property' in the on-line documentation) properties in the check box cell, then its value cannot be returned by the CheckBoxCell.EditedFormattedValue property. This is because the CheckBoxCell.EditedFormattedValue property is formatted using the CheckBoxCell.FormattedValueType type. To refer to the value of the TrueValue and FalseValue properties of the check box cell, you can directly refer to the respective property based on the value in the CheckBoxCell.EditedFormattedValue property.

### Using Code

This example creates a check box cell and gets the value being edited.

```vbnet
Imports GrapeCity.Win.MultiRow
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    Dim checkBoxCell1 As New CheckBoxCell()
    checkBoxCell1.TrueValue = "Yes"
    checkBoxCell1.FalseValue = "No"
    GcMultiRow1.Template = Template.CreateGridTemplate(New Cell() {checkBoxCell1})
End Sub
```
Detect Value Change (CheckBoxCell)

You can use the GcMultiRow.CellEditedFormattedValueChanged (CellEditedFormattedValueChanged Event in the on-line documentation) event to detect changes in the checked status of the check box cell. The value being edited can be returned by the Cell.EditedFormattedValue (EditedFormattedValue Property in the on-line documentation) property.

Using Code

This code creates a test template.

[VB]
Imports GrapeCity.WinForms.MultiRow
Dim checkBoxCell1 As New CheckBoxCell()
checkBoxCell1.Text = "Test"
GcMultiRow1.Template = Template.CreateGridTemplate(New Cell() {checkBoxCell1})

[CS]
using GrapeCity.WinForms.MultiRow;
CheckBoxCell checkBoxCell1 = new CheckBoxCell();
checkBoxCell1.Text = "Test";
GcMultiRow1.Template = Template.CreateGridTemplate(new Cell[] { checkBoxCell1 });

This code shows the current value when the checkbox status has been changed by the user.

[VB]
Imports GrapeCity.WinForms.MultiRow
Private Sub GcMultiRow1_CellEditedFormattedValueChanged(ByVal ByVal sender As System.Object, ByVal e As CellEditedFormattedValueChangedEventArgs) Handles GcMultiRow1.CellEditedFormattedValueChanged
    Dim currentCell As Cell = gMultiRow1.Rows(e.RowIndex).Cells(e.CellIndex)
    If e.Scope = CellScope.Row Then
        If TypeOf currentCell Is CheckBoxCell Then
            Console.WriteLine("Cell value being edited: "{0}", currentCell.EditedFormattedValue)
            Console.WriteLine("Cell Value: "{0}", currentCell.Value)
        End If
    End If
End Sub

[CS]
using GrapeCity.WinForms.MultiRow;
private void gcMultiRow1_CellEditedFormattedValueChanged(object sender, CellEditedFormattedValueChangedEventArgs e)
{
    Cell currentCell = gcMultiRow1.Rows[e.RowIndex].Cells[e.RowIndex];
    if (e.Scope == CellScope.Row)
    {
        if (currentCell is CheckBoxCell)
        {
            Console.WriteLine("Cell value being edited: "{0}", currentCell.EditedFormattedValue);
            Console.WriteLine("Cell Value: "{0}", currentCell.Value);
        }
    }
}

ComboBoxCell

The combo box cell provides features similar to the .NET Framework's ComboBox control.
(System.Windows.Forms.ComboBox). The user can use this combo box cell to enter a value from the drop-down list. The height of the combo box is automatically adjusted according to cell height.

![ComboBox](Inventory.png)

**Features**

In addition to the **Base Cell** features, the following features can also be used in the combo box cell.

- Display Drop-down button
- Display Drop-down list
- Bind to Data Source
- Events

For more details on each feature, refer to **ComboBoxCell (ComboBoxCell Class' in the on-line documentation)** class.

**Data Type**

The combo box cell uses the value Object type. The ComboBoxCell.ValueType (Value Type Property' in the on-line documentation) property can be used to check the type. For input and display, a value cast into a String type is used, and this type can be checked with the ComboBoxCell.FormattedValueType (Formatted Value Type Property' in the on-line documentation) property. The behavior during casting can be modified by overriding the ToString method of the Object type. To modify the behavior when values are read in a cell, you can override the ComboBoxCell.OnCellFormatting (OnCellFormatting Method' in the on-line documentation) method. The behavior when the value is written back from the cell can be modified by overriding the ComboBoxCell.OnCellParsing (OnCellParsing Method' in the on-line documentation) method.

**Cell Edit Control**

The combo box cell value is edited in the **ComboBoxEditingControl (ComboBoxEditingControl Class' in the on-line documentation)**. This control is inherited from the IEditingControl Interface and System.Windows.Forms.ComboBox class. The type of the cell edit control can be checked with the Cell.EditType (Edit Type Property' in the on-line documentation) property.

**Style**

The combo box cell supports the following **CellStyle (CellStyle Class' in the on-line documentation)** class members. The cell style is set with the ComboBoxCell.Style (Style Property' in the on-line documentation) property.

<table>
<thead>
<tr>
<th>CellStyle Member</th>
<th>Non-Editable State</th>
<th>Editable State</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>BackgroundGradientEffect</td>
<td>Enabled only when ComboBoxCell.FlatStyle is Flat or Popup.</td>
<td>-</td>
</tr>
<tr>
<td>Border</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>DataSourceNullValue</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledBackColor</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>DisabledForeColor</td>
<td>Enabled only when ComboBoxCell.FlatStyle is Flat or Popup</td>
<td>-</td>
</tr>
<tr>
<td>EditingBackColor</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td>EditingForeColor</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td>Font</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>ForeColor</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
</tbody>
</table>
To enable the **GDI+ Compatible Mode**, set the `UseCompatibleTextRendering` property to `True`.

### Shortcut Keys

The table below shows the list of keys processed when editing the combo box cell and the keys processed by the GcMultiRow control.

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Key</th>
<th>ComboBoxCell</th>
<th>GcMultiRow</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Keys.PageUp</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.PageDown</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>Modifier</td>
<td>Key</td>
<td>TextBoxCell</td>
<td>GcMultiRow</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------</td>
<td>-------------</td>
<td>------------</td>
</tr>
<tr>
<td>None</td>
<td>Keys.Delete</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.BackSpace</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Keys.Control</td>
<td>Keys.Left</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Keys.Right</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Keys.Up</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Keys.Down</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td>Keys.Shift</td>
<td>Keys.Left</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Keys.Right</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Keys.Up</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Keys.Down</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td>Keys.Alt</td>
<td>Keys.Up</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Down</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

When multiple cells are selected using Keys.Control + Keys.C, they are processed by the GcMultiRow control. The following keys are either not processed or are processed by the GcMultiRow control, if the ComboBoxCell.DropDownStyle property is set to MultiRowComboBoxStyle DropDownList.
Events

You can use the GcMultiRow.CellContentClick ('CellContentClick Event' in the on-line documentation) event to get a click in the cell content area. You can use the GcMultiRow.CellContentDoubleClick ('CellContentDoubleClick Event' in the on-line documentation) event for a double-click. Use the GcMultiRow.CellEditedFormattedValueChanged ('CellEditedFormattedValueChanged Event' in the on-line documentation) event to get any changes to the cell value. The ComboBoxEditingControl class events can be used to handle the processing of events when the combo box cell is being edited.

For more information see the following:

- Get Index Of Selected Value (ComboBoxCell)
- How to Use SelectedIndexChanged Event (ComboBoxCell)

Get Index Of Selected Value (ComboBoxCell)

Search the cell value (Cell.Value) in the ComboBox.Items property collection to get the index of the value selected in the combo box cell.

Using Code

This example creates a combo box cell and searches for the selected item.

[VB]

```vbnet
Imports GrapeCity.Win.MultiRow

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    Dim comboBoxCell1 As New ComboBoxCell
    comboBoxCell1.Name = "comboBoxCell1"
    comboBoxCell1.Items.Add("Tokyo")
    comboBoxCell1.Items.Add("Nagoya")
    comboBoxCell1.Items.Add("Osaka")
    GcMultiRow1.Template = Template.CreateGridTemplate(New Cell() {comboBoxCell1})
    GcMultiRow1.SetValue(0, "comboBoxCell1", "Nagoya")
End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    Dim gcMultiRow As GcMultiRow = Me.GcMultiRow1
    If TypeOf gcMultiRow.CurrentCell Is ComboBoxCell Then
        Dim comboBoxCell As ComboBoxCell = TryCast(gcMultiRow.CurrentCell, ComboBoxCell)
        Dim selectedValue As Object = comboBoxCell.Value
        Dim selectedIndex As Integer = -1
        If Not selectedValue = Nothing Then
            selectedIndex = comboBoxCell.Items.IndexOf(selectedValue)
        End If
        MessageBox.Show(String.Format("Selected Index: {0}", selectedIndex))
    Else
        MessageBox.Show("Present cell is not ComboBoxCell.")
    End If
End Sub
```

[CS]

```cs
using GrapeCity.Win.MultiRow;

private void Form1_Load(object sender, EventArgs e)
{
    ComboBoxCell comboBoxCell1 = new ComboBoxCell();
    comboBoxCell1.Name = "comboBoxCell1";
    comboBoxCell1.Items.Add("Tokyo");
```

Copyright © GrapeCity, inc. All rights reserved.
How to Use SelectedIndexChanged Event (ComboBoxCell)

In the combo box cell, there are two ways to perform processing (similar to the ComboBox.SelectedIndexChanged event). One is by implementing the GcMultiRow.CellEditedFormattedValueChanged event and another is by using the ComboBoxEditingControl.SelectedIndexChanged event. Implementation through the GcMultiRow.CellEditedFormattedValueChanged event is recommended.

Using Code

This example uses the CellEditedFormattedValueChanged event.

[Vb]
```vbnet
Imports GrapeCity.Win.MultiRow
    Dim gcMultiRow As GcMultiRow = TryCast(sender, GcMultiRow)
    Dim currentCell As Cell = gcMultiRow.Rows(e.RowIndex).Cells(e.CellIndex)
    If TypeOf currentCell Is ComboBoxCell Then
        Console.WriteLine("{0},{1}", e.Reason.ToString(), currentCell.EditedFormattedValue)
    End If
End Sub
```

[Cs]
```csharp
using GrapeCity.Win.MultiRow;
{
    GcMultiRow gcMultiRow = sender as GcMultiRow;
    Cell currentCell = gcMultiRow.Rows[e.RowIndex].Cells[e.CellIndex];
    if (currentCell is ComboBoxCell)
    {
        Console.WriteLine("{0},{1}", e.Reason.ToString(), currentCell.EditedFormattedValue);
    }
}
```

When using the ComboBoxEditingControl.SelectedIndexChanged event, you will need to use an implementation that is different from the TextBoxEditingControl.TextChanged event, in order to avoid catching events that are fired during cell initialization.

Using Code

This example uses the SelectedIndexChanged event.

[Vb]
```vbnet
Imports GrapeCity.Win.MultiRow
Private _comboBoxEditingControl1 As ComboBoxEditingControl = Nothing
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    Dim comboBoxCell1 As New ComboBoxCell()
    comboBoxCell1.Name = "ComboBoxCell1"
    comboBoxCell1.Items.AddRange("A", "B", "C")
    Dim labelCell1 As New LabelCell()
    labelCell1.Name = "LabelCell1"
    Dim labelCell2 As New LabelCell()
    labelCell2.Name = "LabelCell2"
    GcMultiRow1.Template = Template.CreateGridTemplate(New Cell() {comboBoxCell1, labelCell1, labelCell2})
End Sub
Private Sub GcMultiRow1_EditingControlShowing(ByVal sender As System.Object, ByVal e As GrapeCity.Win.MultiRow.EditingControlShowingEventArgs) Handles GcMultiRow1.EditingControlShowing
    ' Adding cell edit control event.
    If TypeOf e.Control Is ComboBoxEditingControl Then
        Me._comboBoxEditingControl1 = TryCast(e.Control, ComboBoxEditingControl)
        AddHandler Me._comboBoxEditingControl1.SelectedIndexChanged, AddressOf Me._comboBoxEditingControl1_SelectedIndexChanged
    End If
End Sub
Private Sub _comboBoxEditingControl1_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Dim comboBox As ComboBoxEditingControl = TryCast(sender, ComboBoxEditingControl)
    comboBox.GcMultiRow.CurrentRow.Cells("LabelCell2").Value = comboBox.SelectedIndex
End Sub
    ' Delete the event of cell edit control.
    If Me._comboBoxEditingControl1 Is Nothing Then
        RemoveHandler Me._comboBoxEditingControl1.SelectedIndexChanged, AddressOf Me._comboBoxEditingControl1_SelectedIndexChanged
    End If
End Sub
```

[Cs]
```csharp
using GrapeCity.Win.MultiRow;
private void comboBoxEditingControl1_SelectedIndexChanged(object sender, EventArgs e)
{
    ComboBoxEditingControl comboBox = sender as ComboBoxEditingControl;
    comboBox.GcMultiRow.CurrentRow.Cells["LabelCell2"].Value = comboBox.SelectedIndex;
}
```

Copyright © GrapeCity, inc. All rights reserved.
using GrapeCity.WinForms;

private ComboBoxEditingControl _comboBoxEditingControl1 = null;
private void Form1_Load(object sender, System.EventArgs e)
{
    ComboBoxCell comboBoxCell1 = new ComboBoxCell();
    comboBoxCell1.Name = "comboBoxCell1";
    comboBoxCell1.Items.AddRange("A", "B", "C");
    LabelCell labelCell1 = new LabelCell();
    labelCell1.Name = "labelCell1";
    LabelCell labelCell2 = new LabelCell();
    labelCell2.Name = "labelCell2";
    GcMultiRow1.Template = Template.CreateGridTemplate(new Cell[] {comboBoxCell1, labelCell1, labelCell2});
}
private void gcMultiRow1_EditingControlShowing(object sender, GrapeCity.WinForms.MultiRow.EditingControlShowingEventArgs e)
{
    // Adding cell edit control event.
    if (e.Control is ComboBoxEditingControl)
    {
        this._comboBoxEditingControl1 = e.Control as ComboBoxEditingControl;
        this._comboBoxEditingControl1.SelectedIndexChanged += new EventHandler(_comboBoxEditingControl1_SelectedIndexChanged);
    }
}
private void _comboBoxEditingControl1_SelectedIndexChanged(object sender, EventArgs e)
{
    ComboBoxEditingControl comboBox = sender as ComboBoxEditingControl;
    comboBox.GcMultiRow.CurrentRow.Cells["labelCell1"].Value = comboBox.Items[comboBox.SelectedIndex].ToString();
    comboBox.GcMultiRow.CurrentRow.Cells["labelCell2"].Value = comboBox.SelectedIndex;
}
{
    // Delete the event of cell edit control
    if (this._comboBoxEditingControl1 != null)
    {
        this._comboBoxEditingControl1.SelectedIndexChanged -= new EventHandler(_comboBoxEditingControl1_SelectedIndexChanged);
        this._comboBoxEditingControl1 = null;
    }
}

MaskedTextBoxCell

The masked text box cell offers features equivalent to the .NET Framework’s MaskedTextBox control (System.Windows.Forms.MaskedTextBox). You can use the masked text box cell to provide the user with formatted input.

Features

In addition to the Base Cell features, the following features can also be used in the masked text box cell.

- Input Formatted String
- Display Literals
- Enter Password

For more details on each feature, refer to the MaskedTextBoxCell (‘MaskedTextBoxCell Class’ in the on-line documentation) class.

Data Types

The masked text box cell uses the Object type value and this type can be checked with the MaskedTextBoxCell.GetValueType (‘ValueType Property’ in the on-line documentation) property. The value used for input and display is the one cast into the String type based on the MaskedTextBoxCell.Mask (‘Mask Property’ in the on-line documentation) property format. This type can be checked with the MaskedTextBoxCell.FormattedValueType (‘FormattedValueType Property’ in the on-line documentation) property. To modify the behavior when values are read in a cell, you can override the MaskedTextBoxCell.OnCellFormatting (‘OnCellFormatting Method’ in the on-line documentation) method. Override the MaskedTextBoxCell.OnCellParsing (‘OnCellParsing Method’ in the on-line documentation) method to modify behavior when the value is written back from the cell.

Cell Edit Control

The masked text box cell value can be edited in the MaskedTextBoxEditingControl control. This control inherits the IEditingControl interface and System.Windows.Forms.MaskedTextBox class. The cell edit control type can be checked...
with the MaskedTextBoxCell.`EditType` (*EditType Property* in the on-line documentation) property.

**Styles**

The masked text box cell supports the following `CellStyle` (*CellStyle Class* in the on-line documentation) class members. You can set the cell style using the MaskedTextBoxCell.`Style` (*Style Property* in the on-line documentation) property.

### CellStyle Members

| BackColor | Enabled | Font | Enabled |
| Border | Enabled | Font | Enabled |
| DataSourceNullValue | Enabled | Format | Enabled |
| DisabledBackColor | Enabled | FormatProvider | Enabled |
| DisabledForeColor | Enabled | GradientColors | Enabled |
| Font | Enabled | GradientDirection | - |
| ForeColor | Enabled | GradientStyle | - |
| Format | Enabled | Image | - |
| FormatProvider | Enabled | ImageAlign | - |
| GradientColors | Enabled | ImeMode | Enabled |
| GradientDirection | Enabled | ImeSentenceMode | Enabled |
| GradientStyle | Enabled | InputScope | Enabled |
| Image | Enabled | LineAdjustment | Enabled only for GDI+ Compatible Mode |
| ImageAlign | Enabled | Margin | Enabled |
| ImeMode | Enabled | Multiline | - |
| ImeSentenceMode | Enabled | NullValue | Enabled |
| InputScope | Enabled | Padding | Enabled |
| LineAdjustment | Enabled only for GDI+ Compatible Mode | PatternColor | Enabled |
| Margin | Enabled | PatternStyle | - |
| Multiline | Enabled | SelectionBackColor | Enabled |
| NullValue | Enabled | SelectionForeColor | - |
| Padding | Enabled | Tag | Enabled |
| PatternColor | Enabled | TextAdjustment | Enabled only for GDI+ Compatible Mode |
| PatternStyle | Enabled | TextAlign | Enabled only for horizontal |
| SelectionBackColor | Enabled | TextAngle | Enabled only for GDI+ Compatible Mode |
| SelectionForeColor | Enabled | TextEffect | - |

"MultiRow Windows Forms Developer's Guide" 232
Copyright © GrapeCity, inc. All rights reserved.
To set **GDI+ Compatible Mode** as Enabled, set the MaskedTextBoxCell.Style.UseCompatibleTextRendering property to True.

**Shortcut Keys**

The table below lists the keys processed when the masked text box cell is in an edit state and lists keys that are processed by the GcMultiRow control.

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Key</th>
<th>MaskedTextBoxCell</th>
<th>GcMultiRow</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Keys.PageUp</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.PageDown</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.End</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Home</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Left</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Right</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Up</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Down</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Insert</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Delete</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.BackSpace</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>Keys.Control</td>
<td>Keys.PageUp</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.PageDown</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.End</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.Home</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.Left</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Right</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Up</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.Down</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.A</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.C</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.V</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.X</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>Keys.Shift</td>
<td>Keys.Left</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Right</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Up</td>
<td>Enabled</td>
<td>-</td>
</tr>
</tbody>
</table>
Keys.Down Enabled
Keys.Home Enabled
Keys.End Enabled

When multiple cells are selected, Keys.Control + Keys.C is processed by the GcMultiRow control.

**Events**

You can use the GcMultiRow.CellContentClick ('CellContentClick Event' in the on-line documentation) event to get a click in the cell content area. Use the GcMultiRow.CellContentDoubleClick ('CellContentDoubleClick Event' in the on-line documentation) event for a double-click. Use the GcMultiRow.CellEditedFormattedValueChanged ('CellEditedFormattedValueChanged Event' in the on-line documentation) event to handle changes to the cell. MaskedTextBoxEditingControl class events can be used to handle processing when the masked text box cell is being edited.

**Comparison with Standard Control**

The table below shows a comparison between major properties of the masked text box cell and the System.Windows.Forms.MaskedTextBox control.

<table>
<thead>
<tr>
<th>MaskedTextBoxCell</th>
<th>MaskedTextBox</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllowPromptAsInput</td>
<td>AllowPromptAsInput</td>
</tr>
<tr>
<td>AsciiOnly</td>
<td>AsciiOnly</td>
</tr>
<tr>
<td>BeepOnError</td>
<td>BeepOnError</td>
</tr>
<tr>
<td>Culture</td>
<td>Culture</td>
</tr>
<tr>
<td>CutCopyMaskFormat</td>
<td>CutCopyMaskFormat</td>
</tr>
<tr>
<td>Style.FormatProvider</td>
<td>FormatProvider</td>
</tr>
<tr>
<td>DisplayTextMaskFormat</td>
<td>None</td>
</tr>
<tr>
<td>None</td>
<td>HidePromptOnLeave</td>
</tr>
<tr>
<td>InsertKeyMode</td>
<td>InsertKeyMode</td>
</tr>
<tr>
<td>None</td>
<td>IsOverwriteMode</td>
</tr>
<tr>
<td>Mask</td>
<td>Mask</td>
</tr>
<tr>
<td>MaskCompleted</td>
<td>MaskCompleted</td>
</tr>
<tr>
<td>MaskedTextBoxProvider</td>
<td>MaskedTextBoxProvider</td>
</tr>
<tr>
<td>MaskFull</td>
<td>MaskFull</td>
</tr>
<tr>
<td>PasswordChar</td>
<td>PasswordChar</td>
</tr>
<tr>
<td>PromptChar</td>
<td>PromptChar</td>
</tr>
<tr>
<td>RejectInputOnFirstFailure</td>
<td>RejectInputOnFirstFailure</td>
</tr>
<tr>
<td>ResetOnPrompt</td>
<td>ResetOnPrompt</td>
</tr>
<tr>
<td>ResetOnSpace</td>
<td>ResetOnSpace</td>
</tr>
<tr>
<td>None</td>
<td>SelectedText</td>
</tr>
<tr>
<td>SkipLiterals</td>
<td>SkipLiterals</td>
</tr>
<tr>
<td>DisplayText</td>
<td>Text</td>
</tr>
</tbody>
</table>
DateTimePickerCell

The date time picker cell provides features similar to that of the DateTimePicker control (System.Windows.Forms.DateTimePicker) in the .NET Framework. The user can enter dates and times using the date time picker cell. The appearance of the date time picker cell depends on Windows settings.

Features

The date time picker cell provides the following features in addition to the Base Cell features.

- Input of date and time
- Display of drop-down calendar
- Specifying input range for date

Refer to the DateTimePickerCell ('DateTimePickerCell Class' in the on-line documentation) class reference for details on each of these features.

Data Type

The date time picker cell stores Object type values. This type can be verified using the DateTimePickerCell.ValueType ('Value Type Property' in the on-line documentation) property. Values transformed into a Date type (System.DateTime type in case of C#) based on the format of the DateTimePickerCell.Format ('Format Property' in the on-line documentation) property are used for input and display. This type can be verified using the DateTimePickerCell.FormattedValueType ('FormattedValueType Property' in the on-line documentation). To modify the behavior when values are read in a cell, you can override the DateTimePickerCell.OnCellFormatting ('OnCellFormatting Method' in the on-line documentation) method. The behavior when the value is written back from the cell can be modified by overriding the DateTimePickerCell.OnCellParsing ('OnCellParsing Method' in the on-line documentation) method.

Cell Edit Control

The date time picker cell values can be edited using the DateTimePickerEditingControl ('DateTimePickerEditingControl Class' in the on-line documentation) control. This control inherits the IEditingControl ('IEditingControl Interface' in the on-line documentation) interface and System.Windows.Forms.DateTimePicker class. The cell editing control type can be verified using the DateTimePickerCell.EditType ('EditType Property' in the on-line documentation) property.

Styles

The date time picker cell supports the following CellStyle ('CellStyle Class' in the on-line documentation) class members. You can set cell styles using the DateTimePickerCell.Style ('Style Property' in the on-line documentation) property.

<table>
<thead>
<tr>
<th>CellStyle Member</th>
<th>Non-editable state</th>
<th>Editable state</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>Enabled</td>
<td>-</td>
</tr>
</tbody>
</table>
Border | Enabled | Enabled
--- | --- | ---
DataSourceNullValue | Enabled | Enabled
DisabledBackColor | Enabled | -
DisabledForeColor | Enabled | -
Font | Enabled | Enabled
ForeColor | Enabled | -
Format | - | -
FormatProvider | - | -
GradientColors | Enabled | -
GradientDirection | Enabled | -
GradientStyle | Enabled | -
Image | Enabled | -
ImageAlign | Enabled | -
ImeMode | Enabled | Enabled
InputScope | Enabled | Enabled
ImeSentenceMode | Enabled | Enabled
LineAdjustment | Enabled only in GDI+ compatibility mode | -
Margin | Enabled | Enabled
Multiline | Enabled | Enabled
NullValue | Enabled | Enabled
Padding | Enabled | Enabled
PatternColor | Enabled | -
PatternStyle | Enabled | -
SelectionBackColor | Enabled | -
SelectionForeColor | Enabled | -
Tag | Enabled | Enabled
TextAdjustment | Enabled only in GDI+ compatibility mode | -
TextAlign | Enabled | Only for vertical direction
TextAngle | Enabled only in GDI+ compatibility mode | -
TextEffect | Enabled | -
TextImageRelation | Enabled | -
TextIndent | Enabled | -
TextVertical | Enabled only in GDI+ compatibility mode | -
UseCompatibleTextRendering | Enabled | -
WordWrap | Enabled | -

The **GDI+ Compatibility Mode** can be enabled by setting the `UseCompatibleTextRendering` property to True.

**Shortcut Keys**
The table below lists all the supported keys when editing the date time picker cell and lists keys processed by the GcMultiRow control.

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Key</th>
<th>DateTimePickerCell</th>
<th>GcMultiRow</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Keys/PageUp</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys/PageDown</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys/End</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys/Home</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys/Left</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys/Right</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys/Up</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys/Down</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys/Insert</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys/Delete</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys/BackSpace</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys/F4</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>Keys/Control</td>
<td>Keys/PageUp</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys/PageDown</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys/End</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys/Home</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys/Left</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys/Right</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys/Up</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys/Down</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys/A</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys/C</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys/V</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys/X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Keys/Shift</td>
<td>Keys/Left</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys/Right</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys/Up</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys/Down</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys/Home</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys/End</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>Keys/Alt</td>
<td>Keys/Up</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys/Down</td>
<td>Enabled</td>
<td>-</td>
</tr>
</tbody>
</table>

*Keys.Control + Keys.C is processed by the GcMultiRow control in case multiple cells have been selected.*
The following keys are not processed when the DateTimePicker.ShowUpDown property is set to True.

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Key</th>
<th>DateTimePickerCell</th>
<th>GcMultiRow</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Keys.F4</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Keys.Alt</td>
<td>Keys.Up</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Down</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Events**

The GcMultiRow.CellContentClick ('CellContentClick Event' in the on-line documentation) event can be used to implement the processing when clicking in the content area of the cell. The GcMultiRow.CellContentDoubleClick ('CellContentDoubleClick Event' in the on-line documentation) event can be used for a double-click. The GcMultiRow.CellEditedFormattedValueChanged ('CellEditedFormattedValueChanged Event' in the on-line documentation) event can be used to implement the processing when the cell value is changed. Events in the DateTimePickerEditingControl class can be used to handle the processing for events when the date time picker cell is being edited. The same set of events used for the System.Windows.Forms.DateTimePicker class can be used for the DateTimePickerEditingControl class.

**Standard Control Comparison**

The following table shows a comparison between the main properties of the date time picker cell and the System.Windows.Forms.DateTimePicker control.

<table>
<thead>
<tr>
<th>DateTimePickerCell</th>
<th>DateTimePicker</th>
</tr>
</thead>
<tbody>
<tr>
<td>CalendarFont</td>
<td>CalendarFont</td>
</tr>
<tr>
<td>CalendarForeColor</td>
<td>CalendarForeColor</td>
</tr>
<tr>
<td>CalendarMonthBackground</td>
<td>CalendarMonthBackground</td>
</tr>
<tr>
<td>CalendarTitleBackColor</td>
<td>CalendarTitleBackColor</td>
</tr>
<tr>
<td>CalendarTitleForeColor</td>
<td>CalendarTitleForeColor</td>
</tr>
<tr>
<td>CalendarTrailingForeColor</td>
<td>CalendarTrailingForeColor</td>
</tr>
<tr>
<td>CustomFormat</td>
<td>CustomFormat</td>
</tr>
<tr>
<td>DropDownListAlign</td>
<td>DropDownListAlign</td>
</tr>
<tr>
<td>Format</td>
<td>Format</td>
</tr>
<tr>
<td>MaxDate</td>
<td>MaxDate</td>
</tr>
<tr>
<td>MinDate</td>
<td>MinDate</td>
</tr>
<tr>
<td>ShowUpDown</td>
<td>ShowUpDown</td>
</tr>
<tr>
<td>ShowDropDownButtonText</td>
<td>None</td>
</tr>
<tr>
<td>None</td>
<td>PreferredHeight</td>
</tr>
<tr>
<td>None</td>
<td>RightToLeftLayout</td>
</tr>
<tr>
<td>None</td>
<td>ShowCheckBox</td>
</tr>
<tr>
<td>None</td>
<td>Checked</td>
</tr>
<tr>
<td>DisplayText</td>
<td>Text</td>
</tr>
<tr>
<td>Value</td>
<td>Value</td>
</tr>
</tbody>
</table>

See the following for more information:
• Automatically Confirm Initial Value as Input Value (DateTimePickerCell)
• Get Formatted Values (DateTimePickerCell)
• Usage of ValueChanged Event (DateTimePickerCell)
• Display Alternate Text for Null Values (DateTimePickerCell)
• Hide Drop-Down Button (DateTimePickerCell)

Display Alternate Text for Null Values (DateTimePickerCell)

You can show alternate text when the date time picker cell value is null (Nothing in case of Visual Basic) by inheriting or using a custom cell.

If you render the GcMultiRow control using owner draw, you need to add separate processing to render the drop-down button (since you cannot render it before that).

Using Code

The following code creates a user-defined cell, MyDateTimePickerCell, that inherits the date time picker cell.

[VB]

Imports System.Windows.Forms
Imports GrapeCity.Win.MultiRow

Public Class MyDateTimePickerCell
    Inherits DateTimePickerCell
    Protected Overrides Sub OnPaint(ByVal e As CellPaintingEventArgs)
        MyBase.OnPaint(e)
        If e.Value Is Nothing Then
            Dim placeHolder As String = "(None)"
            If e.Enabled = False Then brushColor = e.CellStyle.DisabledForeColor
            Using brush As New SolidBrush(brushColor)
                Dim contentSize As SizeF = e.Graphics.MeasureString(placeHolder, e.CellStyle.Font)
            End Using
        End If
    End Sub
End Class

[CS]

using System.Drawing;
using GrapeCity.Win.MultiRow;

public class MyDateTimePickerCell : DateTimePickerCell
{
    protected override void OnPaint(CellPaintingEventArgs e)
    {
        base.OnPaint(e);
        if (e.Value != null)
        {
            string placeHolder = "(None)";
            if (e.Enabled == false)
                brushColor = e.CellStyle.DisabledForeColor;
            using (Brush brush = new SolidBrush(brushColor))
            {
                SizeF contentSize = e.Graphics.MeasureString(placeHolder, e.CellStyle.Font);
                float y = e.CellBounds.Y + ((e.CellBounds.Height - contentSize.Height) / 2);
            }
        }
    }
}

Using the Designer

Use the following steps to use this cell in the designer:
1. Save the above code in a file such as MyDateTimePickerCell.vb or MyDateTimePickerCell.cs.
2. Add the file to the project.
3. Build the project.
4. Open the template designer.
5. Select ‘MyDateTimePickerCell’ from the toolbox and place it on the designer by dragging it.

Using Code
The following code creates a user-defined cell by inheriting the date time picker cell.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

Private Sub GcMultiRow1_CellPainting(ByVal sender As System.Object, ByVal e As CellPaintingEventArgs) Handles GcMultiRow1.CellPainting
    Dim gcMultiRow As GcMultiRow = TryCast(sender, GcMultiRow)
    If TypeOf gcMultiRow.Rows(e.RowIndex).Cells(e.CellIndex) Is DateTimePickerCell Then
        Dim placeHolder As String = "(None)"
        Using brush As New SolidBrush(brushColor)
            Dim contentSize As SizeF = e.Graphics.MeasureString(placeHolder, e.CellStyle.Font)
        End Using
        e.PaintErrorIcon(e.ClipBounds)
        e.PaintWaveLine(e.ClipBounds)
        e.PaintBorder(e.ClipBounds)
        e.Handled = True
    End If
End Sub
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;

private void gcMultiRow1_CellPainting(object sender, GrapeCity.Win.MultiRow.CellPaintingEventArgs e)
{
    GcMultiRow gcMultiRow = sender as GcMultiRow;
    if (gcMultiRow.Rows[e.RowIndex].Cells[e.CellIndex] is DateTimePickerCell)
    {
        string placeHolder = "(None)";
        using (Brush brush = new SolidBrush(brushColor))
        {
            SizeF contentSize = e.Graphics.MeasureString(placeHolder, e.CellStyle.Font);
            float y = e.CellBounds.Y + ((e.CellBounds.Height - contentSize.ToSize().Height) / 2);
        }
        e.PaintErrorIcon(e.ClipBounds);
        e.PaintWaveLine(e.ClipBounds);
        e.PaintBorder(e.ClipBounds);
        e.Handled = true;
    }
}
```

**Hide Drop-Down Button (DateTimePickerCell)**

You can hide the drop-down button in the date time picker cell by setting the `ShowDropDownButton` property to `NotShown` and the `ShowUpDown` property to `True`. In this case the spin button is shown instead of the drop-down button while editing a cell, so you may wish to adjust the position of the spin button in code in order to hide it.

**Using Code**

The following code creates a user-defined cell, `DateCell`, that inherits the date time picker cell.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

Public Class DateCell
    Inherits DateTimePickerCell

    Public Sub New()
        Me.ShowDropDownButton = CellButtonVisibility.NotShown
        Me.ShowUpDown = True
    End Sub

    Protected Overrides Function GetEditingControlBounds(ByVal cellBounds As Rectangle, ByVal rowIndex As Integer) As Rectangle
        Return New Rectangle(cellBounds.Location, New Size(cellBounds.Width + SystemInformation.VerticalScrollBarWidth, cellBounds.Height))
    End Function

End Class
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;

public class DateCell : DateTimePickerCell
{
    public DateCell()
    {
        this.ShowDropDownButton = CellButtonVisibility.NotShown;
        this.ShowUpDown = true;
    }

    protected override System.Drawing.Rectangle GetEditingControlBounds(System.Drawing.Rectangle cellBounds, int rowIndex)
    {
    }
}
```
using System.Drawing;
using System.Windows.Forms;
using GrapeCity.Win.MultiRow;

public class DateCell : DateTimePickerCell
{
    public DateCell()
    {
        this.ShowDropDownButton = CellButtonVisibility.NotShown;
        this.ShowUpDown = true;
    }

    protected override Rectangle GetEditingControlBounds(Rectangle cellBounds, int rowIndex)
    {
        return new Rectangle(cellBounds.Location, new Size(cellBounds.Width + SystemInformation.VerticalScrollBarWidth, cellBounds.Height));
    }
}

Using the Designer

Use the following steps to use the date cell in the designer.

1. Save the above code in a file such as DateCell.vb or DateCell.cs.
2. Add the file to the project.
3. Build the project.
4. Open the template designer.
5. Select "DateCell" from the toolbox and place it on the designer by dragging it.

Using Code

This example creates a date cell.

[VB]
Imports GrapeCity.Win.MultiRow
Dim dateCell1 As New DateCell()
GcMultiRow1.Template = Template.CreateGridTemplate(New Cell() { dateCell1 })
GcMultiRow1.RowCount = 10

[CS]
using GrapeCity.Win.MultiRow;
DateCell dateCell1 = new DateCell();
gcMultiRow1.Template = Template.CreateGridTemplate(new Cell[] { dateCell1 });
gcMultiRow1.RowCount = 10;

Get Formatted Values (DateTimePickerCell)

The date time picker cell does not support the DateTimePickerCell.Style.Format property, so the DateTimePickerCell.FormattedValue ('FormattedValue Property' in the on-line documentation) property always returns the string value of the DateTimePickerCell.Value ('Value Property' in the on-line documentation) property regardless of the values of the DateTimePickerCell.Format ('Format Property' in the on-line documentation) property or the DateTimePickerCell.CustomFormat ('CustomFormat Property' in the on-line documentation) property (for example: 9998/12/31 00:00:00).

You can use the Cell.DisplayText ('DisplayText Property' in the on-line documentation) property or the GcMultiRow.GetDisplayText ('GetDisplayText Method' in the on-line documentation) method to get the formatted values of the date time picker cell.

Using Code

This example gets the cell values.

[VB]
Imports GrapeCity.Win.MultiRow
Dim dateTimePickerCell1 As New DateTimePickerCell()
datetimePickerCell1.Format = DateTimePickerFormat.Short
datetimePickerCell1.Value = DateTime.Now
Dim template1 As Template = Template.CreateGridTemplate(New Cell() { dateTimePickerCell1 })
GcMultiRow1.Template = template1
GcMultiRow1.CurrentCellPosition = New CellPosition(0, 0)

[CS]
using GrapeCity.Win.MultiRow;
DateTimePickerCell dateTimePickerCell1 = new DateTimePickerCell();
datetimePickerCell1.Format = DateTimePickerFormat.Short;
datetimePickerCell1.Value = DateTime.Now;
Template template1 = Template.CreateGridTemplate(new Cell[] { dateTimePickerCell1 });
gcMultiRow1.Template = template1;
gcMultiRow1.CurrentCellPosition = new CellPosition(0, 0);
Automatically Confirm Initial Value as Input Value (DateTimePickerCell)

By default the current date and time are displayed as the initial values when the date time picker cell is being edited. This initial value is not saved as the cell value until it is explicitly entered as a cell value. This behavior is due to the `DateTimePickerEditingControl` (`DateTimePickerEditingControl Class` in the on-line documentation), which inherits from the `System.Windows.Forms.DateTimePicker` control. To apply the initial value shown in the date time picker cell as the input value, you will need to write code to confirm input in the date time picker cell, which holds the initial value, once its editing completes.

**Using Code**

This example applies the initial value.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow
Private Sub GcMultiRow1_CellEndEdit(ByVal sender As System.Object, ByVal e As CellEndEditEventArgs) Handles GcMultiRow1.CellEndEdit
    Dim gcMultiRow As GcMultiRow = TryCast(sender, GcMultiRow)
    Dim currentCell As Cell = gcMultiRow.Rows(e.RowIndex).Cells(e.CellIndex)
    If e.EditCanceled = False Then
        If TypeOf currentCell Is DateTimePickerCell Then
            If currentCell.Value Is Nothing Then
                currentCell.Value = Date.Now
            End If
        End If
    End If
End Sub
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;
private void gcMultiRow1_CellEndEdit(object sender, CellEndEditEventArgs e)
{
    GcMultiRow gcMultiRow = sender as GcMultiRow;
    Cell currentCell = gcMultiRow.Rows[e.RowIndex].Cells[e.CellIndex];
    if (e.EditCanceled == false)
    {
        if (currentCell is DateTimePickerCell)
        {
            if (currentCell.Value == null)
            {
                currentCell.Value = DateTime.Now;
            }
        }
    }
}
```

**Usage of ValueChanged Event (DateTimePickerCell)**

You can use the `DateTimePickerEditingControl.ValueChanged` event to perform the same processing as the `DateTimePicker.ValueChanged` event in the date time picker cell.

**Using Code**

This example uses the `ValueChanged` event.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    GcMultiRow1.Template = Template.CreateGridTemplate(New Cell() {New DateTimePickerCell()})
End Sub

Private Sub GcMultiRow1_EditingControlShowing(ByVal sender As System.Object, ByVal e As EditingControlShowingEventArgs) Handles GcMultiRow1.EditingControlShowing
    If TypeOf e.Control Is DateTimePickerEditingControl Then
        Dim dateTimePickerEditingControl1 As DateTimePickerEditingControl = TryCast(e.Control, DateTimePickerEditingControl)
        RemoveHandler dateTimePickerEditingControl1.ValueChanged, AddressOf Me.dateTimePickerEditingControl1_ValueChanged
        AddHandler dateTimePickerEditingControl1.ValueChanged, AddressOf Me.dateTimePickerEditingControl1_ValueChanged
    End If
End Sub

Private Sub dateTimePickerEditingControl1_ValueChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Dim dateTimePickerEditingControl1 As DateTimePickerEditingControl = TryCast(sender, DateTimePickerEditingControl)
    Dim gcMultiRow As GcMultiRow = dateTimePickerEditingControl1.GcMultiRow
    Console.WriteLine(dateTimePickerEditingControl1.Value.ToString())
End Sub
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;
private void GcMultiRow1_EditingControlShowing(object sender, EditingControlShowingEventArgs e)
{
    if (e.Control is DateTimePickerEditingControl)
    {
        DateTimePickerEditingControl dateTimePickerEditingControl1 = (DateTimePickerEditingControl)e.Control;
        RemoveHandler dateTimePickerEditingControl1.ValueChanged, AddressOf dateTimePickerEditingControl1_ValueChanged;
        AddHandler dateTimePickerEditingControl1.ValueChanged, AddressOf dateTimePickerEditingControl1_ValueChanged;
    }
}

private void dateTimePickerEditingControl1_ValueChanged(object sender, EventArgs e)
{
    DateTimePickerEditingControl dateTimePickerEditingControl1 = (DateTimePickerEditingControl)sender;
    GcMultiRow gcMultiRow = dateTimePickerEditingControl1.GcMultiRow;
    Console.WriteLine(dateTimePickerEditingControl1.Value.ToString());
}
```
using GrapeCity.Win.MultiRow;

private void Form1_Load(object sender, EventArgs e)
{
    gcMultiRow1.Template = Template.CreateGridTemplate(new Cell[] { new DateTimePickerCell() });
geMultiRow1.EditingControlShowing += new EventHandler<EditingControlShowingEventArgs>(gcMultiRow1_EditingControlShowing);
}

private void gcMultiRow1_EditingControlShowing(object sender, EditingControlShowingEventArgs e)
{
    if (e.Control is DateTimePickerEditingControl)
    {
        DateTimePickerEditingControl dateTimePickerEditingControl1 = e.Control as DateTimePickerEditingControl;
dateTimePickerEditingControl1.ValueChanged -= new EventHandler(dateTimePickerEditingControl1_ValueChanged);
dateTimePickerEditingControl1.ValueChanged += new EventHandler(dateTimePickerEditingControl1_ValueChanged);
    }
}

private void dateTimePickerEditingControl1_ValueChanged(object sender, EventArgs e)
{
    DateTimePickerEditingControl dateTimePickerEditingControl1 = sender as DateTimePickerEditingControl;
    GcMultiRow gcMultiRow = dateTimePickerEditingControl1.GcMultiRow;
    Console.WriteLine(dateTimePickerEditingControl1.Value.ToString());
}

## NumericUpDownCell

The numeric up down cell provides features similar to that of the .NET Framework `NumericUpDown` control (`System.Windows.Forms.NumericUpDown`). Users can only enter numeric values in the numeric up down cell. The appearance of the numeric up down cell spin button depends on visual style settings.

### Features

The numeric up down cell provides the following features in addition to the **Basic Cell** features.

- Numeric value input
- Increasing or decreasing values using the spin button
- Numeric separator

Refer to the **NumericUpDownCell ('NumericUpDownCell Class' in the on-line documentation)** class for details on each of these features.

### Data Types

`NumericUpDownCell` stores values of `Object` type and this type can be verified using the `NumericUpDownCell.ValueType ('ValueType Property' in the on-line documentation)` property. In `NumericUpDownCell`, values cast into a `Decimal` type can be used. This type can be verified using the `NumericUpDownCell.FormattedValueType ('FormattedValueType Property' in the on-line documentation)` property. To modify the behavior when values are read in a cell, you can override the `NumericUpDownCell.OnCellFormatting ('OnCellFormatting Method' in the on-line documentation)` method. The behavior when the value is written back from cell can be modified by overriding the `NumericUpDownCell.OnCellParsing ('OnCellParsing Method' in the on-line documentation)` method.

### Cell Editing Control

The numeric up down cell values can be edited using the `NumericUpDownEditingControl`. This control inherits the `IEditingControl ('IEditingControl Interface' in the on-line documentation)` interface and the `System.Windows.Forms.NumericUpDown` class. The cell editing control type can be verified using the `NumericUpDownCell.EditType ('EditType Property' in the on-line documentation)` property.

### Styles

The numeric up down cell supports the following members of the **CellStyle ('CellStyle Class' in the on-line documentation)** class. You can set cell styles using the `NumericUpDownCell.Style ('Style Property' in the on-line documentation)` property.
<table>
<thead>
<tr>
<th><strong>CellStyle Member</strong></th>
<th><strong>Non-editable state</strong></th>
<th><strong>Editable state</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>Border</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>DataSourceNullValue</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledBackColor</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>DisabledForeColor</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>Font</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>ForeColor</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>Format</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FormatProvider</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GradientColors</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>GradientDirection</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>GradientStyle</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>Image</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>ImageAlign</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>ImeMode</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>ImeSentenceMode</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>InputScope</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>LineAdjustment</td>
<td>Enabled only in GDI + Compatibility Mode</td>
<td>-</td>
</tr>
<tr>
<td>Margin</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>Multiline</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>NullValue</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>Padding</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>PatternColor</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>PatternStyle</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>SelectionBackColor</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>SelectionForeColor</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>Tag</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>TextAdjustment</td>
<td>Enabled only in GDI + Compatibility Mode</td>
<td>-</td>
</tr>
<tr>
<td>TextAlign</td>
<td>Enabled</td>
<td>Only for vertical direction</td>
</tr>
<tr>
<td>TextAngle</td>
<td>Enabled only in GDI + Compatibility Mode</td>
<td>-</td>
</tr>
<tr>
<td>TextEffect</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>TextImageRelation</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>TextIndent</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>TextVertical</td>
<td>Enabled only in GDI + Compatibility Mode</td>
<td>-</td>
</tr>
<tr>
<td>UseCompatibleTextRendering</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>WordWrap</td>
<td>Enabled</td>
<td>-</td>
</tr>
</tbody>
</table>
GDI + Compatibility Mode can be enabled by setting the NumericUpDownCell.Style.UseCompatibleTextRendering property to True.

Shortcut Keys

The table below lists all the keys supported while editing the numeric up down cell and the keys processed by the GcMultiRow control.

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Key</th>
<th>DateTimePickerCell</th>
<th>GcMultiRow</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Keys.PageUp</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.PageDown</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.End</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Keys.Home</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Keys.Left</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Keys.Right</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Keys.Up</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Keys.Down</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Keys.Insert</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Keys.Delete</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Keys.BackSpace</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td>Keys.Control</td>
<td>Keys.PageUp</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.PageDown</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.End</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.Home</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.Left</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.Right</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.Up</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.Down</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.A</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.C</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.V</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.X</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>Keys.Shift</td>
<td>Keys.Left</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Right</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Up</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Down</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Home</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.End</td>
<td>Enabled</td>
<td>-</td>
</tr>
</tbody>
</table>

Keys.Control + Keys.C is processed by the GcMultiRow control if multiple cells have been selected.
Events

The GcMultiRow.CellContentClick ('CellContentClick Event' in the on-line documentation) event can be used to implement processing for a mouse click in the cell content area. Use the GcMultiRow.CellContentDoubleClick ('CellContentDoubleClick Event' in the on-line documentation) event for a double-click. The GcMultiRow.CellEditedFormattedValueChanged ('CellEditedFormattedValueChanged Event' in the on-line documentation) event can be used to implement processing when the cell value is changed. The NumericUpDownEditingControl class events can be used to handle processing for events when the numeric up down cell is being edited.

Comparison with Standard Control

The following table shows a comparison of major properties between the numeric up down cell and the System.Windows.Forms.NumericUpDown control.

<table>
<thead>
<tr>
<th>NumericUpDownCell</th>
<th>NumericUpDown</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerations</td>
<td>Accelerations</td>
</tr>
<tr>
<td>Style.BackColor</td>
<td>BackColor</td>
</tr>
<tr>
<td>DecimalPlaces</td>
<td>DecimalPlaces</td>
</tr>
<tr>
<td>Style.ForeColor</td>
<td>ForeColor</td>
</tr>
<tr>
<td>Hexadecimal</td>
<td>Hexadecimal</td>
</tr>
<tr>
<td>Increment</td>
<td>Increment</td>
</tr>
<tr>
<td>InterceptArrowKeys</td>
<td>InterceptArrowKeys</td>
</tr>
<tr>
<td>Maximum</td>
<td>Maximum</td>
</tr>
<tr>
<td>Minimum</td>
<td>Minimum</td>
</tr>
<tr>
<td>None</td>
<td>Text</td>
</tr>
<tr>
<td>ThousandsSeparator</td>
<td>ThousandsSeparator</td>
</tr>
<tr>
<td>Value</td>
<td>Value</td>
</tr>
<tr>
<td>UpDownAlign</td>
<td>UpDownAlign</td>
</tr>
</tbody>
</table>

For more information, see the following topics:

- **Hiding Spin Button (NumericUpDownCell)**
- **Displaying Numeric Value Separator (NumericUpDownCell)**
- **Displaying Negative Values in Red (NumericUpDownCell)**

Displaying Negative Values in Red (NumericUpDownCell)

You can display negative values in red using conditional cell styles.

Using the Designer

1. Select the numeric up down cell to display negative values in (for example: numericUpDownCell1).
2. Select the **Style** property from the **Properties** window and then select **Combined Style** from the drop-down.
3. Click the arrow to the left of the **Style** property to expand the members of CombinedClass and then click the **Items** property collection button [...].

4. Add the cell style in the **CombinedCellStyle** editor and set the following cell style settings.
   - Set the **ForeColor** property to Black
   - Set the **TextAlign** property to **TopRight**

5. Add the conditional cell style and use the following settings for the **Items** property of the **ConditionalCellStyle** property.
   a. Click the [...] button of the **Items** property collection.
   b. Add **Is less than** from the **Conditional Formatting Rules Editor**.
   c. Select **Number** from the condition cell value.
   d. With the **CellStyle** selected in **CellStyle** type, set the **ForeColor** property to Red.

6. Click the **OK** button and verify the changes.
7. Switch the document window tab to **Runtime**.
8. Check the difference in display when values -1,0, and 1 are entered (verify the results by moving the focus to another other cell because string color does not change while the cell is selected).

**Using Code**

This example displays negative cell values when using conditional cell styles.

```vbnet
Imports GrapeCity.Win.MultiRow
```
Dim numericUpDownCell1 As New NumericUpDownCell()
numericUpDownCell1.Name = "numericUpDownCell1"
numericUpDownCell1.Value = 0

Dim baseStyle As NewCellStyle()
baseStyle.ForeColor = Color.Black
baseStyle.TextAlign = MultiRowContentAlignment.TopRight

Dim negativeValueColor As NewCellStyle()
negativeValueColor.ForeColor = Color.Red
Dim conditionalCellStyle1 As New ConditionalCellStyle()
conditionalCellStyle1.Items.Add(negativeValueColor, ConditionalCellStyleOperator.LessThan, 0)

Dim combinedCellStyle1 As New CombinedCellStyle()
combinedCellStyle1.Items.Add(baseStyle)
combinedCellStyle1.Items.Add(conditionalCellStyle1)

numericUpDownCell1.Style = combinedCellStyle1
GcMultiRow1.Template = Template.CreateGridTemplate(New Cell() {numericUpDownCell1})

Hiding Spin Button (NumericUpDownCell)

You can hide the spin button in the numeric up down cell by setting the NumericUpDownCell.ShowSpinButton ('ShowSpinButton Property' in the on-line documentation) property to NotShown; however, this setting is only enabled when the cell is not in an editable state. In order to hide the spin button in an editable state, you need to adjust the position of the spin button so that it is not visible when the numeric up down cell displays the cell editing control.

Using Code

The following code creates a user-defined numeric cell that inherits from the NumericUpDownCell ('NumericUpDownCell Class' in the on-line documentation) class.

[VB]
Imports System.Drawing
Imports System.Windows.Forms
Imports GrapeCity.Win.MultiRow

' NumericUpDownCell without the spin button.
Public Class NumericUpDownCell

[CS]
using GrapeCity.Win.MultiRow;

NumericUpDownCell numericUpDownCell1 = new NumericUpDownCell();
numericUpDownCell1.Name = "numericUpDownCell1";
numericUpDownCell1.Value = 0;

CellStyle baseStyle = new CellStyle();
baseStyle.ForeColor = Color.Black;
baseStyle.TextAlign = MultiRowContentAlignment.TopRight;

CellStyle negativeValueColor = new CellStyle();
negativeValueColor.ForeColor = Color.Red;
ConditionalCellStyle conditionalCellStyle1 = new ConditionalCellStyle();
conditionalCellStyle1.Items.Add(negativeValueColor, ConditionalCellStyleOperator.LessThan, 0);

CombinedCellStyle combinedCellStyle1 = new CombinedCellStyle();
combinedCellStyle1.Items.Add(baseStyle);
combinedCellStyle1.Items.Add(conditionalCellStyle1);

numericUpDownCell1.Style = combinedCellStyle1;
gcMultiRow1.Template = Template.CreateGridTemplate(new Cell[] { numericUpDownCell1 });
Inherits NumericUpDownCell

Public Sub New()
    Me.ShowSpinButton = CellButtonVisibility.NotShown
End Sub

Protected Overrides Function GetEditingControlBounds(ByVal cellBounds As System.Drawing.Rectangle, ByVal rowIndex As Integer) As System.Drawing.Rectangle
    Return New Rectangle(cellBounds.Location, New Size(cellBounds.Width + SystemInformation.VerticalScrollBarWidth, cellBounds.Height))
End Function
End Class

[C#]

using System.Drawing;
using System.Windows.Forms;
using GrapeCity.Win.MultiRow;
// NumericUpDownCell without the spin button.
public class NumericCell : NumericUpDownCell
{
    public NumericCell()
    {
        this.ShowSpinButton = CellButtonVisibility.NotShown;
    }
    protected override Rectangle GetEditingControlBounds(Rectangle cellBounds, int rowIndex)
    {
        return new Rectangle(cellBounds.Location, new Size(cellBounds.Width + SystemInformation.VerticalScrollBarWidth, cellBounds.Height));
    }
}

Using the Designer

Use the following steps to use the custom NumericCell in the designer.
1. Save the above code in a file such as NumericCell.vb or NumericCell.cs.
2. Add the file to a project.
3. Build the project.
4. Open the template designer.
5. Select “NumericCell” from the toolbar and place it on the designer by dragging it.

Using Code

This example uses the custom cell.

[VB]
Imports GrapeCity.Win.MultiRow
Dim numericCell1 As New NumericCell()
GcMultiRow1.Template = Template.CreateGridTemplate(New Cell() { numericCell1 })
GcMultiRow1.RowCount = 10

[CS]

using GrapeCity.Win.MultiRow;
NumericCell numericCell1 = new NumericCell();
gcMultiRow1.Template = Template.CreateGridTemplate(new Cell[] { numericCell1 });
gcMultiRow1.RowCount = 10;

Displaying Numeric Value Separator (NumericUpDownCell)

The method to display the numeric value separator for the numeric up down cell is similar to that of the System.Windows.Forms.NumericUpDown control. By setting the NumericUpDownCell.ThousandsSeparator (‘ThousandsSeparator Property’ in the on-line documentation) property to True, numeric values get displayed with the appropriate numeric separator depending upon the culture that has been set in the operating system.

100,000

The NumericUpDownCell.Style.Format property cannot be used.

Designer Settings

Use the following steps to set the separator.
1. Add the numeric up down cell to a row (for example: numericUpDownCell1).
2. Select numericUpDownCell1 and set the NumericUpDownCell.ThousandsSeparator property to True in the Properties window.
Using Code

This example creates a numeric up down cell and sets the ThousandsSeparator (‘ThousandsSeparator Property’ in the on-line documentation) property.

[VB]

Imports GrapeCity.Win.MultiRow

Dim numericUpDownCell1 As New NumericUpDownCell()
numericUpDownCell1.Name = "numericUpDownCell1"
numericUpDownCell1.ThousandsSeparator = True
numericUpDownCell1.Value = 100000

GcMultiRow1.Template = Template.CreateGridTemplate(New Cell() { numericUpDownCell1 })
GcMultiRow1.RowCount = 10

[CS]

using GrapeCity.Win.MultiRow;

NumericUpDownCell numericUpDownCell1 = new NumericUpDownCell();
numericUpDownCell1.Name = "numericUpDownCell1";
numericUpDownCell1.ThousandsSeparator = true;
numericUpDownCell1.Value = 100000;

gcMultiRow1.Template = Template.CreateGridTemplate(new Cell[] { numericUpDownCell1 });
gcMultiRow1.RowCount = 10;

DomainUpDownCell

The domain up down cell offers features similar to the .NET Framework's DomainUpDown control. The user can use this cell type to provide string selections using a spin button. You can use the combo box cell to select a string from a list just like you would in a domain up down cell; however, when you frequently need to select values around consecutive values, the domain up down cell is faster than a combo box cell. For instance, it can be used for rating notations such as "AAA", "AA", "A", "BBB" or for a set of steps such as “1. Order”, “2. Inventory Check”, “3. Credit Check”, “4. Dispatch”, and so on.

Features

In addition to the features of the Base Cell, the domain up down cell lets users select a value using the spin button. For more details on this feature, refer to the DomainUpDownCell class (on-line documentation).

Data Type

The domain up down cell uses an Object value type and this type can be checked with the DomainUpDownCell.ValueType (‘ValueType Property’ in the on-line documentation) property. The value cast into a String type is used for input and display. This type can be confirmed with the DomainUpDownCell.FormattedValueType (‘FormattedValueType Property’ in the on-line documentation) property. You can modify behavior during casting by overriding the Object type ToString method. To modify the behavior when values are read in a cell, you can override the DomainUpDownCell.OnCellFormatting (‘OnCellFormatting Method’ in the on-line documentation) method. The behavior when the value is written...
back from cell can be modified by overriding the DomainUpDownCell.OnCellParsing ('OnCellParsing Method' in the on-line documentation) method.

Cell Edit Control

The value of the domain up down cell can be edited with the DomainUpDownEditingControl control. This control inherits from the IEditingControl interface and the System.Windows.Forms.DomainUpDown class. The cell edit control type can be checked with the DomainUpDownCell.EditType ('EditType Property' in the on-line documentation) property.

Styles

The domain up down cell supports the following members of the CellStyle ('CellStyle Class' in the on-line documentation) class. You can set a cell style with the DomainUpDownCell.Style ('Style Property' in the on-line documentation) property.

<table>
<thead>
<tr>
<th>CellStyle Members</th>
<th>Non-Editable State</th>
<th>Editable State</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>Border</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>DataSourceNullValue</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>DisabledBackColor</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledForeColor</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>Font</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>ForeColor</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>Format</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>FormatProvider</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>GradientColors</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>GradientDirection</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>GradientStyle</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>Image</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>ImageAlign</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>ImeMode</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>ImeSentenceMode</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>InputScope</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>LineAdjustment</td>
<td>Enabled only for GDI+ Compatible Mode</td>
<td>-</td>
</tr>
<tr>
<td>Margin</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>Multiline</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>NullValue</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>Padding</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>PatternColor</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>PatternStyle</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>SelectionBackColor</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>SelectionForeColor</td>
<td>Enabled</td>
<td>-</td>
</tr>
</tbody>
</table>
Tag | Enabled | Enabled
---|--------|--------
TextAdjustment | Enabled only for GDI+ Compatible Mode | -
TextAlign | Enabled | Enabled
TextAngle | Enabled only for GDI+ Compatible Mode | -
TextEffect | Enabled | -
TextImageRelation | Enabled | -
TextIndent | Enabled | -
TextVertical | Enabled only for GDI+ Compatible Mode | -
UseCompatibleTextRendering | Enabled | -
WordWrap | Enabled | -

To set **GDI+ Compatibility Mode** as Enabled, set the UseCompatibleTextRendering property to True.

**Shortcut Keys**

The table below shows a list of keys processed when the domain up down cell is in an edit state and keys that are processed by the GcMultiRow control.

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Key</th>
<th>DomainUpDownCell</th>
<th>GcMultiRow</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Keys.PageUp</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.PageDown</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.End</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Home</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Left</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Right</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Up</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Down</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Insert</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Delete</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.BackSpace</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keys.Control</td>
<td>Keys.PageUp</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.PageDown</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.End</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.Home</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.Left</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Right</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Up</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.Down</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.A</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.C</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.V</td>
<td>Enabled</td>
<td>-</td>
</tr>
</tbody>
</table>
Keys/X | Enabled | -
--- | --- | ---
Keys/Z | Enabled | -
Keys/Shift | Keys/Left | Enabled | -
| Keys/Right | Enabled | -
| Keys/Up | Enabled | -
| Keys/Down | Enabled | -
| Keys/Home | Enabled | -
| Keys/End | Enabled | -

- When multiple cells are selected, Keys/Control + Keys/C is processed by the GcMultiRow control.
- When the cursor is before the first character of the domain up down cell, Keys/Left is processed by GcMultiRow.
- When the cursor is after the last character of the domain up down cell, Keys/Right is processed by the GcMultiRow control.
- When the DomainUpDownEditingControl.SelectedIndex property is 0 or -1 and the DomainUpDownEditingControl.Wrap property is set to False, Keys/Up is processed by the GcMultiRow control.
- When the DomainUpDownEditingControl.SelectedIndex property is 0 or -1 and the DomainUpDownEditingControl.Wrap property is set to False, Keys/Down is processed by the GcMultiRow control.

**Events**

You can use the GcMultiRow.CellContentClick ("CellContentClick Event' in the on-line documentation) event to catch a click in the cell content area. You can use the GcMultiRow.CellContentDoubleClick ("CellContentDoubleClick Event' in the on-line documentation) event for a double-click. To implement the processing for any changes in the cell value, you can use the GcMultiRow.CellEditedFormattedValueChanged ("CellEditedFormattedValueChanged Event' in the on-line documentation) event. The events of the DomainUpDownEditingControl class are used to implement any processing in the edit state of the domain up down cell. Events similar to the System.Windows.Forms DOMAINUpDown class are used in the DomainUpDownEditingControl class.

**LabelCell**

The label cell has features similar to the .NET Framework's Label control (System.Windows.Forms.Label). The user can use this label cell to display non-editable text.

![Label](image)

**Features**

In addition to the Base Cell features, the following features can also be used in the label cell.

- Display a Caption
- Display an Image

For more details on each feature, refer to the LabelCell ('LabelCell Class' in the on-line documentation) class.

**Data Types**
The label cell uses the Object value type and this type can be checked with the LabelCell.ValueType (ValueTypeName in the on-line documentation) property. The caption shown on the label uses a value that has been type cast to a String type. This type can be checked with the LabelCell.FormattedValueType (FormattedValueType in the on-line documentation) property. You can modify type cast behavior by overriding the Object type ToString method. Override the LabelCell.OnCellFormatting (OnCellFormatting Method in the on-line documentation) method to change behavior when a value is read in a cell. Override the LabelCell.OnCellParsing (OnCellParsing Method in the on-line documentation) method to change behavior when a value is written back from the cell.

Cell Edit Control

The label cell does not provide a cell edit control. The LabelCell.EditType (EditType Property in the on-line documentation) property always returns a null reference (Nothing in Visual Basic).

Styles

The label cell supports the following CellStyle (CellStyle Class in the on-line documentation) class members. You can set the cell style with the LabelCell.Style (Style Property in the on-line documentation) property.

<table>
<thead>
<tr>
<th>CellStyle Members</th>
<th>Enabled or Disabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>Border</td>
<td>Enabled</td>
</tr>
<tr>
<td>DataSourceNullValue</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledBackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledForeColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>Font</td>
<td>Enabled</td>
</tr>
<tr>
<td>ForeColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>Format</td>
<td>Enabled</td>
</tr>
<tr>
<td>FormatProvider</td>
<td>Enabled</td>
</tr>
<tr>
<td>GradientColors</td>
<td>Enabled</td>
</tr>
<tr>
<td>GradientDirection</td>
<td>Enabled</td>
</tr>
<tr>
<td>GradientStyle</td>
<td>Enabled</td>
</tr>
<tr>
<td>Image</td>
<td>Enabled</td>
</tr>
<tr>
<td>ImageAlign</td>
<td>Enabled</td>
</tr>
<tr>
<td>ImeMode</td>
<td>-</td>
</tr>
<tr>
<td>ImeSentenceMode</td>
<td>-</td>
</tr>
<tr>
<td>InputScope</td>
<td>-</td>
</tr>
<tr>
<td>LineAdjustment</td>
<td>Enabled only for GDI+Compatible Mode</td>
</tr>
<tr>
<td>Margin</td>
<td>Enabled</td>
</tr>
<tr>
<td>Multiline</td>
<td>Enabled</td>
</tr>
<tr>
<td>NullValue</td>
<td>Enabled</td>
</tr>
<tr>
<td>Padding</td>
<td>Enabled</td>
</tr>
<tr>
<td>PatternColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>PatternStyle</td>
<td>Enabled</td>
</tr>
<tr>
<td>SelectionBackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>Property</td>
<td>Status</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>SelectionForeColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>Tag</td>
<td>Enabled</td>
</tr>
<tr>
<td>TextAdjustment</td>
<td>Enabled only for GDI+Compatible Mode</td>
</tr>
<tr>
<td>TextAlign</td>
<td>Enabled</td>
</tr>
<tr>
<td>TextAngle</td>
<td>Enabled only for GDI+Compatible Mode</td>
</tr>
<tr>
<td>TextEffect</td>
<td>Enabled</td>
</tr>
<tr>
<td>TextImageRelation</td>
<td>Enabled</td>
</tr>
<tr>
<td>TextIndent</td>
<td>Enabled</td>
</tr>
<tr>
<td>TextVertical</td>
<td>Enabled only for GDI+Compatible Mode</td>
</tr>
<tr>
<td>UseCompatibleTextRendering</td>
<td>Enabled</td>
</tr>
<tr>
<td>WordWrap</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

To enable **GDI+ Compatibility Mode**, set the `UseCompatibleTextRendering` property to True.

**Shortcut Keys**

No shortcut keys are handled in a label cell.

**Events**

You can use the `GcMultiRow.CellContentClick` event to get a click in the cell content area. You can use the `GcMultiRow.CellContentDoubleClick` event for a double-click.

For more information see the following topics:

- Display String (LabelCell)
- Handling User Click (LabelCell)

**Handling User Click (LabelCell)**

The `GcMultiRow.CellContentClick` event is used to handle a user click on the label cell.

**Using Code**

This example uses the `CellContentClick` event.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

Private Sub GcMultiRow1_CellContentClick(ByVal sender As System.Object, ByVal e As GrapeCity.Win.MultiRow.CellEventArgs) Handles GcMultiRow1.CellContentClick
    Dim gcMultiRow As GcMultiRow = TryCast(sender, GcMultiRow)
    Dim currentCell As Cell = gcMultiRow.Rows(e.RowIndex).Cells(e.CellIndex)
    If TypeOf currentCell Is LabelCell Then
        If e.CellName = "Cell Name" Then
            If currentCell.Enabled = True Then
```
Display String (LabelCell)

You can display a string in the label cell by setting a string type value with the LabelCell.Value property (Cell.Value ('Value Property' in the on-line documentation) property).

Using the Designer

1. Add a label cell type in the row (for example: labelCell1).
2. Select labelCell1 and set any string in the labelCell1.Value property using the Properties window.

Using Code

This example creates a label cell and uses the Value property.

[VB]

Imports GrapeCity.Win.MultiRow

Dim labelCell1 As New LabelCell()
labelCell1.Name = "labelCell1"
labelCell1.Value = "Test"

GcMultiRow1.Template = Template.CreateGridTemplate(New Cell() { labelCell1 })
GcMultiRow1.RowCount = 10

[CS]

using GrapeCity.Win.MultiRow;
LabelCell labelCell1 = new LabelCell();
labelCell1.Name = "labelCell1";
labelCell1.Value = "Test";

gcMultiRow1.Template = Template.CreateGridTemplate(new Cell[] { labelCell1 });
gcMultiRow1.RowCount = 10;

LinkLabelCell

The link label cell has features similar to the .NET Framework's LinkLabel control (System.Windows.Forms.LinkLabel). The user can use the link label cell to click a hyperlink and the developer can customize the behavior when the hyperlink is clicked.

Features

In addition to the **Base Cell** features, the following features can also be used in a link label cell.

- Display String with Link
- Distinguish visited links

For more details on each feature, refer to the [LinkLabelCell](http://www.grapecity.com) class.

Data Types

The link label cell uses the value Object type and this type can be checked with the LinkLabelCell.ValueType property. The link string displayed on the link label uses a value cast into a String type. This type can be checked with the LinkLabelCell.FormattedValueType property. You can modify type cast behavior by overriding the Object type ToString method. Override the LinkLabelCell.OnCellFormatting method to change behavior when a value is read in a cell. Override the LinkLabelCell.OnCellParsing method to change behavior when a value is written back from a cell.

The link label cell does not provide a cell edit control. The LinkLabelCell.EditType property always returns a null reference (Nothing in Visual Basic).

Styles

The link label cell supports the following CellStyle class members. You can set the cell style with the LinkLabelCell.Style property.

<table>
<thead>
<tr>
<th>CellStyle Members</th>
<th>Enabled or Disabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>Border</td>
<td>Enabled</td>
</tr>
<tr>
<td>DataSourceNullValue</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledBackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledForeColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>Font</td>
<td>Enabled</td>
</tr>
<tr>
<td>Property</td>
<td>Status</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>ForeColor</td>
<td>-</td>
</tr>
<tr>
<td>Format</td>
<td>Enabled</td>
</tr>
<tr>
<td>FormatProvider</td>
<td>Enabled</td>
</tr>
<tr>
<td>GradientColors</td>
<td>Enabled</td>
</tr>
<tr>
<td>GradientDirection</td>
<td>Enabled</td>
</tr>
<tr>
<td>GradientStyle</td>
<td>Enabled</td>
</tr>
<tr>
<td>Image</td>
<td>Enabled</td>
</tr>
<tr>
<td>ImageAlign</td>
<td>Enabled</td>
</tr>
<tr>
<td>ImeMode</td>
<td>-</td>
</tr>
<tr>
<td>ImeSentenceMode</td>
<td>-</td>
</tr>
<tr>
<td>InputScope</td>
<td>-</td>
</tr>
<tr>
<td>LineAdjustment</td>
<td>Enabled only for GDI+ Compatible Mode</td>
</tr>
<tr>
<td>Margin</td>
<td>Enabled</td>
</tr>
<tr>
<td>Multiline</td>
<td>Enabled</td>
</tr>
<tr>
<td>NullValue</td>
<td>Enabled</td>
</tr>
<tr>
<td>Padding</td>
<td>Enabled</td>
</tr>
<tr>
<td>PatternColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>PatternStyle</td>
<td>Enabled</td>
</tr>
<tr>
<td>SelectionBackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>SelectionForeColor</td>
<td>-</td>
</tr>
<tr>
<td>Tag</td>
<td>Enabled</td>
</tr>
<tr>
<td>TextAdjustment</td>
<td>Enabled only for GDI+ Compatible Mode</td>
</tr>
<tr>
<td>TextAlign</td>
<td>Enabled</td>
</tr>
<tr>
<td>TextAngle</td>
<td>Enabled only for GDI+ Compatible Mode</td>
</tr>
<tr>
<td>TextEffect</td>
<td>Enabled</td>
</tr>
<tr>
<td>TextImageRelation</td>
<td>Enabled</td>
</tr>
<tr>
<td>TextIndent</td>
<td>Enabled</td>
</tr>
<tr>
<td>TextVertical</td>
<td>Enabled only for GDI+ Compatible Mode</td>
</tr>
<tr>
<td>UseCompatibleTextRendering</td>
<td>Enabled</td>
</tr>
<tr>
<td>WordWrap</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

To enable **GDI+ Compatibility Mode**, set the `UseCompatibleTextRendering` property to `True`.

**Shortcut Keys**

In the link label cell, `Keys.Space` is allocated for clicking on the link. Keys other than this are processed by the `GcMultiRow` control.

**Events**

You can use the `GcMultiRow.CellContentClick` event to get a click in the cell content area. You can use the `GcMultiRow.CellContentDoubleClick` event for a double-click.
See the following topics for more information:

- **Open Link in Associated Browser (LinkLabelCell)**

Open Link in Associated Browser (LinkLabelCell)

In order to open the URL in an associated browser once the user clicks on the link label cell, you can use the GcMultiRow.CellContentClick ('CellContentClick Event' in the on-line documentation) event.

**Using Code**

This example uses the CellContentClick ('CellContentClick Event' in the on-line documentation) event.

[VB]

```vbnet
Imports System.Diagnostics
Imports GrapeCity.Win.MultiRow

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    Dim linkLabelCell1 As New LinkLabelCell()
    linkLabelCell1.Name = "linkLabelCell1"
    linkLabelCell1.Value = "http://www.grapecity.com"
    GcMultiRow1.Template = Template.CreateGridTemplate(New Cell() {linkLabelCell1})
End Sub

Private Sub GcMultiRow1_CellContentClick(ByVal sender As System.Object, ByVal e As GrapeCity.Win.MultiRow.CellEventArgs) Handles GcMultiRow1.CellContentClick
    Dim gcMultiRow As GcMultiRow = TryCast(sender, GcMultiRow)
    Dim currentCell As Cell = gcMultiRow.Rows(e.RowIndex).Cells(e.CellIndex)
    If TypeOf currentCell Is LabelCell Then
        If e.CellName = "linkLabelCell1" Then
            If currentCell.Enabled = True Then
                Dim command As String = currentCell.Value.ToString()
                Process.Start(command)
            End If
        End If
    End If
End Sub
```

[CS]

```csharp
using System.Diagnostics;
using GrapeCity.Win.MultiRow;

private void Form1_Load(object sender, EventArgs e)
{
    LinkLabelCell linkLabelCell1 = new LinkLabelCell();
    linkLabelCell1.Name = "linkLabelCell1";
    linkLabelCell1.Value = "http://www.grapecity.com";
    gcMultiRow1.Template = Template.CreateGridTemplate(new Cell[] { linkLabelCell1 });
}

private void gcMultiRow1_CellContentClick(object sender, CellEventArgs e)
{
    GcMultiRow gcMultiRow = sender as GcMultiRow;
    Cell currentCell = gcMultiRow.Rows[e.RowIndex].Cells[e.CellIndex];
    if (currentCell is LabelCell)
    {
        if (e.CellName == "linkLabelCell1")
        {
            if (currentCell.Enabled == true)
            {
                string command = currentCell.Value.ToString();
                Process.Start(command);
            }
        }
    }
}
```

**ImageCell**

The image cell has features similar to the .NET Framework’s PictureBox control (System.Windows.Forms.PictureBox). The user can use the image cell to display an image in a cell.

**Features**

In addition to the Base Cell features, the following features can be used in the image cell.

- Display an Image
- Display an Icon
For more details on the features, refer to the ImageCell ('ImageCell Class' in the on-line documentation) class.

## Data Types

The image cell uses the Object value type and this type can be checked with the ImageCell.ValueType ('ValueType Property' in the on-line documentation) property. The ImageCell.ValueIsIcon ('ValueIsIcon Property' in the on-line documentation) property determines what datatype is used when the cell is shown. The System.Drawing.Image type value is used when the ValueIsIcon property is set to False. When the ValueIsIcon property is True, the System.Drawing.Icon type value is used. These types can be checked with the ImageCell.FormattedValueType ('FormattedValueType Property' in the on-line documentation) property. The behavior during casting can be modified by overriding the ToString method of Object type. To modify the behavior when values are read in a cell, you can override the ImageCell.OnCellFormatting ('OnCellFormatting Method' in the on-line documentation) method. The behavior when the value is written back from the cell can be modified by overriding the ImageCell.OnCellParsing ('OnCellParsing Method' in the on-line documentation) method.

## Cell Edit Control

The image cell does not provide the cell edit control. The ImageCell.EditType ('EditType Property' in the on-line documentation) property always returns a null reference (Nothing in Visual Basic).

## Styles

The image cell supports the followingCellStyle ('CellStyle Class' in the on-line documentation) class members. You can set a cell style with the ImageCell.Style property.

<table>
<thead>
<tr>
<th>CellStyle Members</th>
<th>Enabled or Disabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>Border</td>
<td>Enabled</td>
</tr>
<tr>
<td>DataSourceNullValue</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledBackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledForeColor</td>
<td>-</td>
</tr>
<tr>
<td>Font</td>
<td>-</td>
</tr>
<tr>
<td>ForeColor</td>
<td>-</td>
</tr>
<tr>
<td>Format</td>
<td>-</td>
</tr>
<tr>
<td>FormatProvider</td>
<td>-</td>
</tr>
<tr>
<td>GradientColors</td>
<td>Enabled</td>
</tr>
<tr>
<td>GradientDirection</td>
<td>Enabled</td>
</tr>
<tr>
<td>GradientStyle</td>
<td>Enabled</td>
</tr>
<tr>
<td>Image</td>
<td>Enabled</td>
</tr>
<tr>
<td>ImageAlign</td>
<td>Enabled</td>
</tr>
<tr>
<td>ImeMode</td>
<td>-</td>
</tr>
<tr>
<td>ImeSentenceMode</td>
<td>-</td>
</tr>
<tr>
<td>InputScope</td>
<td>-</td>
</tr>
<tr>
<td>LineAdjustment</td>
<td>-</td>
</tr>
<tr>
<td>Margin</td>
<td>Enabled</td>
</tr>
<tr>
<td>Multiline</td>
<td>-</td>
</tr>
<tr>
<td>NullValue</td>
<td>Enabled</td>
</tr>
</tbody>
</table>
The image set with the ImageCell.Style.Image property is drawn over the image set with ImageCell.Image.

**Shortcut Keys**

The image cell does not process shortcut keys.

**Events**

You can use the GcMultiRow.CellContentClick (CellContentClick Event' in the on-line documentation) event event to get a click in the cell content area. You can use the GcMultiRow.CellContentDoubleClick (CellContentDoubleClick Event' in the on-line documentation) event for a double-click.

For more information see the following topics:
- Display Image (ImageCell)
- Handle User Click (ImageCell)
- Display Alternate Image for Empty Value (ImageCell)

**Display Alternate Image for Empty Value (ImageCell)**

Use the ImageCell.Style.NullValue property (CellStyle.NullValue (NullValue Property' in the on-line documentation) property) if you wish to specify an alternate image when the image cell value is empty or does not have an image.

**Using the Designer**

1. Add an image cell in a row (for example: imageCell1).

**Using Code**

This example creates an alternate image for an empty value.

[VB]
Imports GrapeCity.Win.MultiRow

Dim template As Template = New Template()
Dim columnHeaderSection1 As ColumnHeaderSection = New ColumnHeaderSection()
Dim columnHeaderCell1 As ColumnHeaderCell = New ColumnHeaderCell()
Dim imageCell1 As ImageCell = New ImageCell()
imageCell1.Style.NullValue = New Bitmap("PlaceHolder.png")
template.Width = imageCell1.Size.Width
template.Row.Cells.Add(imageCell1)
template.Row.Height = imageCell1.Size.Height

columnHeaderSection1.Cells.Add(columnHeaderCell1)
columnHeaderSection1.Height = columnHeaderCell1.Size.Height
template.ColumnHeader.Add(columnHeaderSection1)

GcMultiRow1.Template = template
GcMultiRow1.RowCount = 10

Handle User Click (ImageCell)

Use the GcMultiRow.CellContentClick ('CellContentClick Event' in the on-line documentation) event to handle a user click with the image cell.

Using Code

This example uses the CellContentClick ('CellContentClick Event' in the on-line documentation) event to determine if the user clicks on the image cell.

[VB]

Imports GrapeCity.Win.MultiRow

Private Sub GcMultiRow1_CellContentClick(ByVal sender As System.Object, _
Display Image (ImageCell)

To set an image in the image cell, you need to set the image data type with the Cell.Value ('Value Property' in the on-line documentation) property.

Using Code

This example puts an image in the image cell.

[VB]
GcMultiRow1.Rows(0).Cells(0).Value = New Bitmap("test.bmp")

[CS]
gcMultiRow1.Rows[0].Cells[0].Value = new Bitmap(@"test.bmp");

To use the icon data type, set the ImageCell.ValueIsIcon ('ValueIsIcon Property' in the on-line documentation) property to True.

ProgressBarCell

The progress bar cell offers features similar to the .NET Framework's ProgressBar control.
You can use the progress bar cell to visually show a percentage in the cell.

### Features

In addition to the Base Cell features, the following features can also be used in the progress bar cell:

- Fill color based on the percentage value
- Display text value

For more details on each feature, refer to the **ProgressBarCell** ([ProgressBarCell Class' in the on-line documentation]) class.

### Data Types

The progress bar cell uses the Object type value and this type can be checked with the **ProgressBarCell. ValueType** ([Value type Property' in the on-line documentation]) property. The value cast into an Integer type is used for input and display. This type can be checked with the **ProgressBarCell.FormattedValueType** ([FormattedValue Type Property' in the on-line documentation]) property. You can override the **ProgressBarCell.OnCellFormatting** ([OnCellFormatting Method' in the on-line documentation]) method to modify the behavior when values are read in a cell. Override the **ProgressBarCell.OnCellParsing** ([OnCellParsing Method' in the on-line documentation]) method to modify behavior when the value is written back from the cell.

### Cell Edit Control

The progress bar cell does not provide a cell edit control. The **ProgressBarCell.EditType** ([Edit Type Property' in the on-line documentation]) property always returns a null reference (Nothing in Visual Basic).

### Styles

The progress bar cell supports the following **CellStyle** ([CellStyle Class' in the on-line documentation]) class members. You can set the cell style using the **ProgressBarCell.Style** ([Style Property' in the on-line documentation]) property.

#### CellStyle Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Enabled or Disabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>Enabled only when <strong>ProgressBarCell.UseVisualProgressBarStyle</strong> ([UseVisualProgressBarStyle Property' in the on-line documentation]) is False</td>
</tr>
<tr>
<td>Border</td>
<td>Enabled</td>
</tr>
<tr>
<td>DataSourceNullValue</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledBackColor</td>
<td>Enabled only when <strong>ProgressBarCell.UseVisualProgressBarStyle</strong> is False</td>
</tr>
<tr>
<td>DisabledForeColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledGradientEffect</td>
<td>Enabled only when <strong>ProgressBarCell.UseVisualProgressBarStyle</strong> is False</td>
</tr>
<tr>
<td>EditingBackColor</td>
<td>-</td>
</tr>
<tr>
<td>EditingForeColor</td>
<td>-</td>
</tr>
<tr>
<td>Font</td>
<td>Enabled</td>
</tr>
<tr>
<td>ForeColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>Format</td>
<td>Enabled</td>
</tr>
<tr>
<td>FormatProvider</td>
<td>Enabled</td>
</tr>
</tbody>
</table>
Image
ImageAlign
ImeMode
ImeSentenceMode
InputScope
LineAdjustment
Margin
MouseOverBackColor
MouseOverForeColor
MouseOverGradientEffect
Multiline
NullValue
Padding
PatternColor
PatternStyle
SelectionBackColor
SelectionForeColor
SelectionGradientEffect
Tag
TextAdjustment
TextAlign
TextAngle
TextEffect
TextImageRelation
TextIndent
TextVertical
UseCompatibleTextRendering
WordWrap

To enable **GDI+ Compatibility Mode**, set the ProgressBarCell.Style.UseCompatibleTextRendering to True.

**Shortcut Keys**

The progress bar cell does not process shortcut keys.

**Events**

You can use the GcMultiRow.CellContentClick (**CellContentClick Event' in the on-line documentation**) event to implement processing when clicking in the cell content area. Use the GcMultiRow.CellContentDoubleClick (**CellContentDoubleClick Event' in the on-line documentation**) event for a double-click.

**Comparison with Standard Control**

The following table shows a comparison between major properties of the progress bar cell and the

### ProgressBarCell

<table>
<thead>
<tr>
<th>Feature</th>
<th>ProgressBarCell</th>
<th>ProgressBar</th>
</tr>
</thead>
<tbody>
<tr>
<td>MarqueeAnimationSpeed</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Maximum</td>
<td>Maximum</td>
<td>Maximum</td>
</tr>
<tr>
<td>Minimum</td>
<td>Minimum</td>
<td>RightToLeft</td>
</tr>
<tr>
<td>None</td>
<td>RightToLeftLayout</td>
<td>Step</td>
</tr>
<tr>
<td>Step</td>
<td>Style</td>
<td>Value</td>
</tr>
<tr>
<td>ProgressBarStyle</td>
<td>Value</td>
<td>None</td>
</tr>
<tr>
<td>Value</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>FillColor</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>FillDirection</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>ShowText</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>TextFormat</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

### RichTextBoxCell

The rich text box cell has features similar to the .NET Framework's RichTextBox control (System.Windows.Forms.RichTextBox). The user can use the rich text box cell to enter formatted text into the cell.

#### Features

In addition to the Base Cell features, the rich text box cell can display and let users input formatted text. For more details on this feature, refer to the RichTextBoxCell ('RichTextBoxCell Class' in the on-line documentation) class.

#### Data Types

The rich text box cell uses the Object type value and this type can be checked with the RichTextBoxCell.ValueType ('ValueType Property' in the on-line documentation) property. The value cast into a String type is used for input and display. This type can be checked with the RichTextBoxCell FormattedValueType ('FormattedValueType Property' in the on-line documentation) property. Override the RichTextBoxCell.OnCellFormatting ('OnCellFormatting Method' in the on-line documentation) method to modify the behavior when values are read in a cell. Override the RichTextBoxCell.OnCellParsing ('OnCellParsing Method' in the on-line documentation) method to modify behavior when the value is written back from the cell.

#### Cell Edit Control

The rich text box cell value can be edited with the RichTextBoxEditingControl control. This control inherits from the IEditingControl interface and the System.Windows.Forms.RichTextBox class. The cell edit control type can be checked with the RichTextBoxCell.EditType ('EditType Property' in the on-line documentation) property.

#### Styles

The rich text box cell supports the following CellStyle ('CellStyle Class' in the on-line documentation) class members. You can set the cell style with the RichTextBoxCell.Style ('Style Property' in the on-line documentation) property.
<table>
<thead>
<tr>
<th><strong>CellStyle Members</strong></th>
<th><strong>Non-Editable State</strong></th>
<th><strong>Editable State</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>BackgroundGradientEffect</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>Border</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>DataSourceNullValue</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledBackColor</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>DisabledForeColor</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DisabledGradientEffect</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>EditingBackColor</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td>EditingForeColor</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Font</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ForeColor</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Format</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>FormatProvider</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>Image</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ImageAlign</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ImeMode</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>ImeSentenceMode</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>InputScope</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>LineAdjustment</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>Margin</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>MouseOverBackColor</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>MouseOverForeColor</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MouseOverGradientEffect</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>Multiline</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>NullValue</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>Padding</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>PatternColor</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>PatternStyle</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>SelectionBackColor</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>SelectionForeColor</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SelectionGradientEffect</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>Tag</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>TextAdjustment</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TextAlign</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TextAngle</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TextEffect</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
In the rich text box cell, the default value of the CellStyle.MultiLine property is MultiRowTriState.True.

To enable **GDI+ Compatibility Mode**, set the RichTextBoxCell.Style.UseCompatibleTextRendering property to True.

### Shortcut Keys

The following table lists keys processed when the rich text box cell is in edit mode and lists keys that are processed by the GcMultiRow control.

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Key</th>
<th>RichTextBoxCell</th>
<th>GcMultiRow</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Keys.PageUp</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.PageDown</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.End</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Home</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Left</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Right</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Up</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Down</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Insert</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Delete</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.BackSpace</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>Keys.Control</td>
<td>Keys.PageUp</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.PageDown</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.End</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Home</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Left</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Right</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Up</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Down</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.A</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.C</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.V</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.X</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>Keys.Shift</td>
<td>Keys.Left</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Right</td>
<td>Enabled</td>
<td>-</td>
</tr>
</tbody>
</table>
Keys.Up Enabled -
Keys.Down Enabled -
Keys.Home Enabled -
Keys.End Enabled -

- Keys.Control + Keys.C is processed by the GcMultiRow control when multiple cells are selected.
- Keys.PageDown and Keys.PageUp are processed by the GcMultiRow control when the value of the rich text cell has not been modified.
- Keys.Left is processed by the GcMultiRow control when the cursor is before the first character of the rich text box cell.
- Keys.Right is processed by the GcMultiRow control when the cursor is after the last character of the rich text box cell.
- Keys.Up is processed by the GcMultiRow control when the cursor of the rich text box cell is in the first row.
- Keys.Down is processed by the GcMultiRow control when the cursor of the rich text box cell is in the last row.
- Keys.Up and Keys.Down are processed by the GcMultiRow control when the rich text box cell does not allow multiple rows.

Events

You can use the GcMultiRow.CellContentClick (‘CellContentClick Event’ in the on-line documentation) event to implement processing when clicking in the cell content area. Use the GcMultiRow.CellContentDoubleClick (‘CellContentDoubleClick Event’ in the on-line documentation) event for a double-click. Use the GcMultiRow.CellEditedFormattedValueChanged (‘CellEditedFormattedValueChanged Event’ in the on-line documentation) event to catch any changes to the cell value. The RichTextBoxEditingControl class event is used to process the RichTextBoxCell editing event.

RadioGroupCell

The radio group cell has features similar to the .NET Framework’s RadioButton control (System.Windows.Forms.RadioButton). The user can select a value with the radio button in the radio group cell.

Features

In addition to the Base Cell features, the radio group cell lets users select a value from multiple items. For more details on this feature, refer to the RadioGroupCell (‘RadioGroupCell Class’ in the on-line documentation) class.

Data Types

The radio group cell uses the Object type value. This type can be checked with the RadioGroupCell.ValueType (‘ValueType Property’ in the on-line documentation) property. The value cast into the Integer type (int type in C#) is used for input and display. This type can be checked with the RadioGroupCell.FormattedValueType (‘FormattedValueType Property’ in the on-line documentation) property. You can override the RadioGroupCell.OnCellFormatting (‘OnCellFormatting Method’ in the on-line documentation) method to modify the behavior when values are read in a cell. Override the RadioGroupCell.OnCellParsing (‘OnCellParsing Method’ in the on-line documentation) method to modify behavior when the value is written back from the cell.

Cell Edit Control

The radio group cell does not support the cell edit control. The RadioGroupCell.EditType (‘EditType Property’ in
the on-line documentation) property always returns a null reference (Nothing in Visual Basic).

**Styles**

The radio group cell supports the following `CellStyle` (`CellStyle Class' in the on-line documentation) members. You can set the cell style using the `RadioGroupCell.Style` (`Style Property' in the on-line documentation) property.

**CellStyle Members**

<table>
<thead>
<tr>
<th>CellStyle Members</th>
<th>Enabled or Disabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>BackgroundGradientEffect</td>
<td>Enabled</td>
</tr>
<tr>
<td>Border</td>
<td>Enabled</td>
</tr>
<tr>
<td>DataSourceNullValue</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledBackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledForeColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledGradientEffect</td>
<td>Enabled</td>
</tr>
<tr>
<td>EditingBackColor</td>
<td>-</td>
</tr>
<tr>
<td>EditingForeColor</td>
<td>-</td>
</tr>
<tr>
<td>Font</td>
<td>Enabled</td>
</tr>
<tr>
<td>ForeColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>Format</td>
<td>Enabled</td>
</tr>
<tr>
<td>FormatProvider</td>
<td>Enabled</td>
</tr>
<tr>
<td>Image</td>
<td>-</td>
</tr>
<tr>
<td>ImageAlign</td>
<td>-</td>
</tr>
<tr>
<td>ImeMode</td>
<td>-</td>
</tr>
<tr>
<td>ImeSentenceMode</td>
<td>-</td>
</tr>
<tr>
<td>InputScope</td>
<td>-</td>
</tr>
<tr>
<td>LineAdjustment</td>
<td>Enabled only for GDI+ Compatible Mode</td>
</tr>
<tr>
<td>Margin</td>
<td>Enabled</td>
</tr>
<tr>
<td>MouseOverBackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>MouseOverForeColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>MouseOverGradientEffect</td>
<td>Enabled</td>
</tr>
<tr>
<td>multiline</td>
<td>Enabled</td>
</tr>
<tr>
<td>NullValue</td>
<td>Enabled</td>
</tr>
<tr>
<td>Padding</td>
<td>Enabled</td>
</tr>
<tr>
<td>PatternColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>PatternStyle</td>
<td>Enabled</td>
</tr>
<tr>
<td>SelectionBackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>SelectionForeColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>SelectionGradientEffect</td>
<td>Enabled</td>
</tr>
</tbody>
</table>
Tag: Enabled
TextAdjustment: Enabled only for GDI+ Compatible Mode
TextAlign: Enabled
TextAngle: Enabled only for GDI+ Compatible Mode
TextEffect: Enabled
TextImageRelation: -
TextIndent: Enabled
TextVertical: Enabled only for GDI+ Compatible Mode
UseCompatibleTextRendering: Enabled
WordWrap: Enabled

To set **GDI+ Compatibility Mode** as Enabled, set the RadioGroupCell.Style.UseCompatibleTextRendering property to True.

**Shortcut Keys**

The following table lists the keys that are processed when the radio group cell is in edit mode and lists keys that are processed by the GcMultiRow control.

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Key</th>
<th>RadioGroupCell</th>
<th>GcMultiRow</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Keys.PageUp</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.PageDown</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.End</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.Home</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.Left</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Right</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Up</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Down</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Insert</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Delete</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.BackSpace</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>Keys.Control</td>
<td>Keys.PageUp</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.PageDown</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.End</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.Home</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.Left</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.Right</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.Up</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.Down</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.A</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.C</td>
<td>Enabled</td>
<td>-</td>
</tr>
</tbody>
</table>
MultiRow Windows Forms Developer's Guide

Keys.Shift

None

Keys.V

-

Enabled

Keys.X

-

Enabled

Keys.Left

-

Enabled

Keys.Right

-

Enabled

Keys.Up

-

Enabled

Keys.Down

-

Enabled

Keys.Home

-

Enabled

Keys.End

-

Enabled

Keys.Space

Enabled

-

272

Keys.Control + Keys.C is processed by the GcMultiRow control when multiple cells are selected.

Events
You can use the GcMultiRow.CellContentClick ('CellContentClick Event' in the on-line documentation) event
to implement the processing for a click in the cell content area. Use the GcMultiRow.CellContentDoubleClick
('CellContentDoubleClick Event' in the on-line documentation) event for a double-click. The
GcMultiRow.CellEditedFormattedValueChanged ('CellEditedFormattedValueChanged Event' in the online documentation) event is fired when the cell value changes.
See the following topic for more information:
Detect Value Changed (RadioGroupCell)
Detect Value Changed (RadioGroupCell)
You can use the GcMultiRow.CellEditedFormattedValueChanged ('CellEditedFormattedValueChanged Event' in the on-line documentation) event to detect any changes in the checked status of the check box cell. Use the
Cell.EditedFormattedValue ('EditedFormattedValue Property' in the on-line documentation) property to get the edited value.

Using Code
This code creates a test template.

[VB]
Imports GrapeCity.Win.MultiRow
Dim radioGroupCell1 As New RadioGroupCell()
radioGroupCell1.Items.AddRange(New String() {"ItemA", "ItemB"})
radioGroupCell1.Size = radioGroupCell1.PreferredSize
GcMultiRow1.Template = Template.CreateGridTemplate(New Cell() {radioGroupCell1})

[CS]
using GrapeCity.Win.MultiRow;
RadioGroupCell radioGroupCell1 = new RadioGroupCell();
radioGroupCell1.Items.AddRange(new string[] {"ItemA", "ItemB"});
radioGroupCell1.Size = radioGroupCell1.PreferredSize;
gcMultiRow1.Template = Template.CreateGridTemplate(new Cell[] { radioGroupCell1 });
This code shows the current value when the user changes the checkbox status.

[VB]
Imports GrapeCity.Win.MultiRow
Private Sub GcMultiRow1_CellEditedFormattedValueChanged(ByVal sender As System.Object, ByVal e As CellEditedFormattedValueChangedEventArgs) Handles GcMultiRow1.CellEditedFormattedValueChanged
Dim gcMultiRow As GcMultiRow = TryCast(sender, GcMultiRow)
Dim currentCell As Cell = gcMultiRow.Rows(e.RowIndex).Cells(e.CellIndex)
If e.Scope = CellScope.Row Then
If TypeOf currentCell Is RadioGroupCell Then
Console.WriteLine("Cell value being edited: {0}", currentCell.EditedFormattedValue)
Console.WriteLine("Cell Value: {0}", currentCell.Value)
End If
End If
End Sub

[CS]
using GrapeCity.Win.MultiRow;
private void gcMultiRow1_CellEditedFormattedValueChanged(object sender, CellEditedFormattedValueChangedEventArgs e)
{
GcMultiRow gcMultiRow = sender as GcMultiRow;
Cell currentCell = gcMultiRow.Rows[e.RowIndex].Cells[e.CellIndex];
if (e.Scope == CellScope.Row)
{
if (currentCell is RadioGroupCell)
{
Console.WriteLine("Cell value being edited: {0}", currentCell.EditedFormattedValue);
Console.WriteLine("Cell Value: {0}", currentCell.Value);

Copyright © GrapeCity, inc. All rights reserved.


ListBoxCell

The ListBoxCell has features similar to the .NET Framework's ListBox control (System.Windows.Forms.ListBox). The user can use this ListBoxCell to display a list of items in the cell.

Features

In addition to the Base Cell features, the ListBoxCell provides a list display.

For more details on list box cells, refer to the ListBoxCell (ListBoxCell Class' in the on-line documentation) class.

Data Type

The ListBoxCell retains the Object value type. This type can be checked using the ListBoxCell.ValueType (ValueType Property' in the on-line documentation) property.

For display, a value that has been type cast to a String type is used. This type can be checked with the ListBoxCell.FormattedValueType (FormattedValueType Property' in the on-line documentation) property.

You can modify type cast behavior by overriding the Object type of the ToString (ToString Method' in the on-line documentation) method. Override the ListBoxCell.OnCellFormatting (OnCellFormatting Method' in the on-line documentation) method to change behavior when a value is read in a cell. Override the ListBoxCell.OnCellParsing (OnCellParsing Method' in the on-line documentation) method to change behavior when a value is written back from the cell.

Cell Editing Control

The ListBoxCell does not provide a cell edit control. The ListBoxCell.EditType (EditText Property' in the on-line documentation) property always returns a null reference (Nothing in case of Visual Basic).

Style

The ListBoxCell supports the followingCellStyle (CellStyle Class' in the on-line documentation) class members. You can set the cell style with the ListBoxCell.Style property.

<table>
<thead>
<tr>
<th>CellStyle Members</th>
<th>Enabled or Disabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>BackgroundGradientEffect</td>
<td>Enabled</td>
</tr>
<tr>
<td>Border</td>
<td>Enabled</td>
</tr>
<tr>
<td>DataSourceNullValue</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledBackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledForeColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledGradientEffect</td>
<td>Enabled</td>
</tr>
<tr>
<td>EditingBackColor</td>
<td>Enabled</td>
</tr>
</tbody>
</table>
To enable **GDI+ Compatible Mode**, set the UseCompatibleTextRendering property to True.

### Shortcut Keys

The table below lists all the keys supported while editing the ListBoxCell and the GcMultiRow control.

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Key</th>
<th>ListBoxCell</th>
<th>GcMultiRow</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>None</th>
<th>Keys.End</th>
<th>Enabled</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keys.Home</td>
<td>Enabled</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Keys.Left</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Keys.Right</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Keys.Up</td>
<td>Enabled</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Keys.Down</td>
<td>Enabled</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Keys.Insert</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Keys.Delete</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Keys.BackSpace</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Keys.Space</td>
<td>Enabled</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Keys.PageUp</td>
<td>Enabled</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Keys.PageDown</td>
<td>Enabled</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Events**

The GcMultiRow.CellEditedFormattedValueChanged (‘CellEditedFormattedValueChanged Event’ in the on-line documentation) event can be used to implement processing when the cell value is changed.

**ListLabelCell**

The list label cell provides the functionality to display labels as a list of items in a bulleted format. You can use the list label cell to implement a bulleted display by simply adding items.

**Features**

In addition to the Base Cell features, the list label cell displays a bulleted list.

For more details on list label cells, refer to the ListLabelCell (‘ListLabelCell Class’ in the on-line documentation) class.

**Data Types**

The list label cell retains the Object value type. This type can be checked using the ListLabelCell.ValueType (‘Value Type Property’ in the on-line documentation) property.

For display, a value that has been type cast to a String type is used. This type can be checked with the ListLabelCell.FormattedValueType (‘Formatted Value Type Property’ in the on-line documentation) property.

You can modify type cast behavior by overriding the Object type of the ToString (‘ToString Method’ in the on-line documentation) method. Override the ListLabelCell.OnCellFormatting (‘OnCellFormatting Method’ in the on-line documentation) method to change behavior when a value is read in a cell. Override the ListLabelCell.OnCellParsing (‘OnCellParsing Method’ in the on-line documentation) method to change behavior when a value is written back from the cell.

**Cell Edit Control**
The list label cell does not provide a cell edit control. The ListLabelCell.EditType (‘EditType Property’ in the on-line documentation) property always returns a null reference (Nothing in Visual Basic).

**Style**

The list label cell supports the following CellStyle (‘CellStyle Class’ in the on-line documentation) class members. You can set the cell style using the ListLabelCell.Style property.

### CellStyle Members

<table>
<thead>
<tr>
<th>Member</th>
<th>Enabled or Disabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>BackgroundGradientEffect</td>
<td>Enabled</td>
</tr>
<tr>
<td>Border</td>
<td>Enabled</td>
</tr>
<tr>
<td>DataSourceNullValue</td>
<td>-</td>
</tr>
<tr>
<td>DisabledBackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledForeColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledGradientEffect</td>
<td>Enabled</td>
</tr>
<tr>
<td>EditingBackColor</td>
<td>-</td>
</tr>
<tr>
<td>EditingForeColor</td>
<td>-</td>
</tr>
<tr>
<td>Font</td>
<td>Enabled</td>
</tr>
<tr>
<td>ForeColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>Format</td>
<td>-</td>
</tr>
<tr>
<td>FormatProvider</td>
<td>-</td>
</tr>
<tr>
<td>Image</td>
<td>-</td>
</tr>
<tr>
<td>ImageAlign</td>
<td>-</td>
</tr>
<tr>
<td>ImeMode</td>
<td>-</td>
</tr>
<tr>
<td>ImeSentenceMode</td>
<td>-</td>
</tr>
<tr>
<td>InputScope</td>
<td>-</td>
</tr>
<tr>
<td>LineAdjustment</td>
<td>-</td>
</tr>
<tr>
<td>Margin</td>
<td>Enabled</td>
</tr>
<tr>
<td>MouseOverBackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>MouseOverForeColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>MouseOverGradientEffect</td>
<td>Enabled</td>
</tr>
<tr>
<td>Multiline</td>
<td>Enabled</td>
</tr>
<tr>
<td>NullValue</td>
<td>-</td>
</tr>
<tr>
<td>Padding</td>
<td>Enabled</td>
</tr>
<tr>
<td>PatternColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>PatternStyle</td>
<td>Enabled</td>
</tr>
<tr>
<td>SelectionBackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>SelectionForeColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>SelectionGradientEffect</td>
<td>Enabled</td>
</tr>
</tbody>
</table>
Tag: Enabled
TextAdjustment: -
TextAlign: -
TextAngle: -
TextEffect: Enabled
TextImageRelation: -
TextIndent: -
TextVertical: -
UseCompatibleTextRendering: Enabled
WordWrap: Enabled

To enable **GDI+ Compatibility Mode**, set the UseCompatibleTextRendering property to True.

**Shortcut Keys**

No shortcut keys are handled in a list label cell.

**Events**

You can use the GcMultiRow.CellContentClick (‘CellContentClick Event’ in the on-line documentation) event to catch a click in the cell content area. You can use the GcMultiRow.CellContentDoubleClick (‘CellContentDoubleClick Event’ in the on-line documentation) event for a double-click.

For more information, see the Setting Bullets (ListLabelCell).

**Setting Bullets (ListLabelCell)**

You can use the ListLabelCell.BulletStyle (‘BulletStyle Property’ in the on-line documentation) property to set bullets in the list items of a list label cell. This section describes how to set bullets.

**Setting Bullets Using the Designer**

1. Add a list label cell in the row (for example: listLabelCell1).
2. Select listLabelCell1, and in the Properties window, set the bullets with the BulletStyle property.

**Using Code**

This example sets the bullets.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow
Dim listLabelCell1 As New ListLabelCell()
listLabelCell1.Name = "listLabelCell1"
listLabelCell1.Items.AddRange(New String() {"aaa", "bbb", "ccc"})
listLabelCell1.BulletStyle = BulletStyle.Numbered
GcMultiRow1.Template = Template.CreateGridTemplate(New Cell() {listLabelCell1})
GcMultiRow1.RowCount = 10
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;
ListLabelCell listLabelCell1 = new ListLabelCell();
listLabelCell1.Name = "listLabelCell1";
listLabelCell1.Items.AddRange(new string[] { "aaa", "bbb", "ccc" });
listLabelCell1.BulletStyle = BulletStyle.Numbered;
gcMultiRow1.Template = Template.CreateGridTemplate(new Cell() { listLabelCell1 });
gcMultiRow1.RowCount = 10;
```

**Setting Characters in Bullets**

You can set any characters in the bullets. Set the ListLabelCell.BulletStyle (‘BulletStyle Property’ in the on-line documentation) property to CustomTextBullet, create a class that implements the ICustomTextBullet (‘ICustomTextBullet Interface’ in the on-line documentation) interface, and set it in the CustomTextBullet (‘CustomTextBullet Property’ in the on-line documentation) property.

**Using Code**
This example creates a class for a custom bullet.

[VB]
Imports GrapeCity.Win.MultiRow
Public Class JapaneseBullet Implements ICustomTextBullet
    Public Function GetTextBullet(ByVal index As Integer) As String Implements GrapeCity.Win.MultiRow.ICustomTextBullet.GetTextBullet
        If index = 0 Then Return "1"
        ElseIf index = 1 Then Return "2"
        ElseIf index = 2 Then Return "3"
        Else Return "Other"
        End If
    End Function
End Class

listLabelCell1.BulletStyle = BulletStyle.CustomTextBullet
listLabelCell1.CustomTextBullet = New JapaneseBullet

[CS]
using GrapeCity.Win.MultiRow;
public class JapaneseBullet : ICustomTextBullet
{
    public string GetTextBullet(int index)
    {
        if(index == 0)
        {
            return "1";
        }
        else if(index == 1)
        {
            return "2";
        }
        else if (index == 2)
        {
            return "3";
        }
        else
        {
            return "Other";
        }
    }
}
listLabelCell1.BulletStyle = BulletStyle.CustomTextBullet;
listLabelCell1.CustomTextBullet = new JapaneseBullet();

Setting Images in Bullets
You can set an image in the bullets. Set the ListLabelCell.BulletStyle (‘BulletStyle Property’ in the on-line documentation) property to CustomImage, and set the image data type in the BulletImage (‘BulletImage Property’ in the on-line documentation) property.

Using Code
This example sets an image in the bullet.

[VB]
Imports GrapeCity.Win.MultiRow
listLabelCell1.BulletStyle = BulletStyle.CustomImage
listLabelCell1.BulletImage = Image.FromFile("test.png")

[CS]
using GrapeCity.Win.MultiRow;
listLabelCell1.BulletStyle = BulletStyle.CustomImage;
listLabelCell1.BulletImage = Image.FromFile("test.png");

ShapeCell
The shape cell provides the ability to display shapes. You can use the shape cell to display shapes such as triangle, circle, square, and so on, in the cell at design time, without writing a single line of code.
Features

In addition to the Base Cell features, the shape cell can also display shapes.

For more details on this feature, refer to the ShapeCell (ShapeCell Class' in the on-line documentation) class.

Data Types

The shape cell uses the Object value type. This type can be checked with the ShapeCell.ValueType (ValueType Property' in the on-line documentation) property.

The text shown in the list label uses a value that has been type cast to a String type. This type can be checked with the ShapeCell.FormattedValueType (FormattedValueType Property' in the on-line documentation) property.

You can modify type cast behavior by overriding the Object type ToString method. Override the ShapeCell.OnCellFormatting (OnCellFormatting Method' in the on-line documentation) method to change behavior when a value is read in a cell. Override the ShapeCell.OnCellParsing (OnCellParsing Method' in the on-line documentation) method to change behavior when a value is written back from the cell.

Cell Edit Control

The shape cell does not provide a cell edit control. The ShapeCell>EditType (EditType Property' in the on-line documentation) property always returns a null reference (Nothing in Visual Basic).

Styles

The shape cell supports the followingCellStyle (CellStyle Class' in the on-line documentation) class members.

You can set the cell style with the ShapeCell.Style (Style Property' in the on-line documentation) property.

<table>
<thead>
<tr>
<th>CellStyle Members</th>
<th>Enabled or Disabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>BackgroundGradientEffect</td>
<td>Enabled</td>
</tr>
<tr>
<td>Border</td>
<td>-</td>
</tr>
<tr>
<td>DataSourceNullValue</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledBackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledForeColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledGradientEffect</td>
<td>Enabled</td>
</tr>
<tr>
<td>EditingBackColor</td>
<td>-</td>
</tr>
<tr>
<td>EditingForeColor</td>
<td>-</td>
</tr>
<tr>
<td>Font</td>
<td>Enabled</td>
</tr>
<tr>
<td>ForeColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>Format</td>
<td>Enabled</td>
</tr>
<tr>
<td>FormatProvider</td>
<td>Enabled</td>
</tr>
<tr>
<td>Image</td>
<td>Enabled</td>
</tr>
<tr>
<td>ImageAlign</td>
<td>Enabled</td>
</tr>
<tr>
<td>ImeMode</td>
<td>-</td>
</tr>
<tr>
<td>ImeSentenceMode</td>
<td>-</td>
</tr>
<tr>
<td>InputScope</td>
<td>-</td>
</tr>
</tbody>
</table>
LineAdjustment  Enabled
Margin  Enabled
MouseOverBackColor  Enabled
MouseOverForeColor  Enabled
MouseOverGradientEffect  Enabled
Multiline  Enabled
NullValue  Enabled
Padding  Enabled
PatternColor  Enabled
PatternStyle  Enabled
SelectionBackColor  Enabled
SelectionForeColor  Enabled
SelectionGradientEffect  Enabled
Tag  Enabled
TextAdjustment  Enabled
TextAlign  Enabled
TextAngle  Enabled
TextEffect  Enabled
TextImageRelation  Enabled
TextIndent  Enabled
TextVertical  Enabled
UseCompatibleTextRendering  Enabled
WordWrap  Enabled

To enable **GDI+ Compatibility Mode**, set the CellStyle.UseCompatibleTextRendering ('UseCompatibleTextRendering Property' in the on-line documentation) property to True.

**Shortcut Keys**

No shortcut keys are handled in a shape cell.

**Events**

You can use the GcMultiRow.CellContentClick ('CellContentClick Event' in the on-line documentation) event to catch a click in the cell content area. You can use the GcMultiRow.CellContentDoubleClick ('CellContentDoubleClick Event' in the on-line documentation) event for a double-click.

For more information, see Setting Shapes (ShapeCell).

**Setting Shapes (ShapeCell)**

You can use the ShapeCell.Renderer ('Renderer Property' in the on-line documentation) property to set shapes in a shape cell. This section describes how to set shapes.

**Setting Shapes using the Designer**
1. Select the shape cell in which you want to set the shape (for example: shapeCell1).
2. Select the **Renderer** property in the **Properties** window, and select **Pentagon** from the drop-down list.

3. Expand the Renderer class by clicking the "+" sign at the left of the **Renderer** property.
4. Change the following properties of the Renderer class:
   - Set the **ArrowLength** property to 30.
   - Set the **Direction** property to **Top**.
   - Set the **LineWidth** property to 3.

**Using Code**

The following example creates a shape cell.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow
Dim pentagonShapeRenderer1 As New PentagonShapeRenderer()
pentagonShapeRenderer1.ArrowLength = 30
pentagonShapeRenderer1.Direction = ShapeDirection.Top
pentagonShapeRenderer1.LineWidth = 3
Dim shapeCell1 As New ShapeCell()
shapeCell1.Name = "shapeCell1"
shapeCell1.Renderer = pentagonShapeRenderer1
GcMultiRow1.Template = Template.CreateGridTemplate(New Cell() {shapeCell1})
GcMultiRow1.RowCount = 10
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;
PentagonShapeRenderer pentagonShapeRenderer1 = new PentagonShapeRenderer();
pentagonShapeRenderer1.ArrowLength = 30;
pentagonShapeRenderer1.Direction = ShapeDirection.Top;
pentagonShapeRenderer1.LineWidth = 3;
ShapeCell shapeCell1 = new ShapeCell();
shapeCell1.Name = "shapeCell1";
shapeCell1.Renderer = pentagonShapeRenderer1;
gcMultiRow1.Template = Template.CreateGridTemplate(new Cell[] { shapeCell1 });
gcMultiRow1.RowCount = 10;
```
Pop-upCell

The pop-up cell offers a series of textboxes and buttons. The cell also provides a window of your choice as the sub-editor of the cell when the user clicks on the button. For example, various input screens or tables (which cannot be provided as cell types), can be implemented as separate windows (forms). They can then be called from the pop-up cell and the result can be stored. The pop-up cell supports all windows (forms) and common dialogs.

Features

In addition to the Base Cell features, the following features can also be used in the pop-up cell.

- Display window (form) and common dialog
- Fetch return value of window (form) and common dialog
- Display value
- Change display location of button

For more details on each feature, refer to the PopupCell ('PopupCell Class' in the on-line documentation) class.

Data Types

The pop-up cell uses the Object type value and this type can be confirmed with the PopupCell.ValueType ('ValueType Property' in the on-line documentation) property.

Cell Edit Control

The pop-up cell value can be edited with the PopupEditingControl control. This control uses a TextBox and a button and implements the IEditingControl interface. The cell edit control type can be checked with the PopupCell>EditType ('EditType Property' in the on-line documentation) property.

Styles

The pop-up cell supports the followingCellStyle ('CellStyle Class' in the on-line documentation) class members. You can set the cell style with the PopupCell.Style ('Style Property' in the on-line documentation) property.

<table>
<thead>
<tr>
<th>CellStyle Members</th>
<th>Non-Editable State</th>
<th>Editable State</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>BackgroundGradientEffect</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td>Border</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>DataSourceNullValue</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledBackColor</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>DisabledForeColor</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>DisabledGradientEffect</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>EditingBackColor</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>EditingForeColor</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>Font</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>ForeColor</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>Format</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>FormatProvider</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
</tbody>
</table>
To set **GDI+ Compatibility Mode** as Enabled, set the `PopupCell.Style.UseCompatibleTextRendering` property to `True`.

### Shortcut Keys

The following table lists the keys that are processed when the pop up cell is in edit mode and also lists keys that are processed by the `GcMultiRow` control.

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Key</th>
<th>PopupCell</th>
<th><code>GcMultiRow</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td><code>Keys.End</code></td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td><code>Keys.Home</code></td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td><code>Keys.Left</code></td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>Keys</td>
<td>Control</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>---------</td>
<td>---------------</td>
<td>---------</td>
</tr>
<tr>
<td>Keys.Right</td>
<td>Enabled</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Keys.Up</td>
<td>-</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>Keys.Down</td>
<td>-</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>Keys.Insert</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Keys.Delete</td>
<td>Enabled</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Keys.BackSpace</td>
<td>Enabled</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Keys.Control</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keys.PageUp</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Keys.PageDown</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Keys.End</td>
<td>-</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>Keys.Home</td>
<td>-</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>Keys.Left</td>
<td>Enabled</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Keys.Right</td>
<td>Enabled</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Keys.Up</td>
<td>-</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>Keys.Down</td>
<td>-</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td>Keys.A</td>
<td>Enabled</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Keys.C</td>
<td>Enabled</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Keys.V</td>
<td>Enabled</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Keys.X</td>
<td>Enabled</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Keys.Shift</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keys.Left</td>
<td>Enabled</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Keys.Right</td>
<td>Enabled</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Keys.Up</td>
<td>Enabled</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Keys.Down</td>
<td>Enabled</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Keys.Home</td>
<td>Enabled</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Keys.End</td>
<td>Enabled</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

- **Keys.Control + Keys.C** is processed by the GcMultiRow control when multiple cells are selected.

**Events**

You can use the GcMultiRow.CellContentClick (**CellContentClick Event** in the on-line documentation) event to implement the processing for a click in the cell content area. You can use the GcMultiRow.CellContentDoubleClick (**CellContentDoubleClick Event** in the on-line documentation) event for a double-click. Use the GcMultiRow.CellEditedFormattedValueChanged (**CellEditedFormattedValueChanged Event** in the on-line documentation) event to catch a change in the cell value. The PopupEditingControl class event is used to process the pop up cell editing event.

**Comparison with Standard Control**

This cell type does not have a corresponding standard control.

**Select Color (Pop-upCell)**

You can select a color in the pop-up cell by using the System.Windows.Forms.ColorDialog.
Using the Designer

1. Add a pop-up cell to the row (for example: popupCell1).
2. Add a ColorDialog to the row (for example: colorDialog1).
3. Select popupCell1, then select the popupCell1.Popup property from the Properties window and click the ellipses button.
4. In the displayed Popup screen, expand the tree, and select colorDialog1.
5. Click the OK button and close the dialog.
6. Select popupCell1, then select the popupCell1.PopupValueMember property from the Properties window and select Color from the drop-down window.
7. Change the designer document window tab to Runtime.
8. Click the popupCell1 button and confirm the Set Color dialog option.
9. Select any color from the Set Color dialog and click the OK button.
10. In the input area of popupCell1, check the selected color string.

Using Code

This example provides a color dialog.

[VB]

Imports GrapeCity.Win.MultiRow

Dim popupCell1 As New PopupCell()
popupCell1.Popup = New ColorDialog()
popupCell1.PopupValueMember = "Color"

GcMultiRow1.Template = Template.CreateGridTemplate(New Cell() { popupCell1 })
GcMultiRow1.RowCount = 10

[CS]

using GrapeCity.Win.MultiRow;

PopupCell popupCell1 = new PopupCell();
popupCell1.Popup = new ColorDialog();
popupCell1.PopupValueMember = "Color";

gcMultiRow1.Template = Template.CreateGridTemplate(new Cell[] { popupCell1 });
gcMultiRow1.RowCount = 10;

SummaryCell

The summary cell displays calculations and results of cell values. You can use the summary cell to define a formula at design time without using code. Calculations are possible in unbound mode when the calculation column of the data source cannot be used.

Summary: Cost

Features

In addition to the Base Cell features, the following features can also be used in the summary cell.

- Calculation of Any Column (Cell)
- Sum
- Average
- Minimum Value
- Maximum Value
- Value Count
- Calculation Using Operators
  - + - * / ()
- Specify whether to calculate hidden row
- User defined formula

For more details on each feature, refer to the **SummaryCell** ("SummaryCell Class' in the on-line documentation) class.

**Data Types**

The summary cell uses the Object type value. This type can be checked with the SummaryCell.**Value**Type ('Value**Type Property' in the on-line documentation) property. The value cast into the String type is used for display. This type can be checked with the SummaryCell.**Formatted**Value**Type** ('Formatted**Value**Type Property' in the on-line documentation) property. You can modify behavior during casting by overriding the **ToString** Object type method. You can override the SummaryCell.**OnCell**Formatting ('OnCell**Formatting** Method' in the on-line documentation) method to modify the behavior when values are read in a cell.

**Cell Edit Control**

The summary cell does not support the cell edit control. The SummaryCell.**Edit**Type ('**Edit**Type Property' in the on-line documentation) property always returns a null reference (Nothing in Visual Basic).

**Styles**

The summary cell supports the following **CellStyle** ('**CellStyle** Class' in the on-line documentation) class members. You can set the cell style with the SummaryCell.**Style** ('**Style** Property' in the on-line documentation) property.

<table>
<thead>
<tr>
<th><strong>CellStyle Members</strong></th>
<th><strong>Enabled or Disabled</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>BackgroundGradientEffect</td>
<td>Enabled</td>
</tr>
<tr>
<td>Border</td>
<td>Enabled</td>
</tr>
<tr>
<td>DataSourceNullValue</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledBackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledForeColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledGradientEffect</td>
<td>Enabled</td>
</tr>
<tr>
<td>EditingBackColor</td>
<td>-</td>
</tr>
<tr>
<td>EditingForeColor</td>
<td>-</td>
</tr>
<tr>
<td>Font</td>
<td>Enabled</td>
</tr>
<tr>
<td>ForeColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>Format</td>
<td>Enabled</td>
</tr>
<tr>
<td>FormatProvider</td>
<td>Enabled</td>
</tr>
<tr>
<td>Image</td>
<td>Enabled</td>
</tr>
<tr>
<td>ImageAlign</td>
<td>Enabled</td>
</tr>
</tbody>
</table>
ImeMode
ImeSentenceMode
InputScope
LineAdjustment
Margin
MouseOverBackColor
MouseOverForeColor
MouseOverGradientEffect
Multiline
NullValue
Padding
PatternColor
PatternStyle
SelectionBackColor
SelectionForeColor
SelectionGradientEffect
Tag
TextAdjustment
TextAlign
TextAngle
TextEffect
TextImageRelation
TextIndent
TextVertical
UseCompatibleTextRendering
WordWrap

To set GDI+ Compatibility Mode as Enabled, set the UseCompatibleTextRendering property to True.

Shortcut Keys

The summary cell does not process any shortcut keys.

Events

You can use the GcMultiRow.CellContentClick ('CellContentClick Event' in the on-line documentation) event to implement the processing when clicking in the cell content area. Use the GcMultiRow.CellContentDoubleClick ('CellContentDoubleClick Event' in the on-line documentation) event for a double-click.

Comparison with Standard Control

This cell type does not correspond to a standard control.

See the following topics for more information:
You can use the summary cell to display a column sum in the footer. The example below shows the sum of values entered in the numeric up down cell in the summary cell of the footer.

Using the Designer

1. Add the numeric up down cell to a row (for example: numericUpDownCell1).
2. In Visual Studio, select Template - Add - Column Footer from the menu and add the column footer.
3. Add the summary cell in the column footer section (for example: summaryCell1).
4. Select summaryCell1 and from Properties window, set the summaryCell1.Style.Format property to ###,###.
5. In the Properties window, select MathStatistics from the drop-down list for sumCell1.Calculation property.
6. Click the plus sign to the right of the summaryCell1.Calculation property and expand the MathStatistics object.
7. Set the StatisticsType property of MathStatistics to Sum.
8. Set the CellName property of MathStatistics to numericUpDownCell1.
9. Change the designer document window tab to Runtime.

Enter a value in the numericUpDownCell1 of each row and the sum is displayed in the footer.

Using Code

```vbnet
Imports GrapeCity.Win.MultiRow
Dim numericUpDownCell1 As New NumericUpDownCell()
Dim summaryCell1 As New SummaryCell()
' Set display format of cell value.
summaryCell1.Style.Format = "###,###"
' Display Sum of cell column "numericUpDownCell1"
summaryCell1.Calculation = New MathStatistics(StatisticsType.Sum, "numericUpDownCell1")
Dim cells As Cell() = {numericUpDownCell1}
Dim template1 As Template = Template.CreateGridTemplate(cells)
Dim columnFooterSection1 As ColumnFooterSection = New ColumnFooterSection()
summaryCell1.Location = template1.Row.Cells("numericUpDownCell1").Location
columnFooterSection1.Cells.Add(summaryCell1)
template1.ColumnFooters.Add(columnFooterSection1)
GcMultiRow1.Template = template1
```

```csharp
MultiRow Windows Forms Developer's Guide
Show Column Sum in Footer (SummaryCell)
Create Statement of Account (SummaryCell)
Displaying an Error Icon for Data Errors (SummaryCell)
```

Copyright © GrapeCity, Inc. All rights reserved.
Using GrapeCity.Win.MultiRow;

NumericUpDownCell numericUpDownCell1 = new NumericUpDownCell();
numericUpDownCell1.Name = "numericUpDownCell1";
SummaryCell summaryCell1 = new SummaryCell();
// Set display format of cell value.
summaryCell1.Style.Format = "###,###";
// Display Sum of cell column "numericUpDownCell1"
summaryCell1.Calculation = new MathStatistics(StatisticsType.Sum, "numericUpDownCell1");

Cell[] cells = { numericUpDownCell1, numericUpDownCell2 };  
Template template1 = Template.CreateGridTemplate(cells);

ColumnFooterSection columnFooterSection = new ColumnFooterSection();
summaryCell1.Location = template1.Row.Cells["numericUpDownCell1"].Location;
columnFooterSection.Cells.Add(summaryCell1);
template1.ColumnFooters.Add(columnFooterSection);
gcMultiRow1.Template = template1;

Using the Designer

When calculating multiple cells, you can use the ExpressionString ('ExpressionString Property' in the on-line documentation) property of the MathStatistics ('MathStatistics Class' in the on-line documentation) class to set the cells you wish to calculate. Specify the formula string to be used for calculation, in the ExpressionString property.

Use the following steps to display the average of the sum total of multiple cells in the footer.

1. Add two numeric cells in the Row (for example: numericUpDownCell1, numericUpDownCell2).
2. Select the Template - Add - Column Footer menu in Visual Studio, and add a column footer (ColumnFooterSection).
3. Add a summary cell in the ColumnFooterSection (for example: summaryCell1).
6. Click the plus mark on the right of the summaryCell1.Calculation property, and expand the properties of the MathStatistics object.
7. Set the StatisticsType property of MathStatistics to Average.
8. Select the ExpressionString property of MathStatistics, and click the ... button.
9. Set the calculation formula in the Expression Builder that is displayed (for example: numericUpDownCell1 + numericUpDownCell2).
10. Click the OK button and close the Expression Builder.
11. Switch the document window tab of the designer to Runtime.

If you enter values in the numericUpDownCell1 and numericUpDownCell2 of each row, the average of the sum total of each row is displayed in the footer.

Using Code

The following code calculates multiple cells.

[VB]

Imports GrapeCity.Win.MultiRow

Dim numericUpDownCell1 As New NumericUpDownCell()
numericUpDownCell1.Name = "numericUpDownCell1"
Dim numericUpDownCell2 As New NumericUpDownCell()
Using the SummaryCell control, you can display an error icon if a number is divided by zero, or if a non-numeric value is set in the calculating formula which results in a data error in the calculated value.
Using the Designer

1. Add two text box cells to the Row (for example: textBoxCell1, textBoxCell2).
2. Set the textBoxCell1.Value property to "10", and set the textBoxCell2.Value property to "1".
3. Add a SummaryCell in the Row (for example: summaryCell1).
4. Add a SummaryCell in the column footer (ColumnFooterSection). For example: summaryCell1.
5. Select the summaryCell1, and from the Properties window, select Expression from the drop-down list of the Calculation property.
6. Click the plus sign at the right of the summaryCell1.Calculation property, then click the ... button of the ExpressionString property to open the Expression Editor.
7. Set "textBoxCell1/textBoxCell2" in the Expression Editor, then click the OK button.
8. Set the summaryCell1.ShowDataErrorIcon property to True.
9. Run the project, enter "10" in textBoxCell1, and "1" in textBoxCell2, and check that the calculation result is displayed in the summary cell.
10. Enter "0" in the textBoxCell2, and check that the error icon is displayed.
11. Enter "a" in the textBoxCell1, "1" in the textBoxCell2, and check that the error icon is displayed.

Using Code

This example shows an error icon.

[VB]

Imports GrapeCity.Win.MultiRow
Dim textBoxCell1 As New TextBoxCell()
textBoxCell1.Name = "textBoxCell1"
textBoxCell1.Value = 10
Dim textBoxCell2 As New TextBoxCell()
textBoxCell2.Name = "textBoxCell2"
textBoxCell2.Value = 0
Dim expression1 As New GrapeCity.Win.MultiRow.Expression()
expression1.ExpressionString = "textBoxCell1 / textBoxCell2"
Dim summaryCell1 As New SummaryCell()
summaryCell1.Name = "summaryCell1"
summaryCell1.Calculation = expression1
' Display the error icon.
summaryCell1.ShowDataErrorIcon = True
GcMultiRow1.Template = Template.CreateGridTemplate(New Cell() {textBoxCell1, textBoxCell2, summaryCell1})

[CS]

using GrapeCity.Win.MultiRow;
TextBoxCell textBoxCell1 = new TextBoxCell();
textBoxCell1.Name = "textBoxCell1";
textBoxCell1.Value = 10;
TextBoxCell textBoxCell2 = new TextBoxCell();
textBoxCell2.Name = "textBoxCell2";
textBoxCell2.Value = 0;
Expression expression1 = new GrapeCity.Win.MultiRow.Expression();
expression1.ExpressionString = "textBoxCell1 / textBoxCell2";
SummaryCell summaryCell1 = new SummaryCell();
summaryCell1.Name = "summaryCell1";
summaryCell1.Calculation = expression1;
// Display the error icon
summaryCell1.ShowDataErrorIcon = true;
gcMultiRow1.Template = Template.CreateGridTemplate(new Cell() { textBoxCell1, textBoxCell2, summaryCell1 });

Create Statement of Account (SummaryCell)

If you wish to use the summary cell to calculate an account balance like a bank passbook (with statements of credits and debits), you need to create your own calculation object which implements the ICalculation ("ICalculation Interface" in the on-line documentation) interface.
Using Code
The code below uses credit and debit values and returns the balance total in the previous row. If there is no previous row (in case of the first row), it lists the balance as 0.

[VB]
Imports GrapeCity.Win.MultiRow
Public Class CustomCalculation
    Implements ICalculation
    Private _balanceCellName As String
    Private _creditsCellName As String
    Private _chargesCellName As String

        If String.IsNullOrEmpty(Me._creditsCellName) Then Return 0
        If String.IsNullOrEmpty(Me._chargesCellName) Then Return 0
        If context.Scope = CellScope.Row Then
            Dim balance As Decimal = 0
            Dim credits As Decimal = 0
            Dim charges As Decimal = 0
            If Not context.SectionIndex = 0 Then
                Dim balanceCellValue As Object = context.GcMultiRow.GetValue(context.SectionIndex - 1, Me._balanceCellName)
                balance = DirectCast(balanceCellValue, Decimal)
            End If
            Dim creditCellValue As Object = context.GcMultiRow.GetValue(context.SectionIndex, Me._creditsCellName)
            If creditCellValue IsNot Nothing Then
                credits = DirectCast(creditCellValue, Decimal)
            End If
            Dim chargeCellValue As Object = context.GcMultiRow.GetValue(context.SectionIndex, Me._chargesCellName)
            If chargeCellValue IsNot Nothing Then
                charges = DirectCast(chargeCellValue, Decimal)
            End If
            Return balance + charges - credits
        End If
        Return Nothing
    End Function

    ' Specify cell for Balance
    Public Property BalanceCellName() As String
        Get
            Return _balanceCellName
        End Get
        Set(ByVal value As String)
            _balanceCellName = value
        End Set
    End Property

    ' Specify cell for Credit
    Public Property CreditsCellName() As String
        Get
            Return _creditsCellName
        End Get
        Set(ByVal value As String)
            _creditsCellName = value
        End Set
    End Property

    ' Specify cell for Charges
    Public Property ChargesCellName() As String
        Get
            Return _chargesCellName
        End Get
        Set(ByVal value As String)
            _chargesCellName = value
        End Set
    End Property

    Public Function Clone() As Object Implements System.ICloneable.Clone
        Dim _customCalculation As New CustomCalculation()
        _customCalculation.BalanceCellName = Me.BalanceCellName
        _customCalculation.CreditsCellName = Me.CreditsCellName
        _customCalculation.ChargesCellName = Me.ChargesCellName
        Return _customCalculation
    End Function
End Class

[CS]
using GrapeCity.Win.MultiRow;
public class CustomCalculation : ICalculation
{
    private string _balanceCellName;
    private string _creditsCellName;
    private string _chargesCellName;
}
public object Calculate(CalculationContext context)
{
    if (string.IsNullOrEmpty(this._creditsCellName)) return 0;
    if (string.IsNullOrEmpty(this._chargesCellName)) return 0;
    if (context.Scope == CellScope.Row)
    {
        decimal balance = 0;
        decimal credits = 0;
        decimal charges = 0;
        if (context.SectionIndex != 0)
        {
            object balanceCellValue = context.GcMultiRow.GetValue(context.SectionIndex - 1, this._balanceCellName);
            balance = (decimal)balanceCellValue;
        }
        object creditsCellValue = context.GcMultiRow.GetValue(context.SectionIndex, this._creditsCellName);
        if (creditsCellValue != null)
        {
            credits = (decimal)creditsCellValue;
        }
        object chargesCellValue = context.GcMultiRow.GetValue(context.SectionIndex, this._chargesCellName);
        if (chargesCellValue != null)
        {
            charges = (decimal)chargesCellValue;
        }
        return balance + charges - credits;
    }
    return null;
}

// Specify cell for Balance
public string BalanceCellName
{
    get { return _balanceCellName; }
    set { _balanceCellName = value; }
}

// Specify cell for Credit
public string CreditsCellName
{
    get { return _creditsCellName; }
    set { _creditsCellName = value; }
}

// Specify cell for Charges
public string ChargesCellName
{
    get { return _chargesCellName; }
    set { _chargesCellName = value; }
}

public object Clone()
{
    CustomCalculation _customCalculation = new CustomCalculation();
    _customCalculation.BalanceCellName = this.BalanceCellName;
    _customCalculation.CreditsCellName = this.CreditsCellName;
    _customCalculation.ChargesCellName = this.ChargesCellName;
    return _customCalculation;
}

This example creates the template.

[VB]
Imports GrapeCity.Win.MultiRow
Dim charges As New NumericUpDownCell()
charges.Name = "Charges"
Dim credits As New NumericUpDownCell()
credits.Name = "Credit"
Dim balance As New SummaryCell()
balance.Name = "Balance"
Dim custom As New CustomCalculation()
custom.BalanceCellName = "Balance"
custom.CreditsCellName = "Credits"
custom.ChargesCellName = "Charges"
balance.Calculation = custom
balance.Style.TextAlign = MultiRowContentAlignment.MiddleRight
Dim template1 = Template.CreateGridTemplate(New Cell() {charges, credits, balance})
template1.ColumnHeader(0).Cells(0).Value = "Charges"
template1.ColumnHeader(0).Cells(1).Value = "Credit"
template1.ColumnHeader(0).Cells(2).Value = "Balance"
GcMultiRow1.Template = template1

[CS]
using GrapeCity.Win.MultiRow;
NumericUpDownCell charges = new NumericUpDownCell();
charges.Name = "Charges";
NumericUpDownCell credits = new NumericUpDownCell();
credits.Name = "Credit";
SummaryCell balance = new SummaryCell();
balance.Name = "Balance";
The print info cell is used to put information into a template that is useful when printing, such as the page number, total number of pages, and so on. You can use this cell to output information without using any code in the print event.

{PageNumber} / {PageCount}

Features

In addition to the Base Cell features, the following features can also be used in the print info cell.

- Output current page number
- Output the total number of pages to be printed
- Output date and time

For more details on each feature, refer to the PrintInfoCell ('PrintInfoCell Class' in the on-line documentation) class.

Data Types

The print info cell uses the Object type value. This type can be checked with the PrintInfoCell.ValueType (ValueType Property in the on-line documentation) property. The value cast into the String type is used for input and display. This type can be checked with the PrintInfoCell.FormattedValueType (FormattedValueType Property in the on-line documentation) property. To modify the behavior when values are read in a cell, you can override the PrintInfoCell.OnCellFormatting ('OnCellFormatting Method' in the on-line documentation) method. Override the PrintInfoCell.OnCellParsing ('OnCellParsing Method' in the on-line documentation) method to modify behavior when the value is written back from the cell.

Cell Edit Control

The print info cell does not provide a cell edit control. The PrintInfoCell.EditType (EditType Property in the on-line documentation) property always returns a null reference (Nothing in Visual Basic).

Styles

The print info cell supports the followingCellStyle ('CellStyle Class' in the on-line documentation) class members. You can set the cell style with the PrintInfoCell.Style (Style 'Style Property' in the on-line documentation) property.

<table>
<thead>
<tr>
<th>CellStyle Members</th>
<th>Enabled or Disabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>BackgroundGradientEffect</td>
<td>Enabled</td>
</tr>
<tr>
<td>Border</td>
<td>Enabled</td>
</tr>
<tr>
<td>DataSourceNullValue</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledBackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledForeColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>Property</td>
<td>Status</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>DisabledGradientEffect</td>
<td>Enabled</td>
</tr>
<tr>
<td>EditingBackColor</td>
<td>-</td>
</tr>
<tr>
<td>EditingForeColor</td>
<td>-</td>
</tr>
<tr>
<td>Font</td>
<td>Enabled</td>
</tr>
<tr>
<td>ForeColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>Format</td>
<td>Enabled</td>
</tr>
<tr>
<td>FormatProvider</td>
<td>Enabled</td>
</tr>
<tr>
<td>Image</td>
<td>Enabled</td>
</tr>
<tr>
<td>ImageAlign</td>
<td>Enabled</td>
</tr>
<tr>
<td>ImeMode</td>
<td>-</td>
</tr>
<tr>
<td>ImeSentenceMode</td>
<td>-</td>
</tr>
<tr>
<td>InputScope</td>
<td>-</td>
</tr>
<tr>
<td>LineAdjustment</td>
<td>Enabled only for GDI+ CompatibleMode</td>
</tr>
<tr>
<td>Margin</td>
<td>Enabled</td>
</tr>
<tr>
<td>MouseOverBackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>MouseOverForeColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>MouseOverGradientEffect</td>
<td>Enabled</td>
</tr>
<tr>
<td>Multiline</td>
<td>Enabled</td>
</tr>
<tr>
<td>NullValue</td>
<td>Enabled</td>
</tr>
<tr>
<td>Padding</td>
<td>Enabled</td>
</tr>
<tr>
<td>PatternColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>PatternStyle</td>
<td>Enabled</td>
</tr>
<tr>
<td>SelectionBackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>SelectionForeColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>SelectionGradientEffect</td>
<td>Enabled</td>
</tr>
<tr>
<td>Tag</td>
<td>Enabled</td>
</tr>
<tr>
<td>TextAdjustment</td>
<td>Enabled only for GDI+ CompatibleMode</td>
</tr>
<tr>
<td>TextAlign</td>
<td>Enabled</td>
</tr>
<tr>
<td>TextAngle</td>
<td>Enabled only for GDI+ CompatibleMode</td>
</tr>
<tr>
<td>TextEffect</td>
<td>Enabled</td>
</tr>
<tr>
<td>TextImageRelation</td>
<td>Enabled</td>
</tr>
<tr>
<td>TextIndent</td>
<td>Enabled</td>
</tr>
<tr>
<td>TextVertical</td>
<td>Enabled only for GDI+ CompatibleMode</td>
</tr>
<tr>
<td>UseCompatibleTextRendering</td>
<td>Enabled</td>
</tr>
<tr>
<td>WordWrap</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

To set **GDI+ Compatibility Mode** as Enabled, set the UseCompatibleTextRendering property to True.

**Shortcut Keys**
The print info cell does not process shortcut keys.

**Events**

You can use the GcMultiRow.CellContentClick (**CellContentClick Event' in the on-line documentation**) event to implement processing when clicking in the cell content area. Use the GcMultiRow.CellContentDoubleClick (**CellContentDoubleClick Event' in the on-line documentation**) event for a double-click.

**Comparison with Standard Control**

This cell type does not correspond to a standard control.

**TrackBarCell**

The track bar cell has features similar to the .NET Framework's TrackBar (System.Windows.Forms.TrackBar). The user can use the track bar cell to visually input numeric values based on the slider and the scale.

**Features**

In addition to the **Base Cell** features, the following features can also be used in the track bar cell.

- Slider
- Scale
- Change to Horizontal or Vertical direction

For more details on each feature, refer to the **TrackBarCell (TrackBarCell Class' in the on-line documentation)** class.

**Data Types**

The track bar cell uses the Object type value. This type can be checked with the TrackBarCell.ValueType (**ValueType Property' in the on-line documentation**) property. The edited value in the track bar cell is the one cast into an Integer type. This type can be checked with the TrackBarCell.FormattedValueType (**FormattedValueType Property' in the on-line documentation**) property. Override the TrackBarCell.OnCellFormatting (**OnCellFormatting Method' in the on-line documentation**) method to modify the behavior when values are read in a cell. Override the TrackBarCell.OnCellParsing (**OnCellParsing Method' in the on-line documentation**) method to modify the behavior when the value is written back from the cell.

**Cell Edit Control**

The track bar cell does not support the cell edit control. The TrackBarCell.EditType (**EditType Property' in the on-line documentation**) property always returns a null reference (Nothing in Visual Basic).

**Styles**

The track bar cell supports the following **CellStyle (CellStyle Class' in the on-line documentation)** class members. You can set the cell style with the TrackBarCell.Style (**Style Property' in the on-line documentation**) property.

<table>
<thead>
<tr>
<th><strong>CellStyle Members</strong></th>
<th><strong>Enabled or Disabled</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>BackgroundGradientEffect</td>
<td>Enabled</td>
</tr>
<tr>
<td>Border</td>
<td>Enabled</td>
</tr>
<tr>
<td>DataSourceNullValue</td>
<td>Enabled</td>
</tr>
<tr>
<td>Property</td>
<td>Value</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>DisabledBackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>DisabledForeColor</td>
<td>-</td>
</tr>
<tr>
<td>DisabledGradientEffect</td>
<td>Enabled</td>
</tr>
<tr>
<td>EditingBackColor</td>
<td>-</td>
</tr>
<tr>
<td>EditingForeColor</td>
<td>-</td>
</tr>
<tr>
<td>Font</td>
<td>-</td>
</tr>
<tr>
<td>ForeColor</td>
<td>-</td>
</tr>
<tr>
<td>Format</td>
<td>-</td>
</tr>
<tr>
<td>FormatProvider</td>
<td>-</td>
</tr>
<tr>
<td>Image</td>
<td>-</td>
</tr>
<tr>
<td>ImageAlign</td>
<td>-</td>
</tr>
<tr>
<td>ImeMode</td>
<td>-</td>
</tr>
<tr>
<td>ImeSentenceMode</td>
<td>-</td>
</tr>
<tr>
<td>InputScope</td>
<td>-</td>
</tr>
<tr>
<td>LineAdjustment</td>
<td>-</td>
</tr>
<tr>
<td>Margin</td>
<td>Enabled</td>
</tr>
<tr>
<td>MouseOverBackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>MouseOverForeColor</td>
<td>-</td>
</tr>
<tr>
<td>MouseOverGradientEffect</td>
<td>Enabled</td>
</tr>
<tr>
<td>Multiline</td>
<td>-</td>
</tr>
<tr>
<td>NullValue</td>
<td>-</td>
</tr>
<tr>
<td>Padding</td>
<td>Enabled</td>
</tr>
<tr>
<td>PatternColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>PatternStyle</td>
<td>Enabled</td>
</tr>
<tr>
<td>SelectionBackColor</td>
<td>Enabled</td>
</tr>
<tr>
<td>SelectionForeColor</td>
<td>-</td>
</tr>
<tr>
<td>SelectionGradientEffect</td>
<td>Enabled</td>
</tr>
<tr>
<td>Tag</td>
<td>Enabled</td>
</tr>
<tr>
<td>TextAdjustment</td>
<td>-</td>
</tr>
<tr>
<td>TextAlign</td>
<td>-</td>
</tr>
<tr>
<td>TextAngle</td>
<td>-</td>
</tr>
<tr>
<td>TextEffect</td>
<td>-</td>
</tr>
<tr>
<td>TextImageRelation</td>
<td>-</td>
</tr>
<tr>
<td>TextIndent</td>
<td>-</td>
</tr>
<tr>
<td>TextVertical</td>
<td>-</td>
</tr>
<tr>
<td>UseCompatibleTextRendering</td>
<td>-</td>
</tr>
<tr>
<td>WordWrap</td>
<td>-</td>
</tr>
</tbody>
</table>
Shortcut Keys

The table below lists keys processed when the track bar cell is in edit mode and lists keys that are processed by the GcMultiRow control.

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Key</th>
<th>TrackBarCell</th>
<th>GcMultiRow</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Keys.PageUp</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.PageDown</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.End</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.Home</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.Left</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.Right</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.Up</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Down</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Insert</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Delete</td>
<td>Enabled</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.Backspace</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Keys.Control</td>
<td>Keys.PageUp</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.PageDown</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.End</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.Home</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.Left</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.Right</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.Up</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.Down</td>
<td>-</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Keys.A</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.C</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.V</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Keys.X</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Events

You can use the GcMultiRow.CellContentClick (`CellContentClick Event' in the on-line documentation) event to implement processing when clicking in the cell content area. Use the GcMultiRow.CellContentDoubleClick (`CellContentDoubleClick Event' in the on-line documentation) event for a double-click. The GcMultiRow.CellEditedFormattedValueChanged (`CellEditedFormattedValueChanged Event' in the on-line documentation) event is fired when the cell value changes.

Comparison with Standard Control

The table below shows a comparison between major properties of the track bar cell and the System.Windows.Forms.TrackBar control.

<table>
<thead>
<tr>
<th>TrackBarCell</th>
<th>TrackBar</th>
</tr>
</thead>
</table>
FilteringTextBoxCell

The filtering text box cell is an extended text box cell used for row filtering. You can place it in the column header section and use it for creating a filter row. It cannot be placed in a row.

Features

In addition to the features of the TextBoxCell, the following features can also be used.

- Specify name and index of cells for filtering
- Specify value comparison system
- Comparison of user definition

For more details on each feature, refer to the FilteringTextBoxCell ('FilteringTextBoxCell Class' in the on-line documentation) class.

For more information on data types, cell edit control, style, shortcut keys, comparison with the standard control, and events, refer to the TextBoxCell.

User-Defined Cell

The user-defined cell is a cell type created from the base cell or other combination of cell types as the base.

User-Defined Cell Category

The user-defined cell is created as an inherited class of an existing cell type class. Depending upon the required features, you can select any cell type as the base. For example, see the following scenarios:

- Inherit HeaderCell ('HeaderCell Class' in the on-line documentation) and customize header display.
- Inherit ColumnHeaderCell ('ColumnHeaderCell Class' in the on-line documentation) and customize column header display.
- Inherit HeaderCell and create any column header apart from ColumnHeaderCell.
- Inherit TextBoxCell ('TextBoxCell Class' in the on-line documentation) and make frequently used styles as built-in cell type styles.
- Inherit TextBoxCell and display any control while editing the cell (cell edit control).
- Inherit from an existing cell edit control, add any process, and implement it on another cell type.

In the case of adding a new property into the class inherited from a cell or built-in cell type, you need to override the Clone method and copy the value of the new property while duplicating. For more details, refer to Add Custom Property mentioned later.

Create a User-Defined Cell

The user-defined cell can be implemented through code regardless of the type of project (application, class library). You can define a cell and then register it in the toolbox, all in a project that is currently being debugged (the same way you use a user control in a Windows form).

The following steps explain how to create a user-defined cell and place it in the template designer.

1. Open Visual Studio and create a new Windows Form Application project (for example: WindowsApplication1).
2. From the Visual Studio menu, select Project - Add New Item.
3. Select MultiRow 7.0 Template from the list and click on the Add button (for example: Template1.vb, Template1.cs).
4. Once again select Project - Add New Item from the Visual Studio menu.
5. Select Class from the list and add the class file into the project (for example: MyTextBoxCell.vb, MyTextBoxCell.cs).
6. Open the class file and add the following code.

   [VB]
   ```vbnet
   Imports System.Drawing
   Imports System.Windows.Forms
   Imports GrapeCity.Win.MultiRow

   Public Class MyTextBoxCell
       Inherits TextBoxCell
       Public Sub New()
           Me.Style.BackColor = Color.Azure
       End Sub
   End Class
   ```

   [CS]
   ```csharp
   using System.Drawing;
   using System.Windows.Forms;
   using GrapeCity.Win.MultiRow;

   public class MyTextBoxCell : TextBoxCell
   {
       public MyTextBoxCell()
       {
       }
   }
   ```

7. Build the project.
8. Open the added template and display the Template Designer.
10. Confirm the user-defined cell has been added to the top of the ToolBox items.

Now that the user-defined cell has been created, you can drag and drop it from the ToolBox to the Template and you can use it the same way as a built-in cell type.

**Add Custom Property**

Cells added to the GcMultiRow control become clones of the cell instances (GcMultiRow.Template.Row.Cells property), at runtime. So if you have created a user-defined cell and are implementing the Custom property, you need to create a duplicate of the property value in the Clone method.

**Using Code**

The following example creates a custom property.

**[VB]**

```vbnet
Public Class MyTextBoxCell
    Inherits TextBoxCell

    Private _editingBackColor As Color

    Property EditingBackColor() As Color
        Get
            Return _editingBackColor
        End Get
        Set(ByVal value As Color)
            _editingBackColor = value
        End Set
    End Property

End Class
```

**[CS]**

```csharp
public class MyTextBoxCell : TextBoxCell
{
    private Color _editingBackColor;

    public Color EditingBackColor
    {
        get { return _editingBackColor; }
        set { _editingBackColor = value; }
    }
}
```

The code for the Clone method is as follows.

**[VB]**

```vbnet
Public Class MyTextBoxCell
    Inherits TextBoxCell

    Public Overrides Function Clone() As Object
        Dim myTextBoxCell As MyTextBoxCell = DirectCast(MyBase.Clone, MyTextBoxCell)
        myTextBoxCell._editingBackColor = Me.EditingBackColor
        Return myTextBoxCell
    End Function

End Class
```
End Function
End Class

[CS]

```csharp
public class MyTextBoxCell : TextBoxCell
{
    public override object Clone()
    {
        MyTextBoxCell myTextBoxCell = base.Clone() as MyTextBoxCell;
        myTextBoxCell._editingBackColor = this.EditingBackColor;
        return myTextBoxCell;
    }
}
```

For more information, see the following topic:

- Register User Defined Cell in ToolBox

Register User Defined Cell in ToolBox

You can define a cell and then register it in the toolbox in a project that is currently being debugged (similar to how this is done with a user control on a form). This allows you to use the toolbox in the same way as you would when handling a user control in the Windows Form Designer.

Adding User Defined Cell in Same Project

When you add a user defined cell using source code in the same project, building the project displays the user defined cell in the designer toolbox. This does not require any extra steps.

![Toolbox](image)

Adding User Defined Cell Defined in External Assembly

In order to register and use a user defined cell that is built as an assembly (.exe, .dll) into the toolbox, drag and drop this file from explorer to the Visual Studio toolbox.

Rows and Cells

This section explains how to work with rows and cells at runtime.
• Current Row
• Current Cell
• Adding Rows
• Deleting Rows
• Getting and Setting the Row Count
• Freezing Rows
• Selected Rows and Cells
• Restricting Cell Selection
• Disabled Cells
• Hiding Cells and Rows
• Editing Cells
• Displaying Error Messages
• Read-Only Cells
• Cell Note

Current Row

The row in which the user is working is called the current row. You can write code independent of the row index if you reference the rows through the current row. You can access the current row by using the GcMultiRow.CurrentRow (‘CurrentRow Property’ in the on-line documentation) property.

Retrieving the Current Row Index

You can retrieve the current row index using the GcMultiRow.CurrentCellPosition.RowIndex property.

Using Code

This example gets the current row index.

[Vb]

[Cs]
Console.WriteLine("Current row index: {0}", gcMultiRow1.CurrentCellPosition.RowIndex);

Moving the Current Row


Using Code

This example moves the current row.

[Vb]

[Cs]
gcMultiRow1.CurrentCellPosition = new GrapeCity.Win.MultiRow.CellPosition(1, 0);

Accessing Cells in the Current Row

You can access the cells in the current row using the Row.Cells (‘Cells Property’ in the on-line documentation) property.

Using Code

This example gets the current cell index.

[Vb]
Imports GrapeCity.Win.MultiRow
For Each cell As Cell In GcMultiRow1.CurrentRow.Cells

Console.WriteLine("Cell index: {0}", cell.CellIndex)
Next

[CS]

using GrapeCity.Win.MultiRow;

foreach (Cell cell in gcMultiRow1.CurrentRow.Cells)
{
    Console.WriteLine("Cell index: {0}", cell.CellIndex);
}

If the current row does not exist, the GcMultiRow.CurrentRow (CurrentRow Property' in the on-line documentation) property returns a null reference (Nothing in Visual Basic). For example, when the template is not set in the GcMultiRow control or when the GcMultiRow control is in Display mode (GcMultiRow.ViewMode=Display).

Displaying Indicators

For information about displaying the indicator in the current row, see Using Row Headers.

Border

Use the GcMultiRow.CurrentRowBorderLine (CurrentRowBorderLine Property' in the on-line documentation) property to set the border style for the current row.

Using Code

This example sets the CurrentRowBorderLine property.

[VB]


[CS]


Current Cell

The cell in which the user is working is called the current cell. You can write code independent of the cell index if you reference the cells through the current cell. You can access the current cell using the GcMultiRow.CurrentCell (CurrentCell Property' in the on-line documentation) property.

Retrieving the Current Cell Index


Using Code

This example gets the current cell index.

[VB]


[CS]

Console.WriteLine("Current cell index: {0}", gcMultiRow1.CurrentCellPosition.CellIndex);

Moving the Current Cell

To move the current cell, set a new cell position in the GcMultiRow.CurrentCellPosition (CurrentCellPosition Property' in the on-line documentation) property.
Using Code
This example moves the current cell.

[VB]

[CS]
gcMultiRow1.CurrentCellPosition = new GrapeCity.Win.MultiRow.CellPosition(0, 1);

Accessing the Current Cell
You can use the GcMultiRow.CurrentCell property to access the current cell. The following code shows how to change the backcolor of the current cell.

Using Code
This example sets the backcolor for the current cell.

[VB]

[CS]

The GcMultiRow.CurrentCell property returns null (Nothing in Visual Basic) if the current cell does not exist, for example, when the template is not set in the GcMultiRow control or when the GcMultiRow control is in Display mode.

Adding Rows
The user can add rows using the row editor displayed in the bottom row of the GcMultiRow control when the AllowUserToAddRows (`AllowUserToAddRows Property` in the on-line documentation) property is set to True. The following image displays the row editor.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Using Code
This example sets the AllowUserToAddRows property.

[VB]
GcMultiRow1.AllowUserToAddRows = True

[CS]
If the GcMultiRow.AllowUserToAddRows property is set to False, the row editor is not displayed at the bottom of the GcMultiRow control and the user cannot add rows.

Using Code
This example sets the AllowUserToAddRows property to false.

[VB]
```vbnet
GcMultiRow1.AllowUserToAddRows = False
```

[CS]
```csharp
gcMultiRow1.AllowUserToAddRows = false;
```

Adding Rows Using Code
You can add a row using the RowCollection.Insert method or the RowCollection.Add method.

Using Code
This example uses the Add method.

[VB]
```vbnet
GcMultiRow1.Rows.Add()
```

[CS]
```csharp
gcMultiRow1.Rows.Add();
```

You cannot add rows using the RowCollection.Add method when the GcMultiRow control is bound to a data source. If the control is bound to a data source, add rows in the data source (for example, use the BindingSource.AddNew method or DataTable.Rows.Add method).

Adding Rows Using Row Count
You can change the row count of the GcMultiRow control to add additional rows to the control.

Using Code
This example increases the row count.

[VB]
```vbnet
```
You cannot change the row count using the GcMultiRow. **RowCount** (**RowCount Property** in the on-line documentation) property when the GcMultiRow control is bound to a data source. If the control is bound to a data source, change the row count (record count) in the data source.

**Adding Rows Using the Data Source**

When the GcMultiRow control is bound to the data source, you can add rows to the GcMultiRow control through the data source by setting both IBindingList.AllowNew properties to true.

**Events for Adding Rows**

The GcMultiRow. **RowsAdding** (**RowsAdding Event** in the on-line documentation) event occurs when you add a row to the GcMultiRow control. After the row is added, the GcMultiRow. **RowsAdded** (**RowsAdded Event** in the on-line documentation) event occurs. The GcMultiRow. **UserAddedRow** (**UserAddedRow Event** in the on-line documentation) event occurs when the user adds a row using the Row Editor.

**Deleting Rows**

You can delete rows in GcMultiRow.

**Users Deleting Rows**

The user can delete the current row by pressing the **Ctrl + Delete** keys if the GcMultiRow. **AllowUserToDeleteRows** (**AllowUserToDeleteRows Property** in the on-line documentation) property is set to True. This shortcut key is defined in GcMultiRow.**ShortcutKeyManager**.

**Using Code**

This example sets the AllowUserToDeleteRows property.

**[VB]**

```vbnet
GcMultiRow1.AllowUserToDeleteRows = True
```

**[CS]**

```csharp
gcMultiRow1.AllowUserToDeleteRows = true;
```

**Deleting Rows Using Code**

You can delete a row using the RowCollection. **Remove** (**Remove Method** in the on-line documentation) method.

**Using Code**

The following code deletes the current row.

**[VB]**

```vbnet
GcMultiRow1.Rows.RemoveAt(GcMultiRow1.CurrentRow.Index)
```
Dealing Rows Using Row Count

You can delete rows by decreasing the value of the **RowCount** ("RowCount Property' in the on-line documentation) property of the GcMultiRow control.

**Using Code**

This example decreases the row count.

**[VB]**

```vbnet
GcMultiRow1.RowCount -= 1
```

**[CS]**

```csharp
gcMultiRow1.RowCount -= 1;
```

Deleting Rows in the Data Source

When the GcMultiRow control is bound to a data source, you can delete rows from the control through the data source if both the IBindingList.AllowRemove properties of the data source are set to True.

Deleting All Rows

Call the RowCollection.**Clear** ("Clear Method' in the on-line documentation) method to delete all the rows.

**Using Code**

This example uses the Clear method.

**[VB]**

```vbnet
GcMultiRow1.Rows.Clear()
```

**[CS]**

```csharp
gcMultiRow1.Rows.Clear();
```

Events for Deleting Rows

The GcMultiRow.**RowsRemoving** ("RowsRemoving Event' in the on-line documentation) event occurs when you delete rows from the GcMultiRow control. After the rows have been deleted, the GcMultiRow.**RowsRemoved** ("RowsRemoved Event' in the on-line documentation) event occurs. When a user deletes rows using the **Ctrl + Delete** keys, the GcMultiRow.**UserDeletingRow** ("UserDeletingRow Event' in the on-line documentation) event is used.

An exception is thrown if the index of a non-existent row is specified.

Note that a new row is always displayed when the GcMultiRow.**AllowUserToAddRows** ("AllowUserToAddRows Property' in the on-line documentation) property is set to True. To make the row count zero, set the property to False.
and GcMultiRow.\texttt{UserDeletedRow ('UserDeletedRow Event' in the on-line documentation)} events occur.

### Getting and Setting the Row Count

You can use the GcMultiRow.\texttt{RowCount ('RowCount Property' in the on-line documentation)} property to get and set the number of rows in the GcMultiRow control.

> The row count cannot be changed using the GcMultiRow.\texttt{RowCount ('RowCount Property' in the on-line documentation)} property if the GcMultiRow control is bound to a data source. If the control is bound to a data source, you will need to change the row count (record count) in the data source itself.

#### Using Code

The following code sets the number of rows in the GcMultiRow control to 200.

**[VB]**

```vbnet
GcMultiRow1.RowCount = 200
```

**[CS]**

```cs
gcMultiRow1.RowCount = 200;
```

The following code outputs the current row count of the GcMultiRow control to the screen.

**[VB]**

```vbnet
Console.WriteLine(GcMultiRow1.RowCount)
```

**[CS]**

```cs
Console.WriteLine(gcMultiRow1.RowCount);
```

The following code gets the number of displayed rows in the GcMultiRow control. Partially displayed rows are also counted.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

Dim displayedRowCount As Integer = gcMultiRow1.Rows.GetRowCount(MultiRowElementStates.Displayed)
Console.WriteLine(displayedRowCount)
```

**[CS]**

```cs
using GrapeCity.Win.MultiRow;

int displayedRowCount = gcMultiRow1.Rows.GetRowCount(MultiRowElementStates.Displayed);
Console.WriteLine(displayedRowCount);
```

The following code gets the number of rows that are selected in the GcMultiRow control. Partially selected rows are not counted.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

Dim selectedRowCount As Integer = gcMultiRow1.Rows.GetRowCount(MultiRowElementStates.Selected)
Console.WriteLine(selectedRowCount)
```

**[CS]**

```cs
using GrapeCity.Win.MultiRow;

int selectedRowCount = gcMultiRow1.Rows.GetRowCount(MultiRowElementStates.Selected);
Console.WriteLine(selectedRowCount);
```
using GrapeCity.Win.MultiRow;

int selectedRowCount = gcMultiRow1.Rows.GetRowCount(MultiRowElementStates.Selected);
Console.WriteLine(selectedRowCount);

Freezing Rows

You can freeze the first row at the top or specific cells at the extreme left to prevent them from scrolling. Frozen rows can be used to provide an example of input to the user, compare data in rows, or for fixed display of row headers and row footers.

Freezing the Top Rows

You can specify the number of rows you want to fix in the GcMultiRow.FreezeTopRowCount ('FreezeTopRowCount Property' in the on-line documentation) property. This freezes the number of rows specified in the property, counting from the top.

Using Code

This example freezes the top two rows.

[VB]

    GcMultiRow1.RowCount = 10
    GcMultiRow1.FreezeTopRowCount = 2

[CS]

    gcMultiRow1.RowCount = 10;
    gcMultiRow1.FreezeTopRowCount = 2;

Freezing Cells on the Left Edge

You can specify the cells to be frozen on the left using the GcMultiRow.FreezeLeftCellName ('FreezeLeftCellName Property' in the on-line documentation) property or the GcMultiRow.FreezeLeftCellIndex ('FreezeLeftCellIndex Property' in the on-line documentation) property. This freezes (constantly displays) the cells from the left edge to the specified cell in the template.

Using Code

This example freezes a cell.

[VB]

' When specifying the cell index
    GcMultiRow1.FreezeLeftCellIndex = 0

' When specifying the cell name
    GcMultiRow1.FreezeLeftCellName = "TextBoxCell1"

[CS]

    // When specifying the cell index
    gcMultiRow1.FreezeLeftCellIndex = 0;
Freezing Cells on the Right Edge

You can specify the cells to be frozen in the GcMultiRow. FreezeRightCellName property or the GcMultiRow. FreezeRightCellIndex property. This freezes (constantly displays) the cells in the template from the right edge to the specified cell.

Using Code

This example freezes a cell.

[VB]
GcMultiRow1.FreezeRightCellIndex = 0

[CS]

gcMultiRow1.FreezeRightCellIndex = 0;

Freezing the Last Row

By specifying the number of rows to freeze in the GcMultiRow. FreezeBottomRowCount property, the specified number of rows from the bottom are frozen (constantly displayed).

Using Code

This example freezes the bottom two rows.

[VB]
GcMultiRow1.RowCount = 10
GcMultiRow1.FreezeBottomRowCount = 2

[CS]

gcMultiRow1.RowCount = 10;
gcMultiRow1.FreezeBottomRowCount = 2;

Customizing the Line between Non-scrolling and Scrolling Areas

The line that distinguishes the non-scrolling area can be customized using the GcMultiRow. FreezeLines property. You can specify the type and color of the line. When the type of line is set to None or the color is specified as Transparent, the line is not displayed. You can also use the border of the fixed row as the line. By default, the line is a black solid line (LineStyle=Thin, Color=Black).

Using Code
The following code freezes the last two rows and changes the line between the scrolling and non-scrolling rows to a solid blue line.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow
GcMultiRow1.RowCount = 100
GcMultiRow1.FreezeBottomRowCount = 2
Dim freezeLines As FreezeLines = New FreezeLines(New Line(LineStyle.Thin, Color.Blue))
GcMultiRow1.FreezeLines = freezeLines
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;
gcMultiRow1.RowCount = 100;
gcMultiRow1.FreezeBottomRowCount = 2;
FreezeLines freezeLines = new FreezeLines(new Line(LineStyle.Thin, Color.Blue));
gcMultiRow1.FreezeLines = freezeLines;
```

**Using the FreezeLines Editor**

If you select the GcMultiRow control on the Form designer and click the GcMultiRow.FreezeLines ('FreezeLines Property' in the on-line documentation) property button in the Properties window, an editor is displayed in which you can customize the border of the frozen area, as shown in the following figure.

![FreezeLines Editor](image)

**Selected Rows and Cells**

The GcMultiRow.SelectedCells ('SelectedCells Property' in the on-line documentation) property and the GcMultiRow.SelectedRows ('SelectedRows Property' in the on-line documentation) property can be used to get the selected rows and cells in the GcMultiRow control. Also, the GcMultiRow.AreAllCellsSelected ('AreAllCellsSelected Method' in the on-line documentation) method can be used to determine whether all the cells have been selected. See Selecting Cells using Headers for details on selecting cells using the headers.

**Selecting Rows and Cells**

Depending on how the selection has been made, the result of selecting a row is different from selecting all the cells in the row. The following figure depicts selecting a row, on the left, and selecting all the cells in the row, on the right.
Whether a row or its cells are selected by clicking the row header depends on the value of the `HeaderCell.SelectionMode` ("SelectionMode Property' in the on-line documentation) property. For more information, see Selecting Cells using Headers.

### Using Code

The following code enumerates the row number and index of selected cells.

**[VB]**

```vb
Imports GrapeCity.Win.MultiRow

For Each cell As Cell In GcMultiRow1.SelectedCells
    Console.WriteLine("Selected Cell: {0},{1}", cell.CellIndex, cell.RowIndex)
Next
```

**[CS]**

```cs
using GrapeCity.Win.MultiRow;

foreach (Cell cell in gcMultiRow1.SelectedCells)
{
    Console.WriteLine("Selected Cell: {0},{1}", cell.CellIndex, cell.RowIndex);
}
```

The following code enumerates the index of the selected row.

**[VB]**

```vb
Imports GrapeCity.Win.MultiRow

For Each row As Row In GcMultiRow1.SelectedRows
    Console.WriteLine("Selected Row: {0}", row.Index)
Next
```

**[CS]**

```cs
using GrapeCity.Win.MultiRow;

foreach (Row row in gcMultiRow1.SelectedRows)
{
    Console.WriteLine("Selected Row: {0}", row.Index);
}
```

It is also possible to verify whether a row or cell is selected by looking at the row or cell side.
Using Code

The following code gets only the selected rows from all the rows using the Row.Selected Property (Section 'Selected Property' in the on-line documentation) property.

[VB]

Imports GrapeCity.Win.MultiRow

For Each row As Row In GcMultiRow1.Rows
    If row.Selected = True Then
        Console.WriteLine("Selected Row: {0}", row.Index)
    End If
Next

[CS]

using GrapeCity.Win.MultiRow;

foreach (Row row in gcMultiRow1.Rows)
{
    if (row.Selected == true)
        Console.WriteLine("Selected Row: {0}", row.Index);
}

Restricting Cell Selection

GcMultiRow lets you restrict the selection of specific cells. Cells with restricted selection cannot be selected either by the keyboard or mouse or by using code. You might want to use this feature when you need to control focus movement or when the cell has been set as a placeholder. By default, all cells can be selected. The following figure illustrates cell selection.

It is not possible to restrict selection of rows. If you need to restrict row selection, you must restrict the selection of all the cells in the row.

Using Code

The following code restricts the selection of the second cell in the first row. The background color has been changed to differentiate between the cells.

[VB]

GcMultiRow1.Rows(0).Cells(1).Selectable = False

[CS]
The following code allows the selection of the second cell in the first row. By default, all the cells in the grid can be selected.

**[VB]**

GcMultiRow1.Rows(0).Cells(1).Selectable = True

**[CS]**

gcMultiRow1.Rows[0].Cells[1].Selectable = true;

### Disabled Cells

The GcMultiRow control allows you to disable specific cells. The user is not allowed to input into or click the cells that are disabled. Also, the appearance of these disabled cells can be changed.

Disabling cells differs from **Restricting Cell Selection**, as the cell appearance can be changed, and cell movement or selection is allowed in disabled cells. The appearance of disabled cells differs according to the implementation of **Cell Styles** and cell types.

It is not possible to set a disabled appearance for rows. In this case, all the cells in that row need to be disabled.

### Using Code

The following code disables the second cell in the first row. The background color has been changed to differentiate between the cells.

**[VB]**

GcMultiRow1.Rows(0).Cells(1).Enabled = False

**[CS]**

gcMultiRow1.Rows[0].Cells[1].Enabled = false;

The following code enables (removes the disabled state) for the second cell in the first row. By default, all the cells are enabled.

**[VB]**

GcMultiRow1.Rows(0).Cells(1).Enabled = True

**[CS]**

gcMultiRow1.Rows[0].Cells[1].Enabled = true;

### Hiding Cells and Rows
You can use the Visible property to hide specific cells or rows. When you hide a row, the space that was occupied by that row is now occupied by the next row. When you hide cells, the space occupied by those cells become empty. The layout of the cells does not change.

**Using Code**

The following code hides the first row. If the changes in the first row have not been committed (confirmed), or if the row does not exist, an exception is thrown.

**[VB]**

```vbnet
GcMultiRow1.Rows(0).Visible = False
```

**[CS]**

```csharp
gcMultiRow1.Rows[0].Visible = false;
```

To determine whether the rows are hidden, you can use the `GcMultiRow.GetState` method or the `Row.Visible` property (Section: Visible (Visible Property' in the on-line documentation) property).

**Using Code**

This example gets the row state.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

Console.WriteLine(gcMultiRow1.Rows(0).Visible)
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;

Console.WriteLine((gcMultiRow1.GetState(0) & MultiRowElementStates.Visible) == MultiRowElementStates.Visible);
Console.WriteLine(gcMultiRow1.Rows[0].Visible);
```

When the row is set as visible, you can use the `GcMultiRow.GetState` method or the `Row.Displayed` property to determine whether the row is displayed on the screen.

**Using Code**

This example gets the row state.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

Console.WriteLine(gcMultiRow1.Rows(0).Displayed)
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;

Console.WriteLine((gcMultiRow1.GetState(0) & MultiRowElementStates.Displayed) == MultiRowElementStates.Displayed);
Console.WriteLine(gcMultiRow1.Rows[0].Displayed);
```

Because the `GcMultiRow.GetState` method does not instantiate the target row, it provides good performance. Alternatively, the Visible property and the Displayed property are useful because they provide legible code and you can hide settings using these properties.

**Using Code**

The following code hides the second cell in the first row.

**[VB]**

```vbnet
GcMultiRow1.Rows(0).Cells(1).Visible = False
```

**[CS]**

```csharp
gcMultiRow1.Rows[0].Cells[1].Visible = false;
```
To determine whether a cell is hidden, you can use the `GcMultiRow.GetState` method or the `Cell.Visible` property.

### Using Code

This example gets the cell state.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

Console.WriteLine(gcMultiRow1.Rows(0).Cells(0).Visible)
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;

Console.WriteLine((gcMultiRow1.GetState(0, 0) & MultiRowElementStates.Visible) == MultiRowElementStates.Visible);
Console.WriteLine(gcMultiRow1.Rows[0].Cells[0].Visible);
```

When the cell is set as visible, you can use the `GcMultiRow.GetState` method or the `Cell.Displayed` property to determine whether the cell is displayed.

### Using Code

This example gets the cell state.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

Console.WriteLine((GcMultiRow1.GetState(0, 0) And MultiRowElementStates.Displayed) = MultiRowElementStates.Displayed)
Console.WriteLine(gcMultiRow1.Rows(0).Cells(0).Displayed)
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;

Console.WriteLine((gcMultiRow1.GetState(0, 0) & MultiRowElementStates.Displayed) == MultiRowElementStates.Displayed);
Console.WriteLine(gcMultiRow1.Rows[0].Cells[0].Displayed);
```

Because the `GcMultiRow.GetState` method does not instantiate the target cell, it provides good performance. Alternatively, the Visible property and the Displayed property are useful because they provide legible code and you can hide settings using these properties.

### Editing Cells

By default, the `GcMultiRow` control allows users to edit cells, confirm the edit, or cancel the edit. This topic describes editing in the `GcMultiRow` control.

#### User Editing

By default, the user can start editing a cell by:
- Double-clicking the cell
- Typing a character key
- Pressing the `Enter` key
- Pressing the `F2` key

To find out how the user initiated the editing, you can get the value of the `CellBeginEditEventArgs.BeginEditReason` property in the `CellBeginEdit` event.

To complete editing a cell, the user can:
- Press the `Enter` key
- Move the focus from the cell being edited

To find out how the user completed the editing, you can get the value of the `CellEndEditEventArgs.EndEditReason` property in the `CellEndEdit` event.

To cancel the editing of the cell (to discard the edited values), the user can press the `Esc` key.

### Using Code

You can change a cell's edit mode without using shortcut keys as in the following code.

**[VB]**

```vbnet
GcMultiRow1.Focus()
GcMultiRow1.BeginEdit(False)
```
A cell cannot change to edit mode when the grid is in the Display mode or when the cells are in read-only mode. Use the GcMultiRow.BeginEdit (‘BeginEdit Method’ in the on-line documentation) method to confirm the editing. Use the GcMultiRow.CancelEdit (‘CancelEdit Method’ in the on-line documentation) method to cancel the editing. To explicitly commit the edits to the datasource, use the GcMultiRow.CommitEdit (‘CommitEdit Method’ in the on-line documentation) method.

Using Code

This example sets the edit mode.

**VB**

```vbnet
```

**CS**

```csharp
```

The result of editing in the continuous input mode can be confirmed by pressing the **Enter** key or by moving the focus from the cell.

Canceling Editing

You can control whether the cell enters edit mode by using the **CellBeginEditEventArgs.CellBeginEdit** event. For example, you can specify whether to allow the editing of the cell based on the cell’s value.

Restricting Editing

If it is clear while designing the template that the editing of cells should not be allowed, you can restrict the editing of cells, disable the events, or make the cells read-only. See the following topics for details:

- **Restricting Cell Selection**
- **Disabled Cells**
- **Hiding Cells and Rows**
- **Read-Only Cells**

You can change the values of these properties at runtime also.

Getting the Edit State

You can check whether the cell is in edit mode by using the GcMultiRow.IsCurrentCellInEditMode (‘IsCurrentCellInEditMode Method’ in the on-line documentation) property or **Cell.IsInEditMode (‘IsInEditMode Method’ in the on-line documentation) property.**

Controls in Edit Mode

The cell editing control is shown in the cells currently in edit mode. For example, the **TextBoxEditingControl** handles the editing in a text box cell. The cell editing control can be accessed using the GcMultiRow.EditingControl (‘EditingControl Method’ in the on-line documentation) property and GcMultiRow.EditingControlShowing (‘EditingControlShowing Method’ in the on-line documentation) event.

Using Code

The following code compares the value before and during editing while validating the value.

**VB**

```vbnet
Imports GrapeCity.Win.MultiRow

Private Sub GcMultiRow1_CellValidating(ByVal sender As System.Object, ByVal e As CellValidatingEventArgs)
    Handles GcMultiRow1.CellValidating
    If TypeOf GcMultiRow1.CurrentCell Is TextBoxCell Then
        Dim beforeEdit As Object = GcMultiRow1.Rows(e.RowIndex).Cells(e.CellIndex).Value
        If beforeEdit IsNot Nothing Then
            Console.WriteLine("Value before editing:{0}", beforeEdit.ToString())
        Else
            Console.WriteLine("Value before editing:(None)")
        End If
        If GcMultiRow1.EditingControl IsNot Nothing Then
            Console.WriteLine("Value during editing:{0}", GcMultiRow1.EditingControl.Text)
        Else
            Console.WriteLine("Value during editing:(None)")
        End If
    Else
        Console.WriteLine("The present cell is not string type cell.")
    End If
End Sub
```

**CS**

```csharp
private void GcMultiRow1_CellValidating(object sender, CellValidatingEventArgs e)
{
    var cell = e.CurrentCell as TextBoxCell;
    if (cell != null)
    {
        var beforeEdit = e.Row.Cells[e.RowIndex].Value;
        if (beforeEdit != null)
        {
            Console.WriteLine("Value before editing:{0}", beforeEdit.ToString());
        }
        else
        {
            Console.WriteLine("Value before editing:(None)");
        }
        if (GcMultiRow1.EditingControl != null)
        {
            Console.WriteLine("Value during editing:{0}", GcMultiRow1.EditingControl.Text);
        }
        else
        {
            Console.WriteLine("Value during editing:(None)");
        }
    }
    else
    {
        Console.WriteLine("The present cell is not string type cell.");
    }
}
```
using GrapeCity.Win.MultiRow;

private void gcMultiRow1_CellValidating(object sender, CellValidatingEventArgs e)
{
    if (gcMultiRow1.CurrentCell is TextBoxCell)
    {
        object beforeEdit = gcMultiRow1.Rows[e.RowIndex].Cells[e.CellIndex].Value;
        if (beforeEdit != null)
            Console.WriteLine("Value before editing:{0}", beforeEdit.ToString());
        else
            Console.WriteLine("Value before editing:(None)\n");
        if (gcMultiRow1.EditingControl != null)
            Console.WriteLine("Value during editing:{0}", gcMultiRow1.EditingControl.Text);
        else
            Console.WriteLine("Value during editing:(None)\n");
    }
    else
    {
        Console.WriteLine("The present cell is not string type cell.\n");
    }
}

The following code changes the forecolor and backcolor only while editing.

[VB]
Imports GrapeCity.Win.MultiRow
Private Sub GcMultiRow1_EditingControlShowing(ByVal sender As System.Object, ByVal e As EditingControlShowingEventArgs) Handles GcMultiRow1.EditingControlShowing
    e.CellStyle.ForeColor = Color.Blue
End Sub

[CS]
using GrapeCity.Win.MultiRow;
private void gcMultiRow1_EditingControlShowing(object sender, EditingControlShowingEventArgs e)
{
    e.CellStyle.ForeColor = Color.Blue;
}

Displaying Error Messages

In the GcMultiRow control, error messages can be displayed for each row or cell individually.

Error Message for Rows

Error messages for rows can be set using the Row>ErrorText ('ErrorText Property' in the on-line documentation) property. An error message icon is displayed in the RowHeaderCell ('RowHeaderCell Class' in the on-line documentation) if a value other than the null character is set in the Row.ErrorText ('ErrorText Property' in the on-line documentation) property.

As shown in the following figure, error message contents are displayed in a tooltip when the pointer is over the error icon.

Using Code

This example specifies error text.

[VB]
GcMultiRow1.Rows(0).ErrorText = "Row Error"

[CS]
Error Icons and Tooltips are not displayed for the row if the RowHeaderCell does not exist in the corresponding row.

**Error Messages for Cells**

The error message for a cell can be set using the Cell. ErrorText (ErrorText Property' in the on-line documentation) property. An error message icon is displayed in the cell if a value other than the null character is set in the Cell. ErrorText (ErrorText Property' in the on-line documentation) property.

As shown in the following figure, the error message contents are displayed in a tooltip when the pointer is over the error icon.

---

**Using Code**

This example specifies error text for the cell.

**[VB]**

GcMultiRow1.Rows(0).Cells(0).ErrorText = "Cell Error"

**[CS]**

gcMultiRow1.Rows[0].Cells[0].ErrorText = "Cell Error";

An error icon is displayed regardless of whether the cell is being edited.

**Read-Only Cells**

Specific cells can be set to read-only using the Cell. ReadOnly (ReadOnly Property' in the on-line documentation) property. By default, the Cell.ReadOnly property is always set to True for cell types that are not editable, such as ButtonCell (ButtonCell Class' in the on-line documentation) and HeaderCell (HeaderCell Class' in the on-line documentation).

**Using Code**

Use the following code to set the current cell to read-only.

**[VB]**

GcMultiRow1.CurrentCell.ReadOnly = True

**[CS]**

gcMultiRow1.CurrentCell.ReadOnly = true;

**Using Code**

The following code prevents user input in string type cells that are in edit mode; however, users can select or copy the string in the cell.

**[VB]**

Imports GrapeCity.Win.MultiRow

Private Sub GcMultiRow1_EditingControlShowing(ByVal sender As System.Object, ByVal e As EditingControlShowingEventArgs) Handles GcMultiRow1.EditingControlShowing
    If TypeOf e.Control Is TextBoxEditingControl Then
        TryCast(e.Control, TextBoxEditingControl).ReadOnly = True
    End If
End Sub

**[CS]**

using GrapeCity.Win.MultiRow;

private void gcMultiRow1_EditingControlShowing(object sender, EditingControlShowingEventArgs e)
Cell Note

The GcMultiRow control provides the functionality of cell notes. By adding cell notes, you can display additional information on the grid.

Using Code

This example uses the CellNote ('CellNote Class' in the on-line documentation) class to add a cell note to a cell.

[VB]


[CS]


Start Editing a Cell Note

You can use the GcMultiRow.BeginEditCellNote ('BeginEditCellNote Method' in the on-line documentation) method to start editing a cell note that is added to the cell.

Using Code

This example starts editing a cell note.

[VB]

If cell.Note IsNot Nothing Then
    GcMultiRow1.BeginEditCellNote(pos)
End If

[CS]

if (cell.Note != null)
{
    gcMultiRow1.BeginEditCellNote(pos);
}

Removing Cell Notes

Set Null as a value in the Cell.Note ('Note Property' in the on-line documentation) property to remove a cell note that is added to the cell.

Using Code

This example removes a cell note.
Setting the Display Mode of a Cell Note

Use the CellNote.DisplayMode ("DisplayMode Property" in the on-line documentation) property to set the display mode of a cell note.

Using Code

This example sets the display mode of a cell note.

[VB]

Imports GrapeCity.Win.MultiRow
If Cell.Note IsNot Nothing Then
    If cell.Note.DisplayMode = CellNoteDisplayMode.AlwaysShown Then
        cell.Note.DisplayMode = CellNoteDisplayMode.HoverShown
    Else
        cell.Note.DisplayMode = CellNoteDisplayMode.AlwaysShown
    End If
End If

[CS]

using GrapeCity.Win.MultiRow;
Cell cell = gcMultiRow1.CurrentCell;
if (cell.Note != null)
{
    if (cell.Note.DisplayMode == CellNoteDisplayMode.AlwaysShown)
    {
        cell.Note.DisplayMode = CellNoteDisplayMode.HoverShown;
    } else {
        cell.Note.DisplayMode = CellNoteDisplayMode.AlwaysShown;
    }
}

Setting the Style of a Cell Note

Use the CellNote.Style ("Style Property" in the on-line documentation) property to set the style of a cell note.
The style of the connecting line of cell notes, is always a solid line of 1 pixel width.

- The value set in the CellNoteStyle.LineColor ("LineColor Property" in the on-line documentation) property, is applied as the color of the connecting line for the cell note.

Using Code

This example sets the cell note style.

[VB]

```vbnet
Imports GrapeCity.Win.MultiRow
Dim cellNote As CellNote = New CellNote()
cellNote.Style.BackColor = Color.Azure
cellNote.Style.ForeColor = Color.Blue
cellNote.Text = "Cell Note"
' Border style
cellNote.Style.LineColor = Color.Red
cellNote.Style.LineStyle = CellNoteLineStyle.Thin
cellNote.Style.LineWidth = 5
' Color of the triangle symbol
cellNote.Style.TriangleSymbolColor = Color.Blue
' Shadow display
cellNote.Style.ShowShadow = MultiRowTriState.True
GcMultiRow1.CurrentCell.Note = cellNote
```

[CS]

```csharp
using GrapeCity.Win.MultiRow;
CellNote cellNote = new CellNote();
cellNote.Style.BackColor = Color.Azure;
cellNote.Style.ForeColor = Color.Blue;
cellNote.Text = "Cell Note";
// Border style
cellNote.Style.LineColor = Color.Pink;
cellNote.Style.LineStyle = CellNoteLineStyle.Thin;
cellNote.Style.LineWidth = 5;
// Color of the triangle symbol
cellNote.Style.TriangleSymbolColor = Color.Blue;
// Shadow display
cellNote.Style.ShowShadow = MultiRowTriState.True;
gcMultiRow1.CurrentCell.Note = cellNote;
```

Style Application Rules for a Cell Note
If the Style property of the cell note is not null, the setting value of the GcMultiRow.DefaultCellNoteStyle (‘DefaultCellNoteStyle Property’ in the on-line documentation) property is applied, if the following values are set for each property; otherwise, the value that is set for each property, is applied.

<table>
<thead>
<tr>
<th>Property</th>
<th>Setting Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>Color.Empty</td>
</tr>
<tr>
<td>ForeColor</td>
<td></td>
</tr>
<tr>
<td>PatternColor</td>
<td></td>
</tr>
<tr>
<td>LineColor</td>
<td></td>
</tr>
<tr>
<td>TriangleSymbolColorColor</td>
<td></td>
</tr>
<tr>
<td>Font</td>
<td></td>
</tr>
<tr>
<td>BackgroundImage</td>
<td>Null</td>
</tr>
<tr>
<td>ContextMenuStrip</td>
<td></td>
</tr>
<tr>
<td>LineAdjustment</td>
<td>Inherit</td>
</tr>
<tr>
<td>TextEffect</td>
<td></td>
</tr>
<tr>
<td>Multiline</td>
<td></td>
</tr>
<tr>
<td>PatternStyle</td>
<td></td>
</tr>
<tr>
<td>TextAdjustment</td>
<td></td>
</tr>
<tr>
<td>TextAlign</td>
<td></td>
</tr>
<tr>
<td>TextVertical</td>
<td></td>
</tr>
<tr>
<td>BackgroundImageLayout</td>
<td></td>
</tr>
<tr>
<td>BackgroundGradientEffect</td>
<td></td>
</tr>
<tr>
<td>UseCompatibleTextRendering</td>
<td></td>
</tr>
<tr>
<td>WordWrap</td>
<td></td>
</tr>
<tr>
<td>LineStyle</td>
<td></td>
</tr>
<tr>
<td>ShowShadow</td>
<td></td>
</tr>
<tr>
<td>Padding</td>
<td>(-1,-1,-1,-1)</td>
</tr>
<tr>
<td>TextAngle</td>
<td>float.NaN</td>
</tr>
<tr>
<td>LineWidth</td>
<td></td>
</tr>
<tr>
<td>BackgroundImageOpacity</td>
<td>double.NaN</td>
</tr>
<tr>
<td>TextIndent</td>
<td>-1</td>
</tr>
</tbody>
</table>

The default values of the GcMultiRow.DefaultCellNoteStyle property are as follows.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>LightYellow</td>
</tr>
<tr>
<td>ForeColor</td>
<td>Setting value of the ForeColor property of the GcMultiRow control</td>
</tr>
<tr>
<td>Font</td>
<td>Setting value of the Font property of the GcMultiRow control</td>
</tr>
<tr>
<td>BackgroundGradientEffect</td>
<td>Null</td>
</tr>
<tr>
<td>Direction</td>
<td>Center</td>
</tr>
<tr>
<td>BackgroundImage</td>
<td>Null</td>
</tr>
<tr>
<td>BackgroundImageLayout</td>
<td>Tile</td>
</tr>
<tr>
<td>BackgroundImageOpacity</td>
<td>1</td>
</tr>
<tr>
<td>LineAdjustment</td>
<td>None</td>
</tr>
<tr>
<td>Multiline</td>
<td>True</td>
</tr>
<tr>
<td>Padding</td>
<td>(0,0,0,0)</td>
</tr>
<tr>
<td>PatternColor</td>
<td>WindowText</td>
</tr>
</tbody>
</table>
Creating Cell Notes

Use the Cell.Note property to add a cell note to a cell.

- The Edit control for cell notes does not support the rich text format.
- You can only set the cell note at runtime. An exception is thrown if you add a cell note at design time.
- You can set only one cell note in a cell. An exception is thrown if the same cell note is set to multiple cells.
- It is not possible to add cell notes to a new row that has not been committed.

Column Mode

The following sections describe the runtime operation when a column mode template is used.

- Current Column
- Adding Columns
- Deleting Columns
- Column Operation

Current Column

When performing column operations in column mode, you can use the row index to specify the target column since the rows and columns are reversed.

This section describes how to perform operations for the current column.

Retrieving the Index of the Current Column

Using Code

This example gets the current column.

**[VB]**

```vbnet
```

**[CS]**

```csharp
Console.WriteLine("Index of the current column: {0}", gcMultiRow1.CurrentCellPosition.RowIndex);
```

Moving the Current Column

To move the current column, set a new row position in the GcMultiRow.CurrentCellPosition (CurrentCellPosition Property in the on-line documentation) property.

Using Code

This example moves the current column.

**[VB]**

```vbnet
```

**[CS]**

```csharp
```

Adding Columns

When performing column operations using the column mode, you can use the row index to specify the target column
since the rows and columns are reversed.
This section explains the various methods for adding columns.

**Adding Columns by the Users**

The user can add columns using the editor displayed in the right most column of the GcMultiRow control when the `AllowUserToAddRows` ('AllowUserToAddRows Property' in the on-line documentation) property is set to True.

![Example Table](image)

**Using Code**

This example sets the `AllowUserToAddRows` property.

**[VB]**

```vbnet
GcMultiRow1.AllowUserToAddRows = True
```

**[CS]**

```csharp
gcMultiRow1.AllowUserToAddRows = true;
```

If the `GcMultiRow.AllowUserToAddRows` property is set to False, the editor is not displayed in the right most column of the GcMultiRow control, and the user cannot add any columns.

![Example Table](image)

**Using Code**

This example sets the `AllowUserToAddRows` property.

**[VB]**

```vbnet
GcMultiRow1.AllowUserToAddRows = False
```

**[CS]**

```csharp
gcMultiRow1.AllowUserToAddRows = false;
```

**Adding Columns Through Code**

You can add a column using the `RowCollection.Insert` ('Insert Method' in the on-line documentation) method
or the RowCollection.Add ('Add Method' in the on-line documentation) method.

Using Code

This example adds rows.

[VB]

GcMultiRow1.Rows.Add()

[CS]

gcMultiRow1.Rows.Add();

You cannot add rows using the RowCollection.Add method when the GcMultiRow control is bound to a data source. In such cases, add rows in the data source (for example, use the BindingSource.AddNew method or DataTable.Rows.Add method).

Using Code

This example inserts rows.

[VB]

GcMultiRow1.Rows.Insert(2)

[CS]

gcMultiRow1.Rows.Insert(2);

Adding Columns Using Column Count

You can change the column count of the GcMultiRow control to add additional columns to the control.
Using Code

This example increases the column count.

[VB]

```vbnet
GcMultiRow1.RowCount += 3
```

[CS]

```csharp
gcMultiRow1.RowCount += 3;
```

You cannot change the row count using the GcMultiRow.RowCount ('RowCount Property' in the on-line documentation) property when the GcMultiRow control is bound to a data source. In such cases, change the row count (record count) in the data source.

Adding Rows Using the Data Source

When the GcMultiRow control is bound to the data source, you can add rows to the GcMultiRow control though the data source by setting both IBindingList.AllowNew properties to true.

Events for Adding Rows

The GcMultiRow.RowsAdding ('RowsAdding Event' in the on-line documentation) event occurs when you add a row to the GcMultiRow control. After the row is added, the GcMultiRow.RowsAdded ('RowsAdded Event' in the on-line documentation) event occurs. The GcMultiRow.UserAddedRow ('UserAddedRow Event' in the on-line documentation) event occurs when the user adds a row using the Row Editor.

Deleting Columns

You can delete columns with shortcut keys or code.

Deleting Columns Using Keys

The user can delete the current column by pressing the Ctrl + Delete keys if the GcMultiRow.AllowUserToDeleteRows ('AllowUserToDeleteRows Property' in the on-line documentation) property is set to True. This shortcut key is defined in the GcMultiRowShortcutKeyManager ('ShortcutKeyManager Class' in the on-line documentation).
Using Code

This example sets the AllowUserToDeleteRows property.

[VB]

GcMultiRow1.AllowUserToDeleteRows = True

[CS]

gcMultiRow1.AllowUserToDeleteRows = true;

Deleting Columns Using Code

You can delete a column using the RowCollection.Remove (‘Remove Method’ in the on-line documentation) method.

Using Code

The following code deletes the current column.

[VB]

GcMultiRow1.Rows.RemoveAt(GcMultiRow1.CurrentRow.Index)

[CS]

gcMultiRow1.Rows.RemoveAt(gcMultiRow1.CurrentRow.Index);

An exception is thrown if the index of a non-existent column is specified.

Deleting Columns Using Column Count

You can delete columns by decreasing the value of the RowCount (‘RowCount Property’ in the on-line documentation) property of the GcMultiRow control.
Using Code
This example decreases the row count.

[VB]
GcMultiRow1.RowCount -= 3

[CS]
gcMultiRow1.RowCount -= 3;

You cannot change the column count using the GcMultiRow.RowCount property when the GcMultiRow control is bound to a data source. In such cases, modify the row count (number of records) of the data source.

Deleting Columns in the Data Source
When the GcMultiRow control is bound to a data source, you can delete columns from the control through the data source if both the IBindingList.AllowRemove properties of the data source are set to True.

Deleting All Columns
Execute the RowCollection.Clear ('Clear Method' in the on-line documentation) method to delete all the columns.

Using Code
This example uses the Clear ('Clear Method' in the on-line documentation) method.

[VB]
GcMultiRow1.Rows.Clear()

[CS]
gcMultiRow1.Rows.Clear();

A new column is always displayed when the GcMultiRow.AllowUserToAddRows ('AllowUserToAddRows Property' in the on-line documentation) property is set to True. To make the column count zero, set the property to False.

Events for Deleting Columns
The GcMultiRow.RowsRemoving ('RowsRemoving Event' in the on-line documentation) event occurs when you delete columns from the GcMultiRow control. After the columns have been deleted, the GcMultiRow.RowsRemoved ('RowsRemoved Event' in the on-line documentation) event occurs. When a user deletes columns by using the Ctrl + Delete keys, the GcMultiRow.UserDeletingRow ('UserDeletingRow Event' in the on-line documentation) and GcMultiRow.UserDeletedRow ('UserDeletedRow Event' in the on-line documentation) events occur.

Column Operation

This section describes the various operations that can be implemented in the Column mode.

Since the rows and columns are reversed in the column mode, when you perform a column operation, the processing that is carried out is the same as that for the row operation, in the default template mode (when the Template.LayoutMode ('LayoutMode Property' in the on-line documentation) property is set to TopToBottom).

Getting and Setting the Column Count

You can use the GcMultiRow.RowCount ('RowCount Property' in the on-line documentation) property to get and set the number of columns. For details, refer to Getting and Setting the Row Count.

The column count cannot be changed using the GcMultiRow.RowCount ('RowCount Property' in the on-line documentation) property, if the GcMultiRow control is bound to a data source. In such cases, you will need to change the row count (record count) in the data source itself.

Using Code

The following code sets the number of columns in the GcMultiRow control to 10.

[VB]

GcMultiRow1.RowCount = 10

[CS]

gcMultiRow1.RowCount = 10;

The following code outputs the current column count of the GcMultiRow control in the console window.

[VB]

Console.WriteLine(GcMultiRow1.RowCount)

[CS]

Console.WriteLine(gcMultiRow1.RowCount);

Show or Hide Columns

You can use the Row.Visible property (Section.Visible ('Visible Property' in the on-line documentation) property) to show or hide specific columns. For details, please refer to Hiding Cells and Rows.

Using Code

The following code toggles the visibility of the second column.
Freezing Columns

You can use the GcMultiRow. FreezeTopRowCount (‘FreezeTopRowCount Property’ in the on-line documentation) property to freeze the column at the extreme left. In addition, you can use the GcMultiRow. FreezeBottomRowCount (‘FreezeBottomRowCount Property’ in the on-line documentation) property to freeze the column at the extreme right. For details, refer to Freezing Rows.

Using Code

The following code freezes two columns on the extreme left, and one column on the extreme right.

[VB]
GcMultiRow1.FreezeTopRowCount = 2
GcMultiRow1.FreezeBottomRowCount = 1

[CS]
gcMultiRow1.FreezeTopRowCount = 2;
gcMultiRow1.FreezeBottomRowCount = 1;

Column Selection

You can select or delete the selected columns using the Row.Selected property (Section. Selected (‘Selected Property’ in the on-line documentation) property). For details, refer to Selection Mode.

Using Code

The following code selects the first column.

[VB]
GcMultiRow1.Rows(0).Selected = True

[CS]
gcMultiRow1.Rows[0].Selected = true;
Resizing Column Width

You can use the Cell.HorizoneResize (‘HorizontalResize Method’ in the on-line documentation) method to change the column width. For details, refer to Resizing.

Using Code

The following code increases the width of the first column by 10 pixels.

[VB]

GcMultiRow1.Rows(0).Cells(0).HorizontalResize(10)

[CS]

gcMultiRow1.Rows[0].Cells[0].HorizontalResize(10);

Read-Only Columns

Use the Row.ReadOnly property (Section.ReadOnly (‘ReadOnly Property’ in the on-line documentation) property) to set specific columns as read-only. For details, refer to Read-Only Cells.

Using Code

Use the following code to set the first column as read-only.

[VB]

GcMultiRow1.Rows(0).ReadOnly = True

[CS]

gcMultiRow1.Rows[0].ReadOnly = true;

Restricting Column Selection

Use the Row.Selectable property (Section.Selectable (‘Selectable Property’ in the on-line documentation) property) to restrict the selection of specific columns. For details, refer to Restricting Cell Selection.

Using Code

The following code restricts the selection of the first column.

[VB]

GcMultiRow1.Rows(0).Selectable = False

[CS]

gcMultiRow1.Rows[0].Selectable = false;

Managing Data

The following topics provide information about working with data.
Get and Set Data

You can get cell values from the GcMultiRow control using various methods. The values can be handled in MultiRow as shown below.

1. **Cell.Value** ([Value Property' in the on-line documentation]) - Save the value as an Object.
2. **Cell.FormattedValue** ([FormattedValue Property' in the on-line documentation]) - Change the cell value as a formatted value based on the data type and style.
3. **Cell.DisplayText** ([DisplayText Property' in the on-line documentation]) - Change the formatted value to a display value depending on the specific settings of each cell type.

Values other than the cell values (Cell.Value ([Value Property' in the on-line documentation] property) are read-only.

Get and Set Cell Values

Cell values are set with the GcMultiRow.SetValue ([DisplayText Property' in the on-line documentation]) method.

Using Code

The following code sets the value in the second cell of the first row.

**[VB]**

GcMultiRow1.SetValue(0, 1, "ABC")

**[CS]**

gcMultiRow1.SetValue(0, 1, "ABC");

Use the GcMultiRow.GetValue ([GetValue Method' in the on-line documentation]) method to get the cell values.

Using Code

The following code gets the value of the second cell of the first row and shows it in the console.

**[VB]**

Dim cellValue As Object = GcMultiRow1.GetValue(0, 1) Console.WriteLine(cellValue)

**[CS]**

object cellValue = gcMultiRow1.GetValue(0, 1); Console.WriteLine(cellValue);

You can also get and set cell values through the row and cell instances. Use the GcMultiRow.GetValue ([GetValue Method' in the on-line documentation]) and GcMultiRow.SetValue ([SetValue Method' in the on-line documentation]) methods.
methods to work with the values.

Using Code

This example gets and sets values in cells.

[VB]

```vbnet
GcMultiRow1.Rows(0).Cells(1).Value = "DEF"
GcMultiRow1(0, 2).Value = "GHI"
Console.WriteLine(GcMultiRow1(0, 2).Value)
```

[CS]

```csharp
gcMultiRow1.Rows[0].Cells[1].Value = "DEF";
gcMultiRow1[0, 2].Value = "GHI";
Console.WriteLine(gcMultiRow1[0, 2].Value);
```

The cell values are always of the System.Object type. You can confirm this by using the Cell.ValueType ('ValueType Property' in the on-line documentation) property to check the data type of the cell value.

Get the Formatted Cell Value

Use the GcMultiRow.GetFormattedValue ('GetFormattedValue Method' in the on-line documentation) method to get the formatted value related to the cell.

Using Code

The following code gets the value of the second cell of the first row and outputs it to the console.

[VB]

```vbnet
Dim formattedValue As Object = GcMultiRow1.GetFormattedValue(0, 1)
Console.WriteLine(formattedValue)
```

[CS]

```csharp
object formattedValue = gcMultiRow1.GetFormattedValue(0, 1);
Console.WriteLine(formattedValue);
```

You can also get and set the formatted cell values through the row and cell instances. Use the GcMultiRow.GetFormattedValue ('GetFormattedValue Method' in the on-line documentation) method to work with formatted cell values.

Using Code

This example gets the formatted cell value.

[VB]

```vbnet
Dim formattedValue As Object = GcMultiRow1(0, 1).FormattedValue
Console.WriteLine(formattedValue)
```

[CS]

```csharp
object formattedValue = gcMultiRow1[0, 1].FormattedValue;
```
Console.WriteLine(formattedValue);

The data type of the formatted value can be returned with the Cell.FormattedValueType ("FormattedValueType Property' in the on-line documentation) property. Unlike the Cell.ValueType property, the Cell.FormattedValueType ("FormattedValueType Property' in the on-line documentation) property returns the data type aligned with the actual display.

**Using Code**

The result of the following code is String for String type, and Date (System.DateTime in C#) for Date type.

**[VB]**

```vbnet
Console.WriteLine(GcMultiRow1(0, 1).FormattedValueType)
```

**[CS]**

```csharp
Console.WriteLine(gcMultiRow1[0, 1].FormattedValueType);
```

If the cell is still in edit mode or the values have not been saved, the Cell.FormattedValue ("FormattedValue Property' in the on-line documentation) property may not return the current value (since the values have still not been confirmed). In order to get the formatted cell value, regardless of editing state or saving, use the GcMultiRow.GetEditedFormattedValue ("GetEditedFormattedValue Method' in the on-line documentation) method or the Cell.EditedFormattedValue ("EditedFormattedValue Property' in the on-line documentation) property. The cell value format can be customized using the CellFormatting ("CellFormatting Event' in the on-line documentation) event.

**Get the Displayed Cell Values**

Use the GcMultiRow.GetDisplayText ("GetDisplayText Method' in the on-line documentation) method to get the values displayed in the cells.

**Using Code**

This example gets the value in the cell.

**[VB]**

```vbnet
Dim displayValue As String = GcMultiRow1.GetDisplayText(0, 1)
Console.WriteLine(displayValue)
```

**[CS]**

```csharp
string displayValue = gcMultiRow1.GetDisplayText(0, 1);
Console.WriteLine(displayValue);
```

You can also get and set the displayed values through the row and cell instances. Use the GcMultiRow.GetDisplayText ("GetDisplayText Method' in the on-line documentation) method to work with the displayed values.

**Using Code**

This example uses the GetDisplayText method.

**[VB]**

```vbnet
Console.WriteLine(GcMultiRow1(0, 2).DisplayText)
```
Get the Values in Edit mode

The values in edit mode can be returned using System.Windows.Forms.Control.Text.

Using Code

The following code gets the values in edit mode.

[VB]

```vbnet
Dim editingControlValue As String = GcMultiRow1.EditingControl.Text
Console.WriteLine(editingControlValue)
```

[CS]

```csharp
string editingControlValue = gcMultiRow1.EditingControl.Text;
Console.WriteLine(editingControlValue);
```

Get the Default Cell Values

In order to access the default cell values for a newly added row, access the cell values of the template. Use the NewRowNeeded ('NewRowNeeded Event' in the on-line documentation) event to set the default value in the cells when a new row is added.

Data Binding Modes

The GcMultiRow control supports three data binding modes and the user can select the mode they want to use.

Unbound (Disconnected) Mode

Unbound mode is applicable when displaying relatively small amounts of data. In this mode, there is no direct connection with the data source, and all the data is stored in the GcMultiRow control. This mode can be used when data is provided in the read-only mode in the form of tables, or when you are coding for the data source connectivity independently. The GcMultiRow control works in unbound mode when the GcMultiRow.DataSource ('DataSource Property' in the on-line documentation) property is empty, and the GcMultiRow.VirtualMode ('VirtualMode Property' in the on-line documentation) property is set to False. For details on working with data in the unbound mode, refer to Get and Set Data.

Bound Mode

Bound mode implies that you are connecting to a data source and are reading, writing, and synchronizing the data. In this mode, all the data is saved in the data source, and the values in the GcMultiRow control are updated automatically. When you make changes in the GcMultiRow control, the changes get updated automatically. This mode can be used in
structing general database applications. The GcMultiRow control works in bound mode when a data source has been set with the GcMultiRow.DataSource ('DataSource Property' in the on-line documentation) property and the GcMultiRow.VirtualMode ('VirtualMode Property' in the on-line documentation) property has been set to False.

Using Code

The following code displays the data source values in bound mode.

[VB]

```vbnet
GcMultiRow1.DataSource = myDataSet
GcMultiRow1.DataMember = myDataSet.Tables(0).TableName
```

[CS]

```csharp
gcMultiRow1.DataSource = myDataSet;
gcMultiRow1.DataMember = myDataSet.Tables[0].TableName;
```

For details on bound mode, refer to Connecting to Database.

Virtual Mode

Virtual mode applies to cases where large amounts of data need to be handled quickly. The GcMultiRow control does not read the data automatically in this mode. The developer handles the timing for updates and changes to the display and transfers of data. This mode is suited for cases where you need to improve the performance by maintaining only the minimum required data.

- When virtual mode is enabled, the built-in filters and sorting feature of the column header cell do not work.
- You cannot use the GcMultiRow.Sort ('Sort Method' in the on-line documentation) method in virtual mode.

The GcMultiRow control works in virtual mode when the GcMultiRow.VirtualMode ('VirtualMode Property' in the on-line documentation) property is set to True. In virtual mode, the CellValueChanged ('CellValueNeeded Event' in the on-line documentation) event is fired when data is required in the grid. This includes the scrolling and re-rendering of the form.

Using Code

The following code displays the cell coordinates using the GcMultiRow.CellValueChanged ('CellValueNeeded Event' in the on-line documentation) event.

[VB]

```vbnet
Imports GrapeCity.Win.MultiRow

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    GcMultiRow1.ReadOnly = True
    GcMultiRow1.VirtualMode = True
    GcMultiRow1.RowCount = 1000
End Sub

Private Sub GcMultiRow1_CellValueNeeded(ByVal sender As System.Object, ByVal e As CellValueEventArgs) Handles GcMultiRow1.CellValueNeeded
    e.Value = String.Format("\{0\},\{1\}\", e.RowIndex, e.CellIndex)
```

Copyright © GrapeCity, inc. All rights reserved.
using GrapeCity.Win.MultiRow;

private void Form1_Load(object sender, EventArgs e)
{
    gcMultiRow1.ReadOnly = true;
    gcMultiRow1.VirtualMode = true;
    gcMultiRow1.RowCount = 1000;
}

private void gcMultiRow1_CellValueNeeded(object sender, CellValueEventArgs e)
{
    e.Value = string.Format("{0},{1}", e.RowIndex, e.CellIndex);
}

The **CellValuePushed ("CellValuePushed Event" in the on-line documentation)** event fires when the data is required in the grid. This event fires when the user or the developer inputs data in the grid.

### Sorting Data

You can sort rows in the GcMultiRow control using single or multiple cell values.

> When the GcMultiRow.AllowUserToAddRows ("AllowUserToAddRows Property" in the on-line documentation) property is set to True, the bottom-most row is excluded from sorting.

#### Sorting Rows Based on Single Cell Value

Use the GcMultiRow.Sort ("Sort Method" in the on-line documentation) method to sort the rows based on a single cell value. You can specify the cells that will be the basis for sorting, whether the sorting will be in ascending or descending order, or if the sorting needs to differentiate between capital and small letters.

#### Using Code

The following code uses a default template and sorts the rows based on the value of the second cell.

```vb
Imports GrapeCity.Win.MultiRow

' Set the default template
GcMultiRow1.Template = Template.Default
GcMultiRow1.AllowUserToAddRows = False
GcMultiRow1.RowCount = 5

' Set the sample data
GcMultiRow1.Rows(0).SetValues(New String(2) {"E", "1", "VB"})
GcMultiRow1.Rows(1).SetValues(New String(2) {"D", "2", "VB"})
GcMultiRow1.Rows(2).SetValues(New String(2) {"C", "3", "CS"})
GcMultiRow1.Rows(3).SetValues(New String(2) {"B", "4", "VB"})
GcMultiRow1.Rows(4).SetValues(New String(2) {"A", "5", "CS"})

' Sort in ascending order with the first cell as the base
GcMultiRow1.Sort(0, SortOrder.Ascending)

[CS]

using GrapeCity.Win.MultiRow;

// Set the default template
gcMultiRow1.Template = Template.Default;
gcMultiRow1.AllowUserToAddRows = false;
gcMultiRow1.RowCount = 5;

// Set the sample data
gcMultiRow1.Rows[0].SetValues(new string[3] {"E", "1", "VB"});

// Sort in ascending order with the first cell as the base
gcMultiRow1.Sort(0, SortOrder.Ascending);

Use the GcMultiRow.SortOrder (SortOrder Property in the on-line documentation) property to get the current sort status if you used the Sort method to sort.

Sort the Rows Based on Multiple Cell Values

If the same value exists in the first cell, and you would like to use the next cell value for sorting, you can set the condition for sorting in the instance of the SortItem (SortItem Class in the on-line documentation) class.

Using Code

This example uses the SortItem class.

[VB]

Imports GrapeCity.Win.MultiRow

' Set the default template
GcMultiRow1.Template = Template.Default
GcMultiRow1.AllowUserToAddRows = False
GcMultiRow1.RowCount = 5

' Set the sample data
GcMultiRow1.Rows(0).SetValues(New String(2) {"B", "1", "VB"})
GcMultiRow1.Rows(1).SetValues(New String(2) {"B", "2", "VB"})
GcMultiRow1.Rows(2).SetValues(New String(2) {"C", "3", "CS"})
GcMultiRow1.Rows(3).SetValues(New String(2) {"B", "4", "VB"})
GcMultiRow1.Rows(4).SetValues(New String(2) {"A", "5", "CS"})

' Create the basis for sorting
Dim sortItem1 As New SortItem(0, SortOrder.Ascending)
Dim sortItem2 As New SortItem(1, SortOrder.Ascending)
' Sort the rows
GcMultiRow1.Sort(New SortItem(1) {sortItem1, sortItem2}, 0, GcMultiRow1.RowCount - 1)

[CS]
using GrapeCity.Win.MultiRow;

// Set the default template
gcMultiRow1.Template = Template.Default;
gcMultiRow1.AllowUserToAddRows = false;
gcMultiRow1.RowCount = 5;

// Set the sample data
gcMultiRow1.Rows[0].SetValues(new string[3] {"B", "1", "VB"});

// Create the basis for sorting
SortItem sortItem1 = new SortItem(0, SortOrder.Ascending);
SortItem sortItem2 = new SortItem(1, SortOrder.Ascending);
// Sort the rows
gcMultiRow1.Sort(new SortItem[2] {sortItem1, sortItem2}, 0, gcMultiRow1.RowCount - 1);

User-defined Sorting

To independently create a basis for sorting, create a class that implements the System.Collections.IComparer interface and assign its instance to the GcMultiRow.Sort (Sort Method in the on-line documentation) method.

Using Code

The following code gives a higher priority to Null compared to a value.

[VB]

Imports GrapeCity.Win.MultiRow

' Set the default template
GcMultiRow1.Template = Template.Default
GcMultiRow1.AllowUserToAddRows = False
GcMultiRow1.RowCount = 5

' Set the sample data
GcMultiRow1.Rows(0).Cells(0).Value = 3
GcMultiRow1.Rows(1).Cells(0).Value = 1
GcMultiRow1.Rows(2).Cells(0).Value = 2
GcMultiRow1.Rows(3).Cells(0).Value = Nothing
GcMultiRow1.Rows(4).Cells(0).Value = 0

' Sort the rows
GcMultiRow1.Sort(0, SortOrder.Ascending, New NullComparer())

' Sort giving priority to Null
Public Class NullComparer
    Implements System.Collections.IComparer

    Public Function Compare(ByVal x As Object, ByVal y As Object) As Integer Implements System.Collections.IComparer.Compare
        If x = y Then
            Return 0
        End If
If x Is Nothing Then
    Return -1
End If
If y Is Nothing Then
    Return 1
End If

Dim ix As Integer = DirectCast(x, Integer)
Dim iy As Integer = DirectCast(y, Integer)
Return ix - iy
End Function
End Class

[CS]

using GrapeCity.Win.MultiRow;

// Set the default template
gcMultiRow1.Template = Template.Default;
gcMultiRow1.AllowUserToAddRows = false;
gcMultiRow1.RowCount = 5;

// Set the sample data
gcMultiRow1.Rows[0].Cells[0].Value = 3;
gcMultiRow1.Rows[1].Cells[0].Value = 1;
gcMultiRow1.Rows[2].Cells[0].Value = 4;
gcMultiRow1.Rows[3].Cells[0].Value = null;
gcMultiRow1.Rows[4].Cells[0].Value = 0;

// Sort the rows
gcMultiRow1.Sort(0, SortOrder.Ascending, new NullComparer());

public class NullComparer : System.Collections.IComparer
{
    int System.Collections.IComparer.Compare(object x, object y)
    {
        if (x == y)
            return 0;
        if (x == null)
            return -1;
        if (y == null)
            return 1;

        int ix = (int)x;
        int iy = (int)y;
        return ix - iy;
    }
}

Working with Clipboard

The user can copy and paste cell values in GcMultiRow using the clipboard.

Allowing Clipboard Operations
To allow clipboard operations, set the GcMultiRow.AllowClipboard (‘AllowClipboard Property’ in the on-line documentation) property to True. The following clipboard operations are allowed in this case.

- Cut Cell Values
- Copy Cell Values
- Paste Cell Values

The shortcut keys for the clipboard are the same as Windows standard keys. For details, refer to Shortcut Keys.

Clipboard Data Format

The data copied to the clipboard uses a comma to separate the cell values of the Cell.FormattedValue (‘FormattedValue Property’ in the on-line documentation) property, and separates the rows with a new line code (vbCrLf in Visual Basic, “\r\n” in C#). The cell order is based on the cell index (Cell.Index). To change the cell index at design time, refer to Changing Cell Index in the Designer.

Clipboard Events

When the clipboard operations are executed on the GcMultiRow control, the GcMultiRow.ClipboardOperating (‘ClipboardOperating Event’ in the on-line documentation) event occurs. You can use this event to cancel the clipboard operation using your own conditions. For example, you can cancel the built-in clipboard operations and implement your own clipboard operations.

PasswordChar property and InvalidOperationException

When you set the value of the TextBoxCell.PasswordChar (‘PasswordChar Property’ in the on-line documentation) property, you cannot copy the cell values to the clipboard. An InvalidOperationException is thrown if the user tries to copy in this state. In order to suppress the display of the message box due to this exception, use the GcMultiRow.DataError (‘DataError Event’ in the on-line documentation) event.

Set Default Value in New Row

You can reduce the load on the user by automatically setting a default value in a newly added row. There are two ways you can set the default value in the GcMultiRow control.

Set the Template

The value specified in the Cell.Value (‘Value Property’ in the on-line documentation) property is used as the default value in the newly added row.

Set in the Data Source

When a data source is connected to the GcMultiRow control and a default value is specified in the column of the data source (System.Data.DataColumn.DefaultValue), this default value is used for the new grid row. This value has priority compared to the Cell.Value (‘Value Property’ in the on-line documentation) property.

Set using DefaultValuesNeeded Event

You can specify the default value in a newly added row using code in the DefaultValuesNeeded (‘DefaultValuesNeeded Event’ in the on-line documentation) event.

Using Code

This example uses the DefaultValuesNeeded event.

[VB]

Imports GrapeCity.Win.MultiRow
Cell Styles

The formatting and styles that are applied to cells in the MultiRow control are supplied in the `CellStyle` class. The settings of the `CellStyle` class can be changed at runtime as well as design time. This section explains the rules and factors about cell styles that can be used in MultiRow. Refer to **Conditional Cell Styles** for details on how to set styles based on conditions.

- Rules for Applying Styles
- Borders
- BackColor and ForeColor
- Background Pattern
- Background Gradation
- Tag
- Image
- Margins and Padding
- Multiline and Word Wrap
- Text Formatting
- Text Indent
- GDI+ Compatibility Mode
- Mouse Hover Styles
- User-defined Word Wrap
- Customizing the Justification of Single-byte Characters

**Rules for Applying Styles**

**Row Cell Style**

The style for the cells in the row is applied using the following steps at runtime. The priority increases as the number increases.

1. Default cell style for the GcMultiRow control (GcMultiRow.**DefaultCellStyle** in the on-line documentation)
2. Default Row Style (GcMultiRow.RowsDefaultCellStyle ('RowsDefaultCellStyle Property' in the on-line documentation))

3. Default Style for Alternating Rows (GcMultiRow.AlternatingRowsDefaultCellStyle ('AlternatingRowsDefaultCellStyle Property' in the on-line documentation))

4. Default Style for Each Row (Row.DefaultCellStyle ('DefaultCellStyle Property' in the on-line documentation))

5. Style of Each Cell (Cell.Style ('Style Property' in the on-line documentation))

When the style properties are not set, the properties one level up are inherited. For example, when the background color of the cell is set in the default style of GcMultiRow control, and there are no background colors set in the styles after that, the backcolor of the default style of the GcMultiRow control is set into the cells. If the backcolor is set into the styles after that, that backcolor is overwritten. You can confirm the style that has been applied to the cells at runtime by checking the Cell.InheritedStyle ('InheritedStyle Property' in the on-line documentation) property.

Using Code

This example gets the backcolor style.

[VB]

Imports GrapeCity.Win.MultiRow

Dim style As CellStyle = GcMultiRow1.Rows(0).Cells(0).InherittedStyle
Console.WriteLine(style.BackColor)

[CS]

using GrapeCity.Win.MultiRow;

CellStyle style = gcMultiRow1.Rows[0].Cells[0].InherittedStyle;
Console.WriteLine(style.BackColor);

Column Header and Footer Cell Styles

The cell styles for column headers and footers are applied at runtime, in the following order. The rules of priority and inheritance are the same as those for cell styles.
1. GcMultiRow Control Default Style (GcMultiRow.DefaultCellStyle)

2. The default header style of the column header (GcMultiRow.ColumnHeaderCellStyle ('ColumnHeadersDefaultHeaderCellStyle Property' in the on-line documentation)) or default header style of the column footer (GcMultiRow.ColumnHeaderCellStyle ('ColumnFootersDefaultHeaderCellStyle Property' in the on-line documentation))

3. Style of each Header Cell (HeaderCell.Style ('Style Property' in the on-line documentation))

Borders

You can use four types of borders in cells. In the designer, you can use the context menu of the cell or click on the Cell.Style.Border property in the Properties window to display the Border dialog.

Default Borders

By default, the borders that are set on the cells are silver lines on all four sides of the cells. This setting has been inherited from the Template.Style.Border property.

Normal Borders

If you use the Border ('Border Class' in the on-line documentation) class, you can draw borders on all four sides of the cells. The borders can be set on either or all of the four sides, running from one corner to the other. You can also set color and style for each side.
Using Code

This example sets the style border.

[VB]

Imports GrapeCity.Win.MultiRow

[CS]

using GrapeCity.Win.MultiRow;

gcMultiRow1.Rows[0].Cells[0].Style.Border = new Border(LineStyle.Double, Color.Red);

3D Borders

You can draw borders with 3D effects if you use the **ThreeDBorder** (**ThreeDBorder Class** in the on-line documentation) class. The border is displayed on all four sides, enabling 3D display through usage of light and shadow.
Using Code

This example sets a 3D border.

[VB]

Imports GrapeCity.Win.MultiRow

GcMultiRow1.Rows(0).Cells(0).Style.Border = New ThreeDBorder(ThreeDEffect.Sunken)

[CS]

using GrapeCity.Win.MultiRow;

gcMultiRow1.Rows[0].Cells[0].Style.Border = new ThreeDBorder(ThreeDEffect.Sunken);

Rounded Borders

You can draw a rounded border if you use the **RoundedBorder** (**RoundedBorder Class' in the on-line documentation**)
class. You can specify the border color and style for the four sides and four corners. You can also specify the angle of the rounded
corner and whether anti-alias should be applied.
Using Code

This example creates a rounded border.

[VB]

Imports GrapeCity.Win.MultiRow

GcMultiRow1.Rows(0).Cells(0).Style.Border = New RoundedBorder(LineStyle.Double, Color.Red, 0.25F)

[CS]

using GrapeCity.Win.MultiRow;

gcMultiRow1.Rows[0].Cells[0].Style.Border = new RoundedBorder(LineStyle.Double, Color.Red, 0.25f);

Inheriting Borders

When no borders have been set, the style gets inherited from the one above. Refer to Rules for Applying Styles for details on the inheritance rules.

BackColor and ForeColor

You can set three types of backcolors depending on the status of the cells.

Deselected Color

The cell colors, when they are not selected, can be set with the CellStyle.BackColor ('BackColor Property' in the on-line documentation) and CellStyle.ForeColor ('ForeColor Property' in the on-line documentation) properties.

Using Code

This example sets the forecolor and backcolor.

[VB]
Selected Color

The selected cell color can be set with the CellStyle. `SelectionBackColor` ("SelectionBackColor Property' in the on-line documentation) and CellStyle. `SelectionForeColor` ("SelectionForeColor Property' in the on-line documentation) properties.

Using Code

This example sets the `SelectionBackColor` ("SelectionBackColor Property' in the on-line documentation) and `SelectionForeColor` ("SelectionForeColor Property' in the on-line documentation) properties.

[VB]

```vbnet
GcMultiRow1.DefaultCellStyle.SelectionBackColor = Color.Red
GcMultiRow1.DefaultCellStyle.SelectionForeColor = Color.White
```

[CS]

```csharp
gcMultiRow1.DefaultCellStyle.SelectionBackColor = Color.Red;
gcMultiRow1.DefaultCellStyle.SelectionForeColor = Color.White;
```

Edit Mode Color

The color of the cell when it is in edit mode can be set using the CellStyle. `EditingBackColor` ("EditingBackColor Property' in the on-line documentation) and CellStyle. `EditingForeColor` ("EditingForeColor Property' in the on-line documentation) properties.

Using Code

This example sets the `EditingBackColor` ("EditingBackColor Property' in the on-line documentation) and `EditingForeColor` ("EditingForeColor Property' in the on-line documentation) properties.

[VB]

```vbnet
GcMultiRow1.DefaultCellStyle.EditingBackColor = Color.Yellow
GcMultiRow1.DefaultCellStyle.EditingForeColor = Color.Green
```

[CS]

```csharp
gcMultiRow1.DefaultCellStyle.EditingBackColor = Color.Yellow;
```

Disabled Color

The disabled cell color can be set with the CellStyle. `DisabledBackColor` ("DisabledBackColor Property' in the on-line documentation) and CellStyle. `DisabledForeColor` ("DisabledForeColor Property' in the on-line documentation) properties.
Using Code

This example sets the DisabledBackColor ("DisabledBackColor Property" in the on-line documentation) and DisabledForeColor ("DisabledForeColor Property" in the on-line documentation) properties.

[VB]

GcMultiRow1.DefaultCellStyle.DisabledBackColor = SystemColors.Control
GcMultiRow1.DefaultCellStyle.DisabledForeColor = SystemColors.ControlLight

[CS]

gcMultiRow1.DefaultCellStyle.DisabledBackColor = SystemColors.Control;
gcMultiRow1.DefaultCellStyle.DisabledForeColor = SystemColors.ControlLight;

Background Color and Visual Style

When the Windows Visual Style is active for the cells, the backcolor is not shown for the button or the headers. To enable the backcolor when the visual style has been applied, you need to set the ButtonCell.FlatStyle property (ButtonCellBase.FlatStyle ("FlatStyle Property" in the on-line documentation) property) to Standard and the ButtonCell.UseVisualStyleBackColor property (ButtonCellBase.UseVisualStyleBackColor ("UseVisualStyleBackColor Property" in the on-line documentation) property) to False.

Using Code

This example sets the style.

[VB]

Imports GrapeCity.Win.MultiRow

Dim buttonCell As New ButtonCell()
buttonCell.FlatStyle = FlatStyle.Standard
buttonCell.UseVisualStyleBackColor = False
buttonCell.Style.BackColor = Color.Blue
buttonCell.Value = "buttonCell"
GcMultiRow1.Template = Template.CreateGridTemplate(New Cell() {buttonCell})

[CS]

using GrapeCity.Win.MultiRow;

ButtonCell buttonCell = new ButtonCell();
buttonCell.FlatStyle = FlatStyle.Standard;
buttonCell.UseVisualStyleBackColor = false;
buttonCell.Style.BackColor = Color.Blue;
buttonCell.Value = "buttonCell";
gcMultiRow1.Template = Template.CreateGridTemplate(new Cell[] { buttonCell });

Background Pattern

You can set the background pattern in the cell style. The appearance supported in the background pattern is the same as the one supported by the .NET Framework (System.Drawing.Drawing2D.HatchStyle).

The following table shows examples of background patterns.
### MultiRowHatchStyle

<table>
<thead>
<tr>
<th>Inherit</th>
<th>Image after Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch value from Parent</td>
<td>None</td>
</tr>
<tr>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Horizontal</td>
<td></td>
</tr>
<tr>
<td>Vertical</td>
<td></td>
</tr>
<tr>
<td>ForwardDiagonal</td>
<td></td>
</tr>
<tr>
<td>BackwardDiagonal</td>
<td></td>
</tr>
<tr>
<td>LargeGrid</td>
<td></td>
</tr>
<tr>
<td>DiagonalCross</td>
<td></td>
</tr>
<tr>
<td>Percent05</td>
<td></td>
</tr>
<tr>
<td>Percent10</td>
<td></td>
</tr>
<tr>
<td>Percent20</td>
<td></td>
</tr>
<tr>
<td>Percent25</td>
<td></td>
</tr>
</tbody>
</table>

### Using the Designer

1. Select the cells that need a background pattern to be set (for example: textBoxCell1).
2. Select the desired background pattern from the drop-down list of the textBoxCell1.Style.PatternStyle property in the Properties window.
3. Select the desired background pattern color from the drop-down window of the textBoxCell1.Style.PatternColor property in the Properties window.

### Using Code

This example sets the pattern color and style.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow
Dim textBoxCell1 As New TextBoxCell()
textBoxCell1.Style.PatternColor = Color.LightBlue
textBoxCell1.Style.PatternStyle = MultiRowHatchStyle.DiagonalCross
GcMultiRow1.Template = Template.CreateGridTemplate(New Cell() { textBoxCell1 })
GcMultiRow1.RowCount = 10
GcMultiRow1.Rows(1).Cells(0).Style.PatternColor = Color.LightCoral
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;

TextBoxCell textBoxCell1 = new TextBoxCell();
textBoxCell1.Style.PatternColor = Color.LightBlue;
```
Background Gradation

You can set the background gradation in the cell style. The gradation supported in MultiRow is based upon the rendering feature of gradation used in the .NET Framework.

The following image shows the application of Color.Red and Color.Blue in the CellStyle.BackgroundGradientEffect.Colors property.

<table>
<thead>
<tr>
<th>CellStyle.BackgroundGradientEffect.Direction</th>
<th>CellStyle.BackgroundGradientEffect.Style</th>
<th>Application Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inherit</td>
<td>Horizontal</td>
<td><img src="image" alt="Horizontal" /></td>
</tr>
<tr>
<td>Inherit</td>
<td>Vertical</td>
<td><img src="image" alt="Vertical" /></td>
</tr>
<tr>
<td>Inherit</td>
<td>DiagonalUp</td>
<td><img src="image" alt="DiagonalUp" /></td>
</tr>
<tr>
<td>Inherit</td>
<td>DiagonalDown</td>
<td><img src="image" alt="DiagonalDown" /></td>
</tr>
<tr>
<td>Forward</td>
<td>CornerUp</td>
<td><img src="image" alt="CornerUp" /></td>
</tr>
<tr>
<td>Forward</td>
<td>CornerDown</td>
<td><img src="image" alt="CornerDown" /></td>
</tr>
<tr>
<td>Inherit</td>
<td>Rectangular</td>
<td><img src="image" alt="Rectangular" /></td>
</tr>
</tbody>
</table>

Using the Designer

1. Select the cells where gradation needs to be set (for example: textBoxCell1).
2. Select the textBoxCell1 Style.BackgroundGradientEffect property in the Properties window and click the ... (ellipsis) button.
3. On the GradientEffect editor, select Multiple Colors under the Color section.
4. Click the Add button and enter Green in Selected Color.
5. Select Add in the GradientEffect collection editor again, and enter Yellow in Selected Color.
6. Select Style in the bottom-left of the screen, and select Horizontal.
7. Select Direction in the bottom-center of the screen and select bottom-right (Side).
8. Select OK and close the GradientEffect Editor.

The following image displays the GradientEffect Editor.
Using Code

The following code creates a gradation.

[VB]

Imports GrapeCity.Win.MultiRow

Dim gradientEffect1 As New GradientEffect()
gradientEffect1.Colors = New Color() {Color.Green, Color.Yellow}
gradientEffect1.Style = GradientStyle.Horizontal
gradientEffect1.Direction = GradientDirection.Side

Dim textBoxCell1 As New TextBoxCell()
textBoxCell1.Style.BackgroundGradientEffect = gradientEffect1

GcMultiRow1.Template = Template.CreateGridTemplate(New Cell() {textBoxCell1})
GcMultiRow1.RowCount = 10

[CS]

using GrapeCity.Win.MultiRow;

GradientEffect gradientEffect1 = new GradientEffect();
gradientEffect1.Style = GradientStyle.Horizontal;
gradientEffect1.Direction = GradientDirection.Side;
Tag

You can save specific values inside the style tag. For example, you can save hidden data and read and write it at runtime. The tag can be set using theCellStyle.Tag (Tag Property in the on-line documentation) property. The Styles for each of the Cell Types can be used to determine if the cell type supports the style tag.

Using the Designer

1. Select the cells where you want to set the tag (for example: textBoxCell1).
2. Set the desired value in the textBoxCell1.Style.Tag in the Properties window.

The value set in the CellStyle.Tag (Tag Property in the on-line documentation) property is not displayed in the cell. This allows you to to save information that you do not want to display, or for extended data. Since the Tag property is of the type Object, it can save information of any type.

Using Code

The following code sets the upper limit on the number of characters in the Tag property, which is then used in the validation of values.

[VB]

Imports GrapeCity.Win.MultiRow

Private Sub Form1_Load( _
    ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles MyBase.Load
    Dim template As Template = GrapeCity.Win.MultiRow.Template.Default
    template.Row.Cells(0).Style.Tag = 5
    GcMultiRow1.Template = template
End Sub

Private Sub GcMultiRow1_CellValidating( _
    ByVal sender As System.Object, ByVal e As CellValidatingEventArgs ) Handles GcMultiRow1.CellValidating
    Dim gcMultiRow As GcMultiRow = TryCast(sender, GcMultiRow)
    Dim cellStyle As CellStyle = gcMultiRow(e.RowIndex, e.CellIndex).Style
    If cellStyle.Tag IsNot Nothing Then
        Dim maxLength As Integer = CType(cellStyle.Tag, Integer)
        If e.FormattedValue IsNot Nothing Then
            Dim value As String = CType(e.FormattedValue, String)
            If (value.Length > maxLength) Then
                gcMultiRow(e.RowIndex, e.CellIndex).ErrorText = String.Format("Number of characters exceeds maximum limit of characters :{0}.", maxLength)
            Else
                gcMultiRow(e.RowIndex, e.CellIndex).ErrorText = ""
            End If
        End If
    End If
End Sub

[CS]

using GrapeCity.Win.MultiRow;

private void Form1_Load(object sender, EventArgs e)
{
}
Cell and Tag Conditional Styles

Since the conditional cell style (`ConditionalCellStyle` ([`ConditionalCellStyle Class' in the on-line documentation](#))) and named cell style (`NamedCellStyle` ([`NamedCellStyle Class' in the on-line documentation](#))) do not have the `CellStyle.Tag` property, there is a limitation when using the Tag property of each cell when also using these styles. To use the Tag property of each cell while using these styles, the following workaround can be used. You can use arrays or collections to save values for each cell since the Tag property can save values of any type.

Using Code

This example sets the Tag property.

[VB]
```vbnet
GcMultiRow1.Rows(0).Tag = New String() { "Value of Cell1", "Value of Cell2" }
```

[CS]
```cs
gcMultiRow1.Rows[0].Tag = new string[] { "Value of Cell1", "Value of Cell2" };
```

Since the combined cell style (`CombinedCellStyle` ([`CombinedCellStyle Class' in the on-line documentation](#))) can hold other cell styles, you can use the normal style (`CellStyle` ([`CellStyle Class' in the on-line documentation](#))) and other cell styles on one cell. It is not very effective to use this method just for the Tag property, so the Row.Tag property is recommended.

Using Code

This example assigns a style.

[VB]
```vbnet
Imports GrapeCity.Win.MultiRow

' In case of directly using the conditional styles
Dim conditionalCellStyle1 As New ConditionalCellStyle()
GcMultiRow1.CurrentCell.Style = conditionalCellStyle1

' In case of using conditional styles together with combined cell styles
Dim cellStyle1 As New CellStyle()
   cellStyle1.Tag = "Desired Value"
Dim conditionalCellStyle1 As New ConditionalCellStyle()
Dim combinedCellStyle1 As New CombinedCellStyle()
   combinedCellStyle1.Items.Add(cellStyle1)
```
The image and its positioning in the cell are set using the styles. An image that has been set through cell styles, is displayed in front as compared to one displayed in an image cell. The styles can be used to set the image in cases where you want to display a common mark in other cell types as well. **ImageCell** (ImageCell Class' in the on-line documentation) is used when the image is displayed from data that is read from the data source, and other cases where the value is used as the image. To set an image using styles, you can use the CellStyle.Image (Image Property' in the on-line documentation) and the CellStyle.ImageAlign (ImageAlign Property' in the on-line documentation) properties. Whether the cell types support images using styles, can be determined for each of the Cell Types from their Styles.

### Images

Images can be set with the CellStyle.Image (Image Property' in the on-line documentation) property.

#### Using Code

This example sets the image.

**VB**

```vbnet
GcMultiRow1.Rows(0).Cells(0).Style.Image = New Bitmap("test.png")
```

**CS**

```csharp
gcMultiRow1.Rows[0].Cells[0].Style.Image = new Bitmap(@"test.png");
```

### Image Position

The image position can be set with the CellStyle.ImageAlign property (ImageAlign Property' in the on-line documentation).

#### Using Code

This example sets the image postion.

**VB**

```vbnet
Imports GrapeCity.Win.MultiRow
```
Margins and Padding

In cells, you can specify two sizes, one is the **Margin** depicting the space between cells, and the other is **Padding** depicting the space inside the cells. The following image shows a column in which buttonCell1 has a margin and padding of 10 pixels on all four sides. The blue line shows the margin and the red lines show the padding. The margin and padding have not been changed in buttonCell2 and buttonCell3.

![Image showing margins and padding](image)

### Margins

The margin can be set using the CellStyle.**Margin (‘Margin Property’ in the on-line documentation)** property. If you set the margins, you will be able to adjust the distance between the cells when using the snaplines in the designer. Margins cannot be used when the snaplines are disabled or at runtime.

### Padding

The padding can be set using the CellStyle.**Padding (‘Padding Property’ in the on-line documentation)** property. The padding settings can affect the display of the cell contents. The extent of the affect depends on how the cell type has been implemented. Cell borders and border color are not affected by padding.

### Using Code

This example sets the padding.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

Dim template As Template = New Template()
Dim buttonCell1 As ButtonCell = New ButtonCell()
buttonCell1.Name = "buttonCell1"
buttonCell1.Style.Padding = New Padding(20)
buttonCell1.Size = New Size(80, 60)
buttonCell1.Location = New Point(10, 10)

template.Width = 100
template.Row.Cells.Add(buttonCell1)
template.Row.Height = 80
GcMultiRow1.Template = template
GcMultiRow1.RowCount = 10
```
### Multiline and Word Wrap

The control supports multiline strings and word wrap.

**Using the Designer**

1. Select the cells that need to be displayed in multiline or that need wrapping (for example: `textBoxCell1`).
2. Select the **Value** property in the **Properties** window and input the desired string (for example: ABC).
3. Set `textBoxCell1.Style.Multiline` property (CellStyle.Multiline ('Multiline Property' in the on-line documentation) property) to True from the **Properties** window.
4. Set the `textBoxCell1.Style.WordWrap` property (CellStyle.WordWrap ('WordWrap Property' in the on-line documentation) property) to True in the **Properties** window.

You cannot input multiline strings into the designer. In order to allow the user to insert new lines at runtime, refer to `TextBoxCell`.

**Using Code**

When multiline is enabled, the strings in the cells are displayed as multiline text using the new line code.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

GcMultiRow1.Rows(0).Cells(0).Value = "GrapeCity" & vbCrLf & "PowerTools"
GcMultiRow1.Rows(0).Cells(0).Style.Multiline = MultiRowTriState.True
GcMultiRow1.Rows(0).Cells(0).VerticalResize(30)
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;

gcMultiRow1.Rows[0].Cells[0].Value = "GrapeCity\r\nP\nowerTools";
gcMultiRow1.Rows[0].Cells[0].Style.Multiline = MultiRowTriState.True;
```
When both multiline and word wrap are enabled, if the string length exceeds the cell width, the word wrapping occurs automatically.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

GcMultiRow1.Rows(0).Cells(0).Value = "GrapeCity PowerTools"
GcMultiRow1.Rows(0).Cells(0).Style.Multiline = MultiRowTriState.True
GcMultiRow1.Rows(0).Cells(0).Style.WordWrap = MultiRowTriState.True
GcMultiRow1.Rows(0).Cells(0).VerticalResize(30)
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;

gcMultiRow1.Rows[0].Cells[0].Value = "GrapeCity PowerTools";
gcMultiRow1.Rows[0].Cells[0].Style.Multiline = MultiRowTriState.True;
gcMultiRow1.Rows[0].Cells[0].Style.WordWrap = MultiRowTriState.True;
gcMultiRow1.Rows[0].Cells[0].VerticalResize(30);
```

When the **GDI+ Compatibility Mode** is enabled, there might be cases where blank or space characters (visible when wrapped), are not displayed.

Text Formatting

Cell styles support text formats based on the .NET Framework. The text format can be set using the CellStyle.**Format** ("Format Property' in the on-line documentation) property and the CellStyle.**FormatProvider** ("FormatProvider Property' in the on-line documentation) property. You can use the **Styles** of the **Cell Types** to determine whether the cell type supports the text format of that style. For details about usable formats and the support of data types, refer to Formatting Types [http://msdn.microsoft.com/en-us/library/fbxft59x.aspx](http://msdn.microsoft.com/en-us/library/fbxft59x.aspx).

**Text Content Formatting**

You can set the formatting to be applied when the text contents are displayed, by specifying the formatting string in the CellStyle.**Format** ("Format Property' in the on-line documentation) property.

**Using Code**

The following code shows the integer value in currency format using the present culture.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

Dim textBoxCell As TextBoxCell = TryCast(gcMultiRow1.Rows(0).Cells(0), TextBoxCell)
textBoxCell.ReadOnly = True
textBoxCell.Value = 20000
textBoxCell.Style.Format = "c"
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;

```
Culture Specific Formatting

You can specify the culture to be applied to the formatting using the CellStyle. `FormatProvider` ('FormatProvider Property' in the on-line documentation) property.

Using Code

The following code displays the value in the US currency format.

[VB]

```vbnet
Imports GrapeCity.Win.MultiRow
Imports System.Globalization

Dim textBoxCell As TextBoxCell = TryCast(gcMultiRow1.Rows(0).Cells(0), TextBoxCell)
textBoxCell.ReadOnly = True
textBoxCell.Value = 20000
textBoxCell.Style.Format = "\$"
textBoxCell.Style.FormatProvider = New CultureInfo("en-US")
```

[CS]

```csharp
using GrapeCity.Win.MultiRow;
using System.Globalization;

TextBoxCell textBoxCell = gcMultiRow1.Rows[0].Cells[0] as TextBoxCell;
textBoxCell.ReadOnly = true;
textBoxCell.Value = 20000;
textBoxCell.Style.Format = "\$";
textBoxCell.Style.FormatProvider = new CultureInfo("en-US");
```

Text Indent

You can show text in a hierarchical level display by using the indent. You can specify the indent in pixels using the CellStyle. `TextIndent` ('TextIndent Property' in the on-line documentation) property. The Styles in the Cell Types can be used to determine whether the specific cell type supports the indent.

Using the Designer

1. Select the cells for which the indent needs to be set (for example: textBoxCell1).
2. Input the desired string in the Value property in the Properties window (for example: ABC).

Using Code

This example sets the text indent.

[VB]

Imports GrapeCity.Win.MultiRow

GcMultiRow1.ViewMode = ViewMode.Display
GcMultiRow1.AllowUserToAddRows = False
GcMultiRow1.Template = Template.CreateGridTemplate(New Cell() {New TextBoxCell()})
GcMultiRow1.RowCount = 3
GcMultiRow1.Rows(0).Cells(0).Value = "Level1"
GcMultiRow1.Rows(0).Cells(0).Style.TextIndent = 0
GcMultiRow1.Rows(1).Cells(0).Value = "Level2"
GcMultiRow1.Rows(1).Cells(0).Style.TextIndent = 10
GcMultiRow1.Rows(2).Cells(0).Value = "Level3"
GcMultiRow1.Rows(2).Cells(0).Style.TextIndent = 20

[CS]

using GrapeCity.Win.MultiRow;

gcMultiRow1.ViewMode = ViewMode.Display;
gcMultiRow1.AllowUserToAddRows = false;
gcMultiRow1.Template = Template.CreateGridTemplate(new Cell[] {new TextBoxCell()});
gcMultiRow1.RowCount = 3;
gcMultiRow1.Rows[0].Cells[0].Value = "Level1";
gcMultiRow1.Rows[0].Cells[0].Style.TextIndent = 0;
gcMultiRow1.Rows[1].Cells[0].Value = "Level2";
gcMultiRow1.Rows[1].Cells[0].Style.TextIndent = 10;
gcMultiRow1.Rows[2].Cells[0].Value = "Level3";
gcMultiRow1.Rows[2].Cells[0].Style.TextIndent = 20;

GDI+ Compatibility Mode

Cells support the string rendering using both GDI as well as GDI+. GDI is used by default. GDI+ is used when the CellStyle.UseCompatibleTextRendering ("UseCompatibleTextRendering Property' in the on-line documentation) property is set to True. This property works the same as the Label and Button controls in .NET Framework 2.0 and above. The value, True, in the CellStyle.UseCompatibleTextRendering ("UseCompatibleTextRendering Property' in the on-line documentation) property implies GDI+ compatible mode. The Styles of each of the Cell Types can be used to determine whether the cell types support the GDI+ compatibility mode of the styles.

There are a few limitations due to the differences between GDI and GDI+, if the GDI+ compatibility mode is enabled. Refer to the Limitations section of this document for details.

Enable GDI+ Compatibility Mode

You need to set the CellStyle.UseCompatibleTextRendering ("UseCompatibleTextRendering Property' in the on-line documentation) property to True to enable the GDI+ compatibility mode.

Using Code
This example sets the `UseCompatibleTextRendering ('UseCompatibleTextRendering Property' in the online documentation)` property.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

GcMultiRow1.Rows(0).Cells(0).Style.UseCompatibleTextRendering = MultiRowTriState.True
GcMultiRow1.Rows(0).Cells(0).Value = "GDI+ Compatibility Mode"
```

**[CS]**

```cs
using GrapeCity.Win.MultiRow;

gcMultiRow1.Rows[0].Cells[0].Style.UseCompatibleTextRendering = MultiRowTriState.True;
gcMultiRow1.Rows[0].Cells[0].Value = "GDI+ Compatibility Mode";
```

### Distributing Text

You can use the distributed alignment of text when the GDI+ compatibility mode is enabled. To specify the space on the left and right for the distributed text, you can use either or both the padding and equal distribution. Unlike Excel, indent is not applied to the distributed alignment.

**Using Code**

This example uses the distributed alignment.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

GcMultiRow1.Rows(0).Cells(0).Style.UseCompatibleTextRendering = MultiRowTriState.True
GcMultiRow1.Rows(0).Cells(0).Style.TextAdjustment = TextAdjustment.Distribute
GcMultiRow1.Rows(0).Cells(0).Value = "Nebraska"
```

**[CS]**

```cs
using GrapeCity.Win.MultiRow;

gcMultiRow1.Rows[0].Cells[0].Style.UseCompatibleTextRendering = MultiRowTriState.True;
gcMultiRow1.Rows[0].Cells[0].Style.TextAdjustment = TextAdjustment.Distribute;
gcMultiRow1.Rows[0].Cells[0].Value = "Nebraska";
```

### Rotating Text

You can use the text rotation when the GDI+ compatibility mode is enabled.

**Using Code**

This example rotates the text.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow
```
Vertical Text

You can use the vertical text feature when the GDI+ compatibility mode is enabled.

Using Code

This example uses vertical text.

[VB]

Imports GrapeCity.Win.MultiRow

gcMultiRow1.Rows(0).Cells(0).Style.UseCompatibleTextRendering = MultiRowTriState.True
GCMultiRow1.Rows(0).Cells(0).Style.TextVertical = MultiRowTriState.True
GCMultiRow1.Rows(0).Cells(0).VerticalResize(50)
GCMultiRow1.Rows(0).Cells(0).Value = "Yellow Line"

[CS]

using GrapeCity.Win.MultiRow;

gcMultiRow1.Rows[0].Cells[0].Style.UseCompatibleTextRendering = MultiRowTriState.True;
gcMultiRow1.Rows[0].Cells[0].Style.TextVertical = MultiRowTriState.True;
gcMultiRow1.Rows[0].Cells[0].VerticalResize(50);
gcMultiRow1.Rows[0].Cells[0].Value = "Yellow Line";

Limitations

- When the GDI+ compatibility mode is enabled, there might be cases where the space between characters and cell position for cells in edit mode are different from the cells not in edit mode (Microsoft Support Document Online Number - 307208 Why text appears different when drawn with GDIPlus versus GDI http://support.microsoft.com/kb/307208/).
- When the GDI+ compatibility mode is enabled, and you specify Microsoft Sans Serif for the cell font, the single byte Yen character will be displayed as back slash when not in edit mode.
- When the GDI+ compatibility mode is enabled, and word wrap has been enabled, there might be cases where space characters are not displayed.
- When the GDI+ compatibility mode is enabled, there might be cases when the double byte hyphen is not displayed. This problem was fixed in November 2008 (Microsoft Support Document Online Number - 956807). If you use an application that uses GDI+ API functions on Windows Server 2008, Windows Vista, Windows Server 2003, or Windows XP, the Unicode hyphen character (U+2010) is not rendered (http://support.microsoft.com/kb/956807/).
- When the GDI+ compatibility mode is enabled, the tab characters (tab stop) are not displayed. This is a
limitation of MultiRow.

- When the GDI+ compatibility mode is enabled, the Unicode composite and control characters are not displayed correctly (for example: "\u0041\u0301\u304b\u3099\u304b\u309a", "\u202e Lighting\u202c").
- When the GDI+ compatibility mode is enabled, there are cases where the character width is incorrect when the same character is repeated a number of times (for example: "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA").

Mouse Hover Styles

You can set the style for the mouse hover in the cell style. The mouse hover style can set the backcolor, the background gradation, and the forecolor.

Setting the BackColor and ForeColor

You can set the backcolor and forecolor of the mouse hover style, using the CellStyle.MouseOverBackColor ("MouseOverBackColor Property' in the on-line documentation) property, and the CellStyle.MouseOverForeColor ("MouseOverForeColor Property' in the on-line documentation) property.

Using Code

This example sets the MouseOverBackColor and MouseOverForeColor properties.

[VB]
```
GcMultiRow1.Rows(0).Cells(0).Style.MouseOverBackColor = Color.Yellow
GcMultiRow1.Rows(0).Cells(0).Style.MouseOverForeColor = Color.Red
```

[CS]
```
gcMultiRow1.Rows[0].Cells[0].Style.MouseOverBackColor = Color.Yellow;
gcMultiRow1.Rows[0].Cells[0].Style.MouseOverForeColor = Color.Red;
```

Setting the Background Gradient

You can use the CellStyle.MouseOverGradientEffect ("MouseOverGradientEffect Property' in the on-line documentation) property to set gradation in the background color of the mouse hover.

Using Code

This example sets the gradation.

[VB]
```
Dim gradientEffect1 As New GradientEffect()
gradientEffect1.Colors = New Color() {Color.Green, Color.Yellow}
gradientEffect1.Style = GradientStyle.Horizontal
gradientEffect1.Direction = GradientDirection.Side
GcMultiRow1.Rows(0).Cells(0).Style.MouseOverGradientEffect = gradientEffect1
```

[CS]
```
GradientEffect gradientEffect1 = new GradientEffect();
gradientEffect1.Style = GradientStyle.Horizontal;
gradientEffect1.Direction = GradientDirection.Side;
gcMultiRow1.Rows[0].Cells[0].Style.MouseOverGradientEffect = gradientEffect1;
```
Settings for Individual Section
You can set the backcolor and background gradation of the mouse hover style for individual sections.

Using Code
This example sets the gradation.

[VB]
```
Dim gradientEffect1 As New GradientEffect()
gradientEffect1.Colors = New Color() {Color.Green, Color.Yellow}
gradientEffect1.Style = GradientStyle.Horizontal
gradientEffect1.Direction = GradientDirection.Side
GcMultiRow1.Rows(0).MouseOverGradientEffect = gradientEffect1
GcMultiRow1.Rows(1).MouseOverBackColor = Color.Azure
```

[CS]
```
GradientEffect gradientEffect1 = new GradientEffect();
gradientEffect1.Style = GradientStyle.Horizontal;
gradientEffect1.Direction = GradientDirection.Side;
gcMultiRow1.Rows[0].MouseOverGradientEffect = gradientEffect1;
```

User-defined Word Wrap
In MultiRow, you can customize the word wrapping position of the content in the cell, by recognizing a specified number of characters, or double byte numbers, as one word.

You can customize the behavior of word wrap, by creating a class that implements the ICustomWordWrapProvider interface.

Word Wrap by Number of Characters

Using Code
The following code creates a class that implements the ICustomWordWrapProvider interface, which specifies the word wrap position by number of characters.

[VB]
```
Imports GrapeCity.Win.MultiRow
Public Class MyWordWrap Implements ICustomWordWrapProvider
    Public Function GetWordUnits(text As String) As System.Collections.Generic.IEnumerable(Of WordWrapUnit) Implements ICustomWordWrapProvider.GetWordUnits
        Dim list As New List(Of WordWrapUnit)()
        If text = "abcde,fghij" Then
            list.Add(New WordWrapUnit(0, 3))
            list.Add(New WordWrapUnit(3, 8))
        End If
        Return list
    End Function
End Class
```

[CS]
```
using GrapeCity.Win.MultiRow;
public class MyWordWrap : ICustomWordWrapProvider
{
    public IEnumerable<WordWrapUnit> GetWordUnits(string text)
    {
        List<WordWrapUnit> list = new List<WordWrapUnit>();
        if (text == "abcde,fghij")
        {
            list.Add(new WordWrapUnit(0, 3));
            list.Add(new WordWrapUnit(3, 8));
        }
        return list;
    }
}
```
Set the CellStyle: CustomWordWrapProvider (CustomWordWrapProvider Property' in the on-line documentation) property to the class that implements the ICustomWordWrapProvider interface.

Using Code
You need to perform the following steps to enable this feature.
- Set the CellStyle: Multiline (Multiline Property' in the on-line documentation) property to True.
- Set the CellStyle: WordWrap (WordWrap Property' in the on-line documentation) property to True.
- Set the CellStyle: UseCompatibleTextRendering (UseCompatibleTextRendering Property' in the on-line documentation) property to True.

[VB]
```vbnet
GcMultiRow1(0, 0).Style.UseCompatibleTextRendering = MultiRowTriState.True
GcMultiRow1(0, 0).Style.WordWrap = MultiRowTriState.True
GcMultiRow1(0, 0).Style.Multiline = MultiRowTriState.True
GcMultiRow1(0, 0).Style.CustomWordWrapProvider = New MyWordWrap()
```

[CS]
```cs
gcMultiRow1[0, 0].Style.UseCompatibleTextRendering = MultiRowTriState.True;
gcMultiRow1[0, 0].Style.WordWrap = MultiRowTriState.True;
gcMultiRow1[0, 0].Style.Multiline = MultiRowTriState.True;
gcMultiRow1[0, 0].Style.CustomWordWrapProvider = new MyWordWrap();
```

Customizing the Justification of Single-byte Characters

By default, if you use justification in single-byte characters, the spaces are expanded, but if you use the ITextAdjustmentProvider (ITextAdjustmentProvider Interface' in the on-line documentation) interface, you can uniformly display single-byte characters in a cell.

Create a class that implements the ITextAdjustmentProvider (ITextAdjustmentProvider Interface' in the on-line documentation) interface, to uniformly display single-byte characters in a cell.

Using Code
This example creates a class to uniformly display single-byte characters in a cell.

[VB]
```vbnet
Imports GrapeCity.Win.MultiRow
Public Class DistributeEveryChar Implements ITextAdjustmentProvider
    Public Function GetTextElements(text As String) As System.Collections.Generic.IEnumerable(Of String) Implements ITextAdjustmentProvider.GetTextElements
        Dim textElements As New List(Of String)()
        For i As Integer = 0 To text.Length - 1
            textElements.Add(text(i).ToString())
        Next
        Return textElements
    End Function
End Class
```

[CS]
```cs
using GrapeCity.Win.MultiRow;
public class DistributeEveryChar : ITextAdjustmentProvider
{
    public IEnumerable< string > GetTextElements(string text)
    {
        List< string > textElements = new List< string >();
        for (int i = 0; i < text.Length; i++)
        {
            textElements.Add(text[i].ToString());
        }
        return textElements;
    }
}
```

Setting the Justification
Set the CellStyle: TextAdjustmentProvider (TextAdjustmentProvider Property' in the on-line documentation) property to the class that implements the ITextAdjustmentProvider interface (the class that was created for the customization of justification settings).

You need to perform the following steps to enable this feature.
- Set the CellStyle: TextAdjustment (TextAdjustment Property' in the on-line documentation) property to Distribute, DistributeWithSpace, or Justify.
- Set the CellStyle: UseCompatibleTextRendering (UseCompatibleTextRendering Property' in the on-line documentation) property to True.

Using Code
This example sets the justification using the created class.

[VB]
Conditional Cell Styles

You can change the cell styles based on specific conditions and code using conditional styles. For details on the rules for applying styles, and other options, refer to Cell Styles.

- Types of Cell Styles
- Manage Cell Styles with Names
- Changing Cell Styles based on Conditions
- Combining Multiple Styles
- User Defined Conditional Styles
- Manage Cell Styles with Names

Types of Cell Styles

There are five types of cell styles provided in MultiRow, and you can use them as needed.

Default Cell Style

This is the simplest style. It sets the style directly for each cell.

In the default cell style, the result of the style application remains the same. An exception would be where the style is inherited from the parent object such as a grid or the row.

This style is similar in concept to the standard DataGridView control style. This style can be used when the style does not need to change based on certain conditions, or the frequency of reusing the style is low.

Conditional Cell Style

Cell values and built-in conditions are used to determine whether a style needs to be applied to the cells. The conditional style is given priority at design time compared to the dynamic style. Dynamic styles are used to defined complicated conditions.
The styles applied to cells vary depending on certain conditions with conditional cell styles.

For details on conditional cell styles, refer to **Changing Cell Styles based on Conditions**.

**Named Cell Styles**

You can assign a name to the cell style and apply the style to the cells using that name. You can reuse the same design and you can change the style which makes it easier to manage, similar to using skins and themes. You can use other cell styles to specify the style information. You can apply named cell styles to multiple cells.

All cells to which the named cell style has been applied are affected when you change the style.

For details on named cell styles, refer to **Manage Cell Styles with Names**.

**Combined Cell Styles**

Combined cell styles are extendible cell styles that hold other styles. For example, you can apply conditional styles, named cell styles, and dynamic cell styles (explained later) to a single cell.
In the combined cell style, the added cell style overwrites the existing cell style. If there is an option in the added cell style whose value has not been set, it is inherited from the existing cell style.

For details on combined cell styles, refer to **Combining Multiple Styles**.

**Dynamic Cell Styles**

Dynamic cell styles are applied based on user defined conditions. Since the conditions are specified by the developer through code, the conditions can be set through various procedures. This cell style is not available at design time since this cell style uses settings from code. For details on dynamic cell styles, refer to **User Defined Conditional Styles**.

**Manage Cell Styles with Names**

You can manage styles with names if you use named cell styles (**NamedCellStyle ('NamedCellStyle Class' in the on-line documentation) class**). These styles are convenient when you want to develop styles based on sharing, reusing, or using them interchangeably like skins. In the MultiRow designer, you are supplied with the **Template - Named Cell Styles 7.0** in order to be able to use and manage the named cell styles. Named cell styles can be used in both templates and GcMultiRow controls.

**Register Named Cell Styles**

Use the Template.**NamedCellStyles ('NamedCellStyles Property' in the on-line documentation) property** to register the named cell styles in the template.

**Using Code**

The following code adds a cell style "Cool Color" with blue as the backcolor.

```vbnet
Imports GrapeCity.Win.MultiRow

' Create cell style that has blue as the backcolor
Dim blueCellStyle As New CellStyle()
blueCellStyle.BackColor = Color.Blue

Dim template1 As Template = template.Default
```
The named cell style registered in the template is automatically enabled in the GcMultiRow control, when the template is set into the GcMultiRow control. You can change the named cell style in the GcMultiRow control template.

**Using Code**

The following code replaces the cell style that has been registered with the name "Cool color" to a cell style which is light blue.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

Dim blueCellStyle As New CellStyle()
blueCellStyle.BackColor = Color.Blue

Dim template1 As Template = Template.Default
template1.NamedCellStyles.Add("Cool color", blueCellStyle)

GcMultiRow1.Template = template1
GcMultiRow1.Rows(0).Cells(0).Style = New NamedCellStyle("Cool color")

Dim lightblueCellStyle As New CellStyle()
lightblueCellStyle.BackColor = Color.LightBlue

GcMultiRow1.NamedCellStyles.Remove("Cool color")
GcMultiRow1.NamedCellStyles.Add("LightBlueColor", lightblueCellStyle)
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;

CellStyle blueCellStyle = new CellStyle();
blueCellStyle.BackColor = Color.Blue;

Template template1 = Template.Default;

// Register the cell style with a name "Cool color"
template1.NamedCellStyles.Add("Cool color", blueCellStyle);

GcMultiRow1.Template = template1;
GcMultiRow1.Rows(0).Cells(0).Style = New NamedCellStyle("Cool color");

GcMultiRow1.Rows[0].Cells[0].Style = new NamedCellStyle("Cool color");

GcMultiRow1.NamedCellStyles.Remove("Cool color");
GcMultiRow1.NamedCellStyles.Add("LightBlueColor", lightblueCellStyle);
```
template1.NamedCellStyles.Add("Cool color", blueCellStyle);

gcMultiRow1.Template = template1;
gcMultiRow1.Rows[0].Cells[0].Style = new NamedCellStyle("Cool color");

CellStyle lightblueCellStyle = new CellStyle();
lightblueCellStyle.BackColor = Color.LightBlue;

gcMultiRow1.NamedCellStyles.Remove("Cool color");
gcMultiRow1.NamedCellStyles.Add("LightBlueColor", lightblueCellStyle);

Using Named Cell Styles

To set a named cell style in a specific cell, you need to set the NamedCellStyle object in the Cell. Style (‘Style Property’ in the on-line documentation) property.

Using Code

This example uses a NamedCellStyle object.

[VB]

Imports GrapeCity.Win.MultiRow

Dim blueCellStyle As New CellStyle()
blueCellStyle.BackColor = Color.Blue

Dim template1 As Template = template.Default
template1.NamedCellStyles.Add("Cool color", blueCellStyle)
't template.Row.Cells(0).Style = New NamedCellStyle("Cool color")

gcMultiRow1.Template = template1
GcMultiRow1.Rows(0).Cells(0).Style = New NamedCellStyle("Cool color")

[CS]

using GrapeCity.Win.MultiRow;

CellStyle blueCellStyle = new CellStyle();
blueCellStyle.BackColor = Color.Blue;

Template template1 = Template.Default;
template1.NamedCellStyles.Add("Cool color", blueCellStyle);
// template1.Row.Cells[0].Style = new NamedCellStyle("Cool color");

gcMultiRow1.Template = template1;
gcMultiRow1.Rows[0].Cells[0].Style = new NamedCellStyle("Cool color");

The order of registering the named cell style and assigning it to a cell does not make a difference.

Using Code

The following code registers the named cell style after applying it to a cell.

[VB]

Imports GrapeCity.Win.MultiRow

Dim blueCellStyle As New CellStyle()
blueCellStyle.BackColor = Color.Blue

Dim template1 As Template = template.Default
template1.NamedCellStyles.Add("Cool color", blueCellStyle)
't template1.Row.Cells[0].Style = New NamedCellStyle("Cool color")

gcMultiRow1.Template = template1
GcMultiRow1.Rows[0].Cells[0].Style = New NamedCellStyle("Cool color");
Dim template1 As Template = template.Default
GcMultiRow1.Template = template1
GcMultiRow1.Rows(0).Cells(0).Style = New NamedCellStyle("Cool color")

Dim blueCellStyle As New CellStyle()
blueCellStyle.BackColor = Color.Blue
GcMultiRow1.NamedCellStyles.Add("Cool color", blueCellStyle)

[CS]

using GrapeCity.Win.MultiRow;

Template template1 = Template.Default;
gcMultiRow1.Template = template1;

gcMultiRow1.Rows[0].Cells[0].Style = new NamedCellStyle("Cool color");

CellStyle blueCellStyle = new CellStyle();
blueCellStyle.BackColor = Color.Blue;
gcMultiRow1.NamedCellStyles.Add("Cool color", blueCellStyle);

Check Whether the Named Cell Style is Registered

The named cell style is unique in a template or in a control. An ArgumentException occurs when you try to add an already existing named cell style to the Template.NamedCellStyles ("NamedCellStyles Property" in the on-line documentation) property collection. Use the NamedCellStyleDictionary.ContainsKey ("ContainsKey Method" in the on-line documentation) method to confirm whether the named cell style is already registered.

Using Code

This example uses the ContainsKey method.

[VB]

If template1.NamedCellStyles.ContainsKey("Cool color") Then
    Console.WriteLine("Already registered")
End If

[CS]

if (template1.NamedCellStyles.ContainsKey("Cool color"))
{
    Console.WriteLine("Already registered");
}

Delete Named Cell Styles

You can delete the named cell style with the NamedCellStyleDictionary.Remove ("Remove Method" in the on-line documentation) method. The default style will be used in cells where that named cell style was applied. Use the combined cell style to change this behavior explicitly.

Changing Cell Styles based on Conditions

You can apply styles to cells when the cells satisfy a set of pre-defined conditions, with conditional cell styles (ConditionalCellStyle
List of Conditions

The built-in conditions, which you can specify in the ConditionalCellStyleItem ("ConditionalCellStyleItem Class' in the on-line documentation) class, are as follows:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Value 1</th>
<th>Value 2</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsNull</td>
<td>-</td>
<td>-</td>
<td>Cell value is null reference (Nothing in Visual Basic) or DBNull.Value</td>
</tr>
<tr>
<td>IsNotNull</td>
<td>-</td>
<td>-</td>
<td>Cell value is not null reference (Nothing in Visual Basic) or DBNull.Value</td>
</tr>
<tr>
<td>IsEmpty</td>
<td>-</td>
<td>-</td>
<td>Cell value is null reference (Nothing in Visual Basic), DBNull.Value, or String.Empty</td>
</tr>
<tr>
<td>IsNotEmpty</td>
<td>-</td>
<td>-</td>
<td>Cell value is not empty (Nothing in Visual Basic), DBNull.Value, or String.Empty</td>
</tr>
<tr>
<td>IsTrue</td>
<td>-</td>
<td>-</td>
<td>Cell value is True</td>
</tr>
<tr>
<td>IsFalse</td>
<td>-</td>
<td>-</td>
<td>Cell value is False</td>
</tr>
<tr>
<td>Between Valid Value 1 and Value 2.</td>
<td>-</td>
<td>-</td>
<td>Cell value is included in the range defined by Value 1 and Value 2</td>
</tr>
<tr>
<td>NotBetween Valid Value 1 and Value 2.</td>
<td>-</td>
<td>-</td>
<td>Cell value is not included in the range defined by Value 1 and Value 2</td>
</tr>
<tr>
<td>Equals Valid Value 1.</td>
<td>Valid</td>
<td>-</td>
<td>Cell value is the same as that of Value 1</td>
</tr>
<tr>
<td>NotEquals Valid Value 1.</td>
<td>Valid</td>
<td>-</td>
<td>Cell value is not the same as that of Value 1</td>
</tr>
<tr>
<td>GreaterThan Valid Value 1.</td>
<td>Valid</td>
<td>-</td>
<td>Cell value is greater than Value 1</td>
</tr>
<tr>
<td>LessThan Valid Value 1.</td>
<td>Valid</td>
<td>-</td>
<td>Cell value is smaller than Value 1</td>
</tr>
<tr>
<td>GreaterThanOrEquals Valid Value 1.</td>
<td>Valid</td>
<td>-</td>
<td>Cell value is greater than or equal to Value 1</td>
</tr>
<tr>
<td>LessThanOrEquals Valid Value 1.</td>
<td>Valid</td>
<td>-</td>
<td>Cell value is smaller than or equal to Value 1</td>
</tr>
<tr>
<td>Contains Valid Value 1 (String comparison).</td>
<td>Valid</td>
<td>-</td>
<td>Cell value contains Value 1 (String comparison)</td>
</tr>
<tr>
<td>StartsWith Valid Value 1 (String comparison).</td>
<td>Valid</td>
<td>-</td>
<td>Cell value starts with Value 1 (String comparison)</td>
</tr>
<tr>
<td>EndsWith Valid Value 1 (String comparison).</td>
<td>Valid</td>
<td>-</td>
<td>Cell value ends with Value 1 (String comparison)</td>
</tr>
<tr>
<td>IsMatch Valid Value 1.</td>
<td>Valid</td>
<td>-</td>
<td>Cell value matches the Regular Expression of Value 1</td>
</tr>
<tr>
<td>IsNotMatch Valid Value 1.</td>
<td>Valid</td>
<td>-</td>
<td>Cell value does not match the Regular Expression of Value 1</td>
</tr>
</tbody>
</table>

Developers can use dynamic cell styles in order to define independent conditions. For details, refer to User Defined Conditional Styles.

Using Code

The following code sets the cell color to red when the cell value is null (Nothing in Visual Basic) and blue when it is not null.

[VB]

```vbnet
Imports GrapeCity.Win.MultiRow

Dim conditionalCellStyle1 As New ConditionalCellStyle() ' Style when cell value is null
Dim cellStyle1 As New CellStyle()
cellStyle1.BackColor = Color.FromArgb(255, 192, 192)
Dim item1 As New ConditionalCellStyleItem(cellStyle1, ConditionalCellStyleOperator.IsNull)
conditionalCellStyle1.Items.Add(item1)

' Style when the cell value is not null
Dim cellStyle2 As New CellStyle()
cellStyle2.BackColor = Color.FromArgb(255, 192, 192)
Dim item2 As New ConditionalCellStyleItem(cellStyle2, ConditionalCellStyleOperator.IsNotNull)
conditionalCellStyle1.Items.Add(item2)
```

[CS]

```cs
```
using GrapeCity.Win.MultiRow;

ConditionalCellStyle conditionalCellStyle1 = newConditionalCellStyle();

// Style when cell value is null
CellStyle cellStyle1 = newCellStyle();
cellStyle1.BackColor = Color.FromArgb(255, 192, 192);
ConditionalCellStyleItem item1 = newConditionalCellStyleItem(cellStyle1, ConditionalCellStyleOperator.IsNull);
conditionalCellStyle1.Items.Add(item1);

// Style when the cell value is not null
CellStyle cellStyle2 = newCellStyle();
cellStyle2.BackColor = Color.FromArgb(192, 192, 255);
ConditionalCellStyleItem item2 = newConditionalCellStyleItem(cellStyle2, ConditionalCellStyleOperator.IsNotNull);
conditionalCellStyle1.Items.Add(item2);

The following code applies the above conditional cell style to the cells.

[VB]

Dim template1 As Template = Template.Default
template1.Row.Cells(1).Style = conditionalCellStyle1
GcMultiRow1.Template = template1
GcMultiRow1.RowCount = 3

[CS]

Template template1 = Template.Default;
template1.Row.Cells[1].Style = conditionalCellStyle1;
gcMultiRow1.Template = template1;
gcMultiRow1.RowCount = 3;

Since the cell does not have a value in the default state, the backcolor of the cell is set to red. When a value is entered in the cell, the backcolor of the cell becomes blue.

Combining Multiple Styles

Combined cell styles (CombinedCellStyle (CombinedCellStyle Class' in the on-line documentation) class) are applied after combining multiple cell styles. You can combine and use all four styles - CellStyle (CellStyle Class' in the on-line documentation), ConditionCellStyle (ConditionCellStyle Class' in the on-line documentation), NamedCellStyle (NamedCellStyle Class' in the on-line documentation), and DynamicCellStyle (DynamicCellStyle Class' in the on-line documentation).

Using Code

The following code changes the BackColor and ForeColor properties in the normal and conditional cell styles, and then consolidates them using combined cell styles. The style is then applied to a cell.

[VB]

Imports GrapeCity.Win.MultiRow

' Create a cell style with back color as Blue
Dim blueCellStyle As NewCellStyle()
blueCellStyle.BackColor = Color.Azure
Dim template1 As Template = Template.Default
' Register the cell style with a name "Cool Color"
template1.NamedCellStyle.Add("Cool Color", blueCellStyle)

Dim conditionalCellStyle1 As New ConditionalCellStyle()
' Cell style when the value is null
Dim cellStyle1 As NewCellStyle()
cellStyle1.BackColor = Color.Black
User Defined Conditional Styles

You can apply cell styles based on user defined conditions if you use dynamic cell styles (DynamicCellStyle ('DynamicCellStyle Class' in the on-line documentation) class). Since the conditions for dynamic cell styles are set with code, they are not available in the designer.

Using Code

The following code changes the cell backcolor for alternating rows.

[VB]

Imports GrapeCity.Win.MultiRow

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    Dim dynamicCellStyle1 As New DynamicCellStyle()
    dynamicCellStyle1.ConditionHandler = New DynamicCellStyleConditionHandler(AddressOf MyCondition)
    Dim template1 As Template = Template.Default
    template1.Row.DefaultCellStyle = dynamicCellStyle1
    GcMultiRow1.Template = template1
    GcMultiRow1.RowCount = 3
End Sub

[CS]

using GrapeCity.Win.MultiRow;

// Create cell style with Blue as the back color
CellStyle blueCellStyle = new CellStyle();
blueCellStyle.BackColor = Color.Azure;
Template template1 = Template.Default;
// Register the cell style with a name "Cool Color"
template1.NamedCellStyle.Add("Cool Color", blueCellStyle);

ConditionalCellStyle conditionalCellStyle1 = new ConditionalCellStyle();
// Cell style when the value is null
CellStyle cellStyle1 = new CellStyle();
cellStyle1.ForeColor = Color.Black;
ConditionalCellStyleItem item1 = new ConditionalCellStyleItem(cellStyle1, ConditionalCellStyleOperator.IsNull);
conditionalCellStyle1.Items.Add(item1);
// Cell style when the value is not null
CellStyle cellStyle2 = new CellStyle();
cellStyle2.ForeColor = Color.Red;
ConditionalCellStyleItem item2 = new ConditionalCellStyleItem(cellStyle2, ConditionalCellStyleOperator.IsNotNull);
conditionalCellStyle1.Items.Add(item2);
// Create a combined cell style
CombinedCellStyle combinedCellStyle1 = new CombinedCellStyle();
combinedCellStyle1.Items.Add(new NamedCellStyle("Cool Color"));
combinedCellStyle1.Items.Add(conditionalCellStyle1);
GcMultiRow1.Template = template1;
GcMultiRow1.Rows[0].Cells[0].Style = combinedCellStyle1;
Private Function MyCondition(ByVal context As DynamicCellStyleContext) As CellStyle
    Dim cellStyle1 As New CellStyle
    ' Change the Back Color of alternating row
    If context.CellScope = CellScope.Row AndAlso (context.RowIndex Mod 2) = 0 Then
        cellStyle1.BackColor = Color.LightCyan
    Else
        cellStyle1.BackColor = Color.LightSalmon
    End If
    Return cellStyle1
End Function

using GrapeCity.Win.MultiRow;

private void Form1_Load(object sender, EventArgs e)
{
    DynamicCellStyle dynamicCellStyle1 = new DynamicCellStyle();
    dynamicCellStyle1.ConditionHandler = new DynamicCellStyleConditionHandler(MyCondition);

    Template template1 = Template.Default;
    template1.Row.DefaultCellStyle = dynamicCellStyle1;
    gcMultiRow1.Template = template1;
    gcMultiRow1.RowCount = 3;
}

private CellStyle MyCondition(DynamicCellStyleContext context)
{
    CellStyle cellStyle1 = new CellStyle();
    // Change the Back Color of alternating row
    if (context.CellScope == CellScope.Row AndAlso (context.RowIndex % 2) == 0)
    {
        cellStyle1.BackColor = Color.LightCyan;
    }
    else
    {
        cellStyle1.BackColor = Color.LightSalmon;
    }
    return cellStyle1;
}

The following image shows the result of the code at runtime:

![Image showing the result of the code runtime]

Manage Cell Styles with Names

You can manage styles with names if you use named cell styles (NamedCellStyle ('NamedCellStyle Class' in the on-line documentation) class). These styles are convenient when you want to develop styles based on sharing, re-using, or using them interchangeably like skins. In the MultiRow designer, you are supplied with the Template - Named Cell Styles 7.0 in order to be able to use and manage the named cell styles. Named cell styles can be used in
both templates and GcMultiRow controls.

### Register Named Cell Styles

Use the Template.**NamedCellStyles** (**NamedCellStyles Property** in the on-line documentation) property to register the named cell styles in the template.

#### Using Code

The following code adds a cell style "Cool Color" with blue as the backcolor.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

' Create cell style that has blue as the backcolor
Dim blueCellStyle As New CellStyle()
blueCellStyle.BackColor = Color.Blue

Dim template1 As Template = template.Default
' Register the cell style with a name "Cool color"
template1.NamedCellStyles.Add("Cool color", blueCellStyle)

GcMultiRow1.Template = template1
GcMultiRow1.Rows(0).Cells(0).Style = New NamedCellStyle("Cool color")
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;

// Create cell style that has blue as the backcolor
CellStyle blueCellStyle = new CellStyle();
blueCellStyle.BackColor = Color.Blue;

Template template1 = Template.Default;
// Register the cell style with a name "Cool color"
template1.NamedCellStyles.Add("Cool color", blueCellStyle);

gcMultiRow1.Template = template1;
gcMultiRow1.Rows[0].Cells[0].Style = new NamedCellStyle("Cool color");
```

The named cell style registered in the template is automatically enabled in the GcMultiRow control, when the template is set into the GcMultiRow control. You can change the named cell style in the GcMultiRow control template.

#### Using Code

The following code replaces the cell style that has been registered with the name "Cool color" to a cell style which is light blue.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

Dim blueCellStyle As New CellStyle()
blueCellStyle.BackColor = Color.Blue

Dim template1 As Template = Template.Default
```

Copyright © GrapeCity, inc. All rights reserved.
Using Named Cell Styles

To set a named cell style in a specific cell, you need to set the NamedCellStyle object in the Cell.Style (Style Property' in the on-line documentation) property.

Using Code

This example uses a NamedCellStyle object.

[VB]

Imports GrapeCity.Win.MultiRow

Dim blueCellStyle As New CellStyle()
blueCellStyle.BackColor = Color.Blue

Dim template1 As Template = template.Default
template1.NamedCellStyles.Add("Cool color", blueCellStyle)
' template.Row.Cells(0).Style = New NamedCellStyle("Cool color")

GcMultiRow1.Template = template1
GcMultiRow1.Rows(0).Cells(0).Style = New NamedCellStyle("Cool color")

[CS]

using GrapeCity.Win.MultiRow;

CellStyle blueCellStyle = new CellStyle();
blueCellStyle.BackColor = Color.Blue;

Template template1 = Template.Default;
template1.NamedCellStyles.Add("Cool color", blueCellStyle);

GcMultiRow1.Template = template1;
gcMultiRow1.Rows[0].Cells[0].Style = new NamedCellStyle("Cool color");

CellStyle lightblueCellStyle = new CellStyle();
lightblueCellStyle.BackColor = Color.LightBlue;

GcMultiRow1.NamedCellStyles.Remove("Cool color");
gcMultiRow1.NamedCellStyles.Add("LightBlueColor", lightblueCellStyle);
CellStyle blueCellStyle = new CellStyle();
blueCellStyle.BackColor = Color.Blue;

Template template1 = Template.Default;
template1.NamedCellStyles.Add("Cool color", blueCellStyle);
// template1.Row.Cells[0].Style = new NamedCellStyle("Cool color");

gcMultiRow1.Template = template1;
gcMultiRow1.Rows[0].Cells[0].Style = new NamedCellStyle("Cool color");

The order of registering the named cell style and assigning it to a cell does not make a difference.

Using Code
The following code registers the named cell style after applying it to a cell.

[VB]
Imports GrapeCity.Win.MultiRow

Dim template1 As Template = template.Default
GcMultiRow1.Template = template1

GcMultiRow1.Rows(0).Cells(0).Style = New NamedCellStyle("Cool color")

Dim blueCellStyle As New CellStyle()
blueCellStyle.BackColor = Color.Blue
GcMultiRow1.NamedCellStyles.Add("Cool color", blueCellStyle)

[CS]
using GrapeCity.Win.MultiRow;

Template template1 = Template.Default;
gcMultiRow1.Template = template1;

gcMultiRow1.Rows[0].Cells[0].Style = new NamedCellStyle("Cool color");

CellStyle blueCellStyle = new CellStyle();
blueCellStyle.BackColor = Color.Blue;
gcMultiRow1.NamedCellStyles.Add("Cool color", blueCellStyle);

Check Whether the Named Cell Style is Registered
The named cell style is unique in a template or in a control. An ArgumentException occurs when you try to add an already existing named cell style to the Template.NamedCellStyles ('NamedCellStyles Property' in the on-line documentation) property collection. Use the NamedCellStyleDictionary.ContainsKey ('ContainsKey Method' in the on-line documentation) method to confirm whether the named cell style is already registered.

Using Code
This example uses the ContainsKey method.

[VB]
If template1.NamedCellStyles.ContainsKey("Cool color") Then
Console.WriteLine("Already registered")
End If

[CS]

if (template1.NamedCellStyles.ContainsKey("Cool color"))
{
    Console.WriteLine("Already registered");
}

Delete Named Cell Styles

You can delete the named cell style with the NamedCellStyleDictionary. Remove (Remove Method in the on-line documentation) method. The default style will be used in cells where that named cell style was applied. Use the combined cell style to change this behavior explicitly.

User Input Validation

The following topics describe how to validate user input.

- Validation using Control Events
- Value Validation by Cell
- Validation
- Built-in Cell Validators
- Built-in Validation Actions
- Customizing the Validation Information
- User-Defined Cell Validator
- User Defined Validation Actions

Validation using Control Events

Values entered into the cell can be validated by two types of events, similar to the validation performed for the input into the control. Similarly, you can also perform validation for each row.

Notify Validation Error Using Icon and Tooltip

If the value entered by the user in a cell contains an error, you can display an error icon in that cell or in its associated row. Also, the user can confirm error details by placing the pointer over the error icon.

For instructions, refer to Displaying Error Messages.

Notify Validation Error Using Balloon Tip

If the value entered by the user contains an error, you can display validation error details in a balloon tip, which can prompt the user for an appropriate value. This validation indicator can be used if you are using the always edit mode (GcMultiRow.EditType=EditType.AlwaysEdit), that is, when the error icon or error tip cannot be displayed.

Using Code

The following code notifies the user of a validation error using a balloon tip when more than five characters are entered in the first textbox cell, and asks the user to change the value. To run this sample code, you need to keep the ToolTip component (ToolTip1) on the form.

[VB]

Imports GrapeCity.Win.MultiRow

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    GcMultiRow1.ShortcutKeyManager.Unregister(Keys.Enter)
    GcMultiRow1.ShortcutKeyManager.Register(SelectionActions.MoveToNextCell, Keys.Enter)
End Sub
    Dim gcMultiRow As GcMultiRow = DirectCast(sender, GcMultiRow)
    If (e.CellIndex = 0) AndAlso (e.RowIndex = 0) Then
        Dim currentCellRect As Rectangle = gcMultiRow.GetCellDisplayRectangle(e.RowIndex, e.CellIndex)
        Dim value As String = TryCast(e.FormattedValue, String)
        Dim hasError As Boolean = False
        If (Not value Is Nothing) AndAlso (value.Length > 5) Then
            Me.ToolTip1.ToolTipIcon = ToolTipIcon.Error
            Me.ToolTip1.ToolTipTitle = "Validation Error"
            Me.ToolTip1.IsBalloon = True
            Me.ToolTip1.SetToolTip(gcMultiRow.EditingControl, "dummy")
            Me.ToolTip1.Show("The value entered has more than 5 characters.", gcMultiRow.EditingControl, 0, currentCellRect.Height)
            hasError = True
            e.Cancel = True
        End If
        ' Remove balloon tip if there is no error.
        If (hasError = False) Then
            Me.ToolTip1.Hide(Me)
            Me.ToolTip1.RemoveAll()
        End If
    End If
End Sub

Modify Value
You can use the following to modify the value entered by the user depending on the timing.

When over-writing the value of a cell whose editing has been completed
GcMultiRow.SetValue('SetValue Method' in the on-line documentation) method or Cell.Value('Value Property' in the on-line documentation) property
The editing state of a cell can be confirmed using the GcMultiRow.IsEditing property.

Restrict Input of Value
The Cell Edit Control allows you to restrict the input value. For example, TextBoxCell ('TextBoxCell Class' in the on-line documentation) has a restriction of maximum length, NumericUpDownCell ('NumericUpDownCell Class' in the on-line documentation) takes numeric values only, and MaskedTextBoxCell ('MaskedTextBoxClass' in the on-line documentation) restricts formatted strings.

Value and DataSource
A GcMultiRow.DataError('DataError Event' in the on-line documentation) event occurs if the entered value does not match the data type of the datasource.

Value Validation by Cell
You can specify a validation object in the cell and then validate the cell’s value using the validation rule in that object. This procedure is applicable when defining code-less validation rules at design time and can be applied when the value of each cell is highly independent. Value validation using a cell is done by adding validation rules in the Cell.Validators('Validators Property' in the on-line documentation) property. In validation rules, you can also specify an individual validation action to notify the user of the validation result.

Validation
You can use the Cell.Validate ('Validate Method' in the on-line documentation) method to validate a cell's value. The Cell.Validate method returns False if there is a validation error in even one of the Cell.Validators. In addition, if you set the argument (doAction) of the Cell.Validate method to True, the Cell.Validators action is executed when the Cell.Validate method is called.


Using Code

The following code uses the Cell.Validate method to validate all the cells.

[VB]
Imports GrapeCity.Win.MultiRow
For Each r As Row In GcMultiRow1.Rows
    For Each c As Cell In r.Cells
        c.Validate(True)
    Next
Next

[CS]
using GrapeCity.Win.MultiRow;
foreach (Row r in gcMultiRow1.Rows)
{
    foreach (Cell c in r.Cells)
    {
        c.Validate(true);
    }
}

Validating the Value being Edited

If you set the argument of the Cell.Validate method to True, in the GcMultiRow.CellEditedFormattedValueChanged ('CellEditedFormattedValueChanged Event' in the on-line documentation) event, you can validate the value being edited.

Using Code

This example validates the value being edited.

[VB]
Imports GrapeCity.Win.MultiRow
Private Sub GcMultiRow1_CellEditedFormattedValueChanged(sender As Object, e As CellEditedFormattedValueChangedEventArgs) Handles GcMultiRow1.CellEditedFormattedValueChanged
    Dim gcMultiRow As GcMultiRow = TryCast(sender, GcMultiRow)
    Dim currentCell As Cell = gcMultiRow.Rows(e.RowIndex).Cells(e.CellIndex)
    currentCell.Validate(True)
End Sub

[CS]
using GrapeCity.Win.MultiRow;
private void gcMultiRow1_CellEditedFormattedValueChanged(object sender, CellEditedFormattedValueChangedEventArgs e)
{
    GcMultiRow gcMultiRow = sender as GcMultiRow;
    Cell currentCell = gcMultiRow.Columns[e.RowIndex].Cells[e.CellIndex];
    currentCell.Validate(true);
}

Built-in Cell Validators

MultiRow provides ten built-in validators. Each validator has pre-defined validation rules and you can use and customize whichever one you need.

- **RangeValidator**
- **IncludeListValidator**
- **RequiredTypeValidator**
- **RequiredFieldValidator**
- **ExcludeListValidator**
- **CompareCellValidator**
- **CompareValueValidator**
- **PairCharValidator**
- **SurrogateCharValidator**
- **RegularExpressionValidator**
- **CompareStringValidator**
- **TextLengthValidator**
- **EncodingValidator**

**RangeValidator**

You can use RangeValidator ('RangeValidator Class' in the on-line documentation) to validate if a cell's value
is within the specified range or not. A validation error occurs if the value does not fall inside the specified range.

**Using Designer**

Complete the following steps to set RangeValidator. The example sets up a `NumericUpDownCell` (*NumericUpDownCell Class* in the on-line documentation) for which a validation error occurs if the value lies outside the range of 0 to 10.

1. Select a cell for which to validate the value (for example, `numericUpDownCell1`).
2. From the **Properties** window, select the **Validators** property and click the **...** button.
3. From the displayed CellValidator collection editor, select **RangeValidator** from the top-left combo box and click the **Add** button.
4. From the **Members** list, confirm that **RangeValidator** has been selected.
5. Select the **MaxValue** property from the property grid and type **10**.
6. Select the **MinValue** property from the property grid and type **0**.
7. Select the **Actions** property from the property grid and click the **...** button.
8. Add **LineNotify** in the displayed CellValidateAction collection editor.
9. Click the **OK** button and close the CellValidateAction collection editor.
10. Click the **OK** button and close the CellValidator collection editor.
11. Change the document window tab of the designer to **Runtime**.
12. Type **12**, move the focus from the cell, and confirm the occurrence of the validation error.
13. Remove **12** and type **9**, move focus from the cell, and confirm that no error occurs.

**Using Code**

The following code displays a validation error when a number outside the range of 0 to 10 (such as -1 or 50) has been entered.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

Dim numericUpDownCell1 As New NumericUpDownCell()
Dim rangeValidator1 As New RangeValidator()
rangeValidator1.RequiredType = GetType(Integer)
rangeValidator1.MaxValue = 10
rangeValidator1.MinValue = 0
rangeValidator1.Actions.Add(New LineNotify())
numericUpDownCell1.Validators.Add(rangeValidator1)

Dim cells As Cell() = {numericUpDownCell1}
GcMultiRow1.Template = Template.CreateGridTemplate(cells)
GcMultiRow1.RowCount = 10
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;

NumericUpDownCell numericUpDownCell1 = new NumericUpDownCell();
RangeValidator rangeValidator1 = new RangeValidator();
rangeValidator1.RequiredType = typeof(int);
rangeValidator1.MaxValue = 10;
rangeValidator1.MinValue = 0;
rangeValidator1.Actions.Add(new LineNotify());
numericUpDownCell1.Validators.Add(rangeValidator1);```
Validation of Numerical Values Including Commas

You can convert the string representation of a number to an integer equivalent and perform validation for that converted value, by setting the `RangeValidator.ParsingEnabled ('ParsingEnabled Property' in the on-line documentation) property to True. If the ParsingEnabled property is set to True, you can use the `FormatProvider ('FormatProvider Property' in the on-line documentation) property to specify the culture.

Using Code

The following code validates a numerical value with commas, entered in the cell.

[VB]

```vbnet
Imports System.Globalization
Imports GrapeCity.Win.MultiRow

Dim textBoxCell1 As New TextBoxCell()
textBoxCell1.Style.Format = "#,00"
textBoxCell1.ValueType = GetType(Integer)
Dim rangeValidator1 As New RangeValidator()
rangeValidator1.MaxValue = 10000
rangeValidator1.MinValue = 0
rangeValidator1.ParsingEnabled = True
rangeValidator1.FormatProvider = CultureInfo.CurrentCulture
rangeValidator1.Actions.Add(New LineNotify())
rangeValidator1.RequiredType = GetType(Decimal)
textBoxCell1.Validators.Add(rangeValidator1)

Dim cells As Cell() = {textBoxCell1}
gcMultiRow1.Template = Template.CreateGridTemplate(cells)
gcMultiRow1.RowCount = 10
```

[CS]

```cs
using System.Globalization;
using GrapeCity.Win.MultiRow;

TextBoxCell textBoxCell1 = new TextBoxCell();
textBoxCell1.Style.Format = "#,00";
textBoxCell1.ValueType = typeof(int);
RangeValidator rangeValidator1 = new RangeValidator();
rangeValidator1.MaxValue = 10000;
rangeValidator1.MinValue = 0;
rangeValidator1.ParsingEnabled = true;
rangeValidator1.FormatProvider = CultureInfo.CurrentCulture;
rangeValidator1.Actions.Add(new LineNotify());
rangeValidator1.RequiredType = typeof(Decimal);
textBoxCell1.Validators.Add(rangeValidator1);

Cell[] cells = { textBoxCell1 }; gcMultiRow1.Template = Template.CreateGridTemplate(cells); gcMultiRow1.RowCount = 10;
```
IncludeListValidator

You can use IncludeListValidator (IncludeListValidator Class in the on-line documentation) and validate if a cell's value is one of the values in the specified list. A validation error occurs if the value does not match any of the values in the list.

Using Designer

Complete the following steps to set a list of values. The following example uses a TextBoxCell1 for which a validation error occurs if a string other than Tokyo or Osaka has been entered. Blank characters are recognized as validation errors.

1. Select a cell whose values needs to be validated (for example, textBoxCell1).
2. From the Properties window, select the Validators property and click the ... button.
3. From the displayed CellValidator collection editor, select IncludeListValidator from the combo box on the top-left and click the Add button.
4. From the Members list, confirm that IncludeListValidator has been selected.
5. Select the Candidates property from the property grid and click the ... button.
6. In the displayed TextCollection editor, set the required text, in this example:
   - Tokyo
   - Osaka
7. Click the OK button and close the window.
8. Select the Actions property from the property grid and click the ... button.
10. Click the OK button and close the CellValidateAction collection editor.
11. Click the OK button and close the CellValidator collection editor.
12. Change the document window tab of the designer to Runtime.
13. Type Kyoto, move the focus from the cell, and confirm the occurrence of the validation error.
14. Remove Kyoto and type Osaka, move the focus from the cell, and confirm that the validation error does not occur.

Using Code

The following code displays a validation error when text other than Tokyo or Osaka is entered. The blank character is also recognized as a validation error.

[VB]

Imports GrapeCity.Win.MultiRow

Dim textBoxCell11 As New TextBoxCell()
Dim includeListValidator1 As New IncludeListValidator()
includeListValidator1.Candidates = New String() {"Tokyo", "Osaka"}
includeListValidator1.Actions.Add(New LineNotify())
textBoxCell11.Validators.Add(includeListValidator1)

Dim cells As Cell() = {textBoxCell11}
GcMultiRow1.Template = Template.CreateGridTemplate(cells)
GcMultiRow1.RowCount = 10

[CS]
using GrapeCity.Win.MultiRow;

TextBoxCell textBoxCell1 = new TextBoxCell();
IncludeListValidator includeListValidator1 = new IncludeListValidator();
includeListValidator1.Candidates = new string[] { "Tokyo", "Osaka" };
includeListValidator1.Actions.Add(new LineNotify());
textBoxCell1.Validators.Add(includeListValidator1);

Cell[] cells = { textBoxCell1 };
gcMultiRow1.Template = Template.CreateGridTemplate(cells);
gcMultiRow1.RowCount = 10;

**RequiredTypeValidator**

You can use `RequiredTypeValidator` (*RequiredTypeValidator Class’ in the on-line documentation*) and validate if the value entered in a cell matches the specified type. RequiredTypeValidator supports only DateTime, TimeSpan, and Decimal type (Numeric).

**Using Designer**

Complete the following steps to validate the value type. The example shows TextBoxCell for which a validation error occurs if text entered cannot be converted to a numeric value.

1. Select a cell whose value needs to be validated (for example, textBoxCell1).
2. From the **Properties** window, select the **Validators** property and click the **...** button.
3. From the displayed CellValidator collection editor, select **RequiredTypeValidator** from the top-left combo box and click the **Add** button.
4. From the **Members** list, confirm that RequiredTypeValidator has been selected.
5. Select the **RequiredType** property from the property grid and select datatype **Decimal** that shows numeric values.
6. Select the **Actions** property from the property grid and click the **...** button.
7. Add **LineNotify** in the displayed CellValidateAction collection editor.
8. Click the **OK** button and close the CellValidateAction collection editor.
9. Click the **OK** button and close thenCellValidator collection editor.
10. Change the document window tab of the designer to **Runtime**.
11. Enter text that is not numeric, move focus from the cell, and confirm that the validation error is displayed.

**Using Code**

The following code displays a validation error when the value is not an Integer type.

[VB]

Imports GrapeCity.Win.MultiRow

Dim textBoxCell11 As New TextBoxCell()
textBoxCell11.Value = "Hello"

Dim requiredTypeValidator1 As New RequiredTypeValidator()
requiredTypeValidator1.RequiredType = GetType(Decimal)
requiredTypeValidator1.Actions.Add(New LineNotify())
textBoxCell11.Validators.Add(requiredTypeValidator1)

Dim cells As Cell() = {textBoxCell11}
gcMultiRow1.Template = Template.CreateGridTemplate(cells)
Validation of Numerical Values Including Commas

You can convert the string representation of a number to an integer equivalent and perform validation for that converted value, by setting the RequiredTypeValidator.ParsingEnabled property to True. If the RequiredTypeValidator.ParsingEnabled property is set to True, you can use the FormatProvider property to specify the culture.

Using Code

The following code validates a numerical value with commas entered in the cell.

[VB]

Imports System.Globalization
Imports GrapeCity.Win.MultiRow

Dim textBoxCell1 As New TextBoxCell()
textBoxCell1.Style.Format = "#,00"
textBoxCell1.ValueType = GetType(Int32)
Dim requiredTypeValidator1 As New RequiredTypeValidator()
requiredTypeValidator1.Actions.Add(New LineNotify())
requiredTypeValidator1.RequiredType = GetType(Decimal)
requiredTypeValidator1.ParsingEnabled = True
requiredTypeValidator1.FormatProvider = CultureInfo.CurrentCulture
textBoxCell1.Validators.Add(requiredTypeValidator1)

Dim cells As Cell() = {textBoxCell1}
GcMultiRow1.Template = Template.CreateGridTemplate(cells)
GcMultiRow1.RowCount = 10

[CS]

using System.Globalization;
using System.Globalization;
using GrapeCity.Win.MultiRow;
TextBoxCell textBoxCell1 = new TextBoxCell();
textBoxCell1.Style.Format = "#,###00";
textBoxCell1.ValueType = typeof(Int32);

RequiredTypeValidator requiredTypeValidator1 = new RequiredTypeValidator();
requiredTypeValidator1.Actions.Add(new LineNotify());
requiredTypeValidator1.NullIsValid = true;
requiredTypeValidator1.RequiredType = typeof(Decimal);
requiredTypeValidator1.ParsingEnabled = true;
textBoxCell1.Validators.Add(requiredTypeValidator1);

Cell[] cells = { textBoxCell1 };
GcMultiRow1.Template = Template.CreateGridTemplate(cells);
GcMultiRow1.RowCount = 10;

RequiredFieldValidator

You can use RequiredFieldValidator ('RequiredFieldValidator Class' in the on-line documentation) to validate whether a value has been entered in a cell.

Using Designer

Complete the following instructions to set RequiredFieldValidator.

1. Select a cell whose value needs to be validated (for example, textBoxCell1).
2. From the Properties window, select the Validators property and click the ... button.
3. From the displayed CellValidator collection editor, select RequiredFieldValidator from the top-left combo box and click the Add button.
4. Select the Actions property from the property grid and click the ... button.
5. Add LineNotify in the displayed CellValidateAction collection editor.
6. Click the OK button and close the CellValidateAction collection editor.
7. Click the OK button and close the CellValidator collection editor.
8. Change the document window tab of the designer to Runtime.
9. Move to the next cell without making any change in textBoxCell1 and confirm the occurrence of a validation error.

Using Code

The following code displays a validation error when no value has been entered in a cell.

[VB]
Imports GrapeCity.Win.MultiRow

Dim textBoxCell1 As New TextBoxCell()
Dim requiredFieldValidator1 As New RequiredFieldValidator()
requiredFieldValidator1.Actions.Add(New LineNotify())
textBoxCell1.Validators.Add(requiredFieldValidator1)

Dim cells As Cell() = {textBoxCell1}
GcMultiRow1.Template = Template.CreateGridTemplate(cells)
GcMultiRow1.RowCount = 10
using GrapeCity.Win.MultiRow;

TextBoxCell textBoxCell1 = new TextBoxCell();
RequiredFieldValidator requiredFieldValidator1 = new RequiredFieldValidator();
requiredFieldValidator1.Actions.Add(new LineNotify());
textBoxCell1.Validators.Add(requiredFieldValidator1);

Cell[] cells = { textBoxCell1 }; gcMultiRow1.Template = Template.CreateGridTemplate(cells);
gcMultiRow1.RowCount = 10;

ExcludeListValidator

You can use ExcludeListValidator (’ExcludeListValidator Class' in the on-line documentation) and validate whether the cell’s value matches the specified list of values. A validation error occurs if the value matches any of the values in the list.

Using Designer

Complete the following steps to set ExcludeListValidator. In the example, a validation error occurs if the text **Air Mail** or **Sea Mail** has been entered in the cell. Blank characters are not recognized as a validation error.

1. Select a cell whose value needs to be validated (for example, textBoxCell1).
2. From the Properties window, select theValidators property and click the ... button.
3. From the displayed CellValidator collection editor, select ExcludeListValidator from the top-left combo box and click the Add button.
4. From the Members list, confirm that ExcludeListValidator has been selected.
5. Select the Candidates property from the property grid and click the ... button.
6. In the displayed TextCollection editor, set the strings to be excluded, in this example:
   - **Air Mail**
   - **Sea Mail**
7. Click the OK button and close the window.
8. Select the Actions property from the property grid and click the ... button.
10. Click the OK button and close the CellValidateAction collection editor.
11. Click the OK button and close the CellValidator collection editor.
12. Change the document window tab of the designer to Runtime.
13. Type **Air Mail**, move the focus from the cell, and confirm that the validation error occurs.
14. Remove **Air Mail**, type **Economy Air Mail**, and move the focus from the cell to confirm that the validation error does not occur.

Using Code

The following code displays a validation error when the specified text **Air Mail** and **Sea Mail** has been entered in the cell. Blank characters are not recognized as a validation error.

[VB]

```vbnet
Imports GrapeCity.Win.MultiRow

Dim textBoxCell1 As New TextBoxCell()
Dim excludeListValidator1 As New ExcludeListValidator()
```
CompareCellValidator

You can use `CompareCellValidator` ("CompareCellValidator Class' in the on-line documentation) and validate a cell's value by comparing it to any other cell's value. CompareValueValidator can compare only DateTime, TimeSpan, and Decimal type (Numeric) values.

Using Designer

Complete the following steps to validate cell value by comparison. The example displays a validation error when values of two numericUpDown cells differ.

1. Place two numericUpDown cells (for example, numericUpDownCell1, numericUpDownCell2).
2. Select the cell for which to validate the value (for example, numericUpDownCell2).
3. From the Properties window, select the Validators property and click the ... button.
4. From the displayed CellValidator collection editor, select CompareCellValidator from the top-left combo box and click the Add button.
5. From the Members list, confirm that CompareCellValidator has been selected.
6. Select the ComparedCellName property from the property grid and select numericUpDownCell1.
7. Select the ComparedOperator property from the property grid and click Equals button.
8. Select the Actions property from the property grid and click the ... button.
10. Click the OK button and close the CellValidateAction collection editor.
11. Click the OK button and close the CellValidator collection editor.
12. Change the document window tab of the designer to Runtime.
13. Enter different values in numericUpDownCell1 and numericUpDownCell2 to confirm that the validation error occurs.

Using Code

The following code illustrates the validation error when a cell’s value and the value in an adjacent cell differ.

[VB]

```vbnet
Dim cells As Cell() = {textBoxCell1}
GcMultiRow1.Template = Template.CreateGridTemplate(cells)
GcMultiRow1.RowCount = 10
```
Imports GrapeCity.Win.MultiRow

Dim numericUpDownCell1 As New NumericUpDownCell()
numericUpDownCell1.Name = "numericUpDownCell1"
numericUpDownCell1.Value = 12
Dim numericUpDownCell2 As New NumericUpDownCell()
numericUpDownCell2.Name = "numericUpDownCell2"
numericUpDownCell2.Value = 12

Dim compareCellValidator1 As New CompareCellValidator()
compareCellValidator1.RequiredType = GetType(Integer)
compareCellValidator1.ComparedCellName = "numericUpDownCell2"
compareCellValidator1.Actions.Add(New LineNotify())
numericUpDownCell1.Validators.Add(compareCellValidator1)

Dim cells As Cell() = {numericUpDownCell1, numericUpDownCell2}
GcMultiRow1.Template = Template.CreateGridTemplate(cells)
GcMultiRow1.RowCount = 10

[CS]

using GrapeCity.Win.MultiRow;

NumericUpDownCell numericUpDownCell1 = new NumericUpDownCell();
numericUpDownCell1.Name = "numericUpDownCell1";
numericUpDownCell1.Value = 12;
NumericUpDownCell numericUpDownCell2 = new NumericUpDownCell();
numericUpDownCell2.Name = "numericUpDownCell2";
numericUpDownCell2.Value = 12;

CompareCellValidator compareCellValidator1 = new CompareCellValidator();
compareCellValidator1.RequiredType = typeof(int);
compareCellValidator1.ComparedCellName = "numericUpDownCell2";
compareCellValidator1.Actions.Add(new LineNotify());
numericUpDownCell1.Validators.Add(compareCellValidator1);

Cell[] cells = { numericUpDownCell1, numericUpDownCell2 };
GcMultiRow1.Template = Template.CreateGridTemplate(cells);
GcMultiRow1.RowCount = 10;

CompareValueValidator

You can use CompareValueValidator ('CompareValueValidator Class' in the on-line documentation) and validate a cell's value by comparing it to any other value. CompareValueValidator can compare only DateTime, TimeSpan, and Decimal type (Numeric) values.

Using Designer

Complete the following instructions to validate a cell's value by comparison. The example displays a validation error when the value entered in numericUpDown cell is greater than 5.

1. Place a numericUpDownCell (for example, numericUpDownCell1).
2. Select a cell for which to validate the value (for example, numericUpDownCell1).
3. From the Properties window, select the Validators property and click the ... button.
4. From the displayed CellValidator collection editor, select CompareCellValidator from the left-top combo box and click the Add button.
5. From the Members list, confirm that CompareCellValidator has been selected.
6. Select the ComparedValue property from the property grid and type 5.
7. Select the ComparedOperator property from the property grid and type LessThanOrEquals.
8. Select the Actions property from the property grid and click the ... button.
10. Click the OK button and close the CellValidateAction collection editor.
11. Click the OK button and close the CellValidator collection editor.
12. Change the document window tab of the designer to Runtime.
13. Enter 6 in numericUpDownCell1 to confirm that the validation error occurs.

Using Code

The following code illustrates the validation error when a value entered in a cell is greater than 100.

[VB]

Imports GrapeCity.WinForms.MultiRow

Dim textBoxCell1 As New TextBoxCell()
Dim compareValueValidator1 As New CompareValueValidator()
compareValueValidator1.RequiredType = GetType(Integer)
compareValueValidator1.ComparedValue = 100
compareValueValidator1.ComparedOperator = ValidateComparisonOperator.LessThanOrEquals
compareValueValidator1.Actions.Add(New LineNotify())
textBoxCell1.Validators.Add(compareValueValidator1)

Dim cells As Cell() = {textBoxCell1}
GcMultiRow1.Template = Template.CreateGridTemplate(cells)
GcMultiRow1.RowCount = 10

[CS]

using GrapeCity.WinForms.MultiRow;

TextBoxCell textBoxCell1 = new TextBoxCell();

CompareValueValidator compareValueValidator1 = new CompareValueValidator();
compareValueValidator1.RequiredType = typeof(int);
compareValueValidator1.ComparedValue = 100;
compareValueValidator1.ComparedOperator = ValidateComparisonOperator.LessThanOrEquals;
compareValueValidator1.Actions.Add(new LineNotify());
textBoxCell1.Validators.Add(compareValueValidator1);

Cell[] cells = { textBoxCell1 };
gcMultiRow1.Template = Template.CreateGridTemplate(cells);
gcMultiRow1.RowCount = 10;

PairCharValidator

You can use PairCharValidator (PairCharValidator Class in the on-line documentation) and validate if the
pair characters included in the cell value match each other. For example, you can validate if you have forgotten to put a closing bracket.

**Using Designer**

Complete the following steps to validate pair characters. The example displays a validation error when any bracket does not have its corresponding pair.

1. Select a cell whose value needs to be validated (for example, textBoxCell1).
2. From the **Properties** window, select the **Validators** property and click the **...** button.
3. From the displayed CellValidator collection editor, select **PairCharValidator** from the top-left combo box and click the **Add** button.
4. From the **Members** list, confirm that **PairCharValidator** has been selected.
5. Select the **PairChars** property from the property grid and click the **...** button.
6. Add **PairChar** in the displayed PairChar collection editor.
7. In the added PairChar, set the **Left** property to ( and the **Right** property to ).
8. Click the **OK** button and close the PairChar collection editor.
9. Select the **Actions** property from the property grid and click the **...** button.
10. Add **LineNotify** in the displayed CellValidateAction collection editor.
11. Click the **OK** button and close the CellValidateAction collection editor.
12. Click the **OK** button and close the CellValidator collection editor.
13. Change the document window tab of the designer to **Runtime**.
14. Enter **((1+2)*3** in textBoxCell1 to confirm that the validation error occurs.

**Using Code**

The following code displays a validation error when text entered in a cell does not have the corresponding bracket.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

Dim textBoxCell1 As New TextBoxCell()
Dim pairCharValidator1 As New PairCharValidator()
Dim parChar1 As New PairChar("(", ")")
pairCharValidator1.PairChars.Add(parChar1)
pairCharValidator1.Actions.Add(New LineNotify())
textBoxCell1.Validators.Add(pairCharValidator1)

Dim cells As Cell() = {textBoxCell1}
GcMultiRow1.Template = Template.CreateGridTemplate(cells)
GcMultiRow1.RowCount = 10
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;

TextBoxCell textBoxCell1 = new TextBoxCell();
PairCharValidator pairCharValidator = new PairCharValidator();
PairChar pairChar1 = new PairChar('(' , ')');
pairCharValidator.PairChars.Add(pairChar1);
pairCharValidator.Actions.Add(new LineNotify());
textBoxCell1.Validators.Add(pairCharValidator);

Cell[] cells = { textBoxCell1 };```

---

Copyright © GrapeCity, inc. All rights reserved.
SurrogateCharValidator

You can use **SurrogateCharValidator** (**SurrogateCharValidator Class' in the on-line documentation**) and validate if surrogate characters have been entered in the cell.

**Using Designer**

Complete the following steps to validate surrogate characters. The example displays a validation error if characters other than double byte katakana have been entered in textBoxCell.

1. Select a cell for which to validate the value (for example, textBoxCell1).
2. From the properties window, select the **Validators** property and click the **...** button.
3. From the displayed CellValidator collection editor, select **SurrogateCharValidator** from the top-left combo box and click the **Add** button.
4. From the **Members** list, confirm that SurrogateCharValidator has been selected.
5. Select the **Actions** property from the property grid and click the **...** button.
6. Add **LineNotify** in the displayed CellValidateAction collection editor.
7. Click the **OK** button and close the CellValidateAction collection editor.
8. Click the **OK** button and close the CellValidator collection editor.
9. Change the document window tab of the designer to **Runtime**.
10. Enter environment–dependent characters (for example, one of the conversion results of しかる(Japanese characters)).

**Using Code**

The following code displays a validation error when surrogate characters are entered into the cell.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

Dim textBoxCell11 As New TextBoxCell()
Dim surrogateCharValidator1 As New SurrogateCharValidator()
surrogateCharValidator1.Actions.Add(New LineNotify())
textBoxCell11.Validators.Add(surrogateCharValidator1)

Dim cells As Cell() = {textBoxCell11}
gcMultiRow1.Template = Template.CreateGridTemplate(cells)
gcMultiRow1.RowCount = 10
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;

TextBoxCell textBoxCell11 = new TextBoxCell();
SurrogateCharValidator surrogateCharValidator1 = new SurrogateCharValidator();
surrogateCharValidator1.Actions.Add(new LineNotify());
textBoxCell11.Validators.Add(surrogateCharValidator1);

Cell[] cells = { textBoxCell11 };
gcMultiRow1.Template = Template.CreateGridTemplate(cells);
```
(gcMultiRow1.RowCount = 10;

RegularExpressionValidator

You can use **RegularExpressionValidator** ('RegularExpressionValidator Class' in the on-line documentation) and validate if a cell's value matches the specified regular expression. For further details about Regular Expressions, refer to [.NET Framework Regular Expressions](#).

**Using Designer**

Complete the following steps to validate using a regular expression. The example displays a validation error if a string other than an e-mail address has been entered in the textBoxCell.

1. Select a cell for which to validate the value (for example, textBoxCell1).
2. From the **Properties** window, select the **Validators** property and click the **...** button.
3. From the displayed CellValidator collection editor, select **RegularExpressionValidator** from the top-left combo box and click the **Add** button.
4. From the **Members** list, confirm that **RegularExpressionValidator** has been selected.
5. Select the **Expression** property from the property grid and click the **...** button for pre-defined expressions.
6. Select the expression for e-mail address \w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)* and click the **OK** button.
7. Select the **Actions** property from the property grid and click the **...** button.
8. Add **LineNotify** in the displayed CellValidateAction collection editor.
9. Click the **OK** button and close the CellValidateAction collection editor.
10. Click the **OK** button and close the CellValidator collection editor.
11. Change the document window tab of the designer to **Runtime**.
12. Enter **john.willsonxyz.com**, move the focus from the cell, and verify the occurrence of a validation error.
13. Now add the @ sign to correct the e-mail address and move to another cell. Confirm that no error occurs.

**Using Code**

The following code displays a validation error if a string other than an e-mail address has been entered in the cell.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

Dim textBoxCell1 As New TextBoxCell()
Dim regularExpressionValidator1 As New RegularExpressionValidator()
regularExpressionValidator1.Expression = "\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*"
regularExpressionValidator1.Actions.Add(New LineNotify())
textBoxCell1.Validators.Add(regularExpressionValidator1)

Dim cells As Cell() = {textBoxCell1}
GcMultiRow1.Template = Template.CreateGridTemplate(cells)
GcMultiRow1.RowCount = 10
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;

TextBoxCell textBoxCell1 = new TextBoxCell();
RegularExpressionValidator regularExpressionValidator1 = new RegularExpressionValidator();
regularExpressionValidator1.Expression = @"\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*";
regularExpressionValidator1.Actions.Add(new LineNotify());
```
**textBoxCell1.Validators.Add(regularExpressionValidator1);**

Cell[] cells = {textBoxCell1};
gcMultiRow1.Template = Template.CreateGridTemplate(cells);
gcMultiRow1.RowCount = 10;

**CompareStringValidator**

You can use the compare string validator to validate whether a cell's value matches the specified conditions. A validation error occurs if the value does not match the specified conditions.

**Using the Designer**

Complete the following steps to validate a cell's value by comparison.

1. Select a cell for which you want to validate the value (for example, textBoxCell1).
2. From the Properties window, select the Validators property and click the ... button.
3. From the displayed CellValidator Collection editor, select CompareStringValidator from the top-left combo box and click the Add button.
4. From the Members list, confirm that CompareStringValidator has been selected.
5. Select the ComparedOperator property from the property grid and select Equals.
6. Select the ComparedString property from the property grid and type AAA.
7. Select the Actions property from the property grid and click the ... button.
9. Click the OK button and close the CellValidateAction collection editor.
10. Click the OK button and close the CellValidator collection editor.
11. Change the document window tab of the designer to Runtime.
12. Enter BBB and confirm that the validation error occurs when you move to another cell.
13. Delete BBB, enter AAA, and confirm that the validation error does not occur when you move to another cell.

**Using Code**

The following code illustrates the validation error when a string other than "AAA" is entered in the cell. Blank text is also recognized as a validation error.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow
Dim textBoxCell1 As New TextBoxCell()
Dim compareStringValidator1 As New CompareStringValidator()
compareStringValidator1.ComparedString = "AAA"
compareStringValidator1.Actions.Add(New LineNotify())
textBoxCell1.Validators.Add(compareStringValidator1)
Dim cells As Cell() = {textBoxCell1}
GcMultiRow1.Template = Template.CreateGridTemplate(cells)
GcMultiRow1.RowCount = 10
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;
TextBoxCell textBoxCell1 = new TextBoxCell();
CompareStringValidator compareStringValidator1 = new CompareStringValidator();
```
TextLengthValidator

You can use the TextLengthValidator (‘TextLengthValidator Class’ in the on-line documentation) class to validate whether the height of a cell’s value falls within the specified range. A validation error occurs if the height of the value does not fall within the specified range.

Using the Designer

Use the following instructions to validate the height of a cell’s value.

1. Select a cell for which to validate the value (for example, textBoxCell1).
2. From the Properties window, select the Validators property and click the ... button.
3. From the displayed CellValidator collection editor, select TextLengthValidator from the left-top combo box and click the Add button.
4. From the Members list, confirm that TextLengthValidator has been selected.
5. Select the Encoding property from the property grid and select Unicode.
6. Select the LengthUnit property from the property grid and select Byte.
7. Select the MaximumLength property from the property grid and type 5.
8. Select the Actions property from the property grid and click the ... button.
10. Click the OK button and close the CellValidateAction collection editor.
11. Click the OK button and close the CellValidator collection editor.
12. Change the document window tab of the designer to Runtime.
13. Enter AAAAAA and confirm that the validation error occurs when you move to another cell.
14. Delete AAAAAA, enter AAA and confirm that the validation error does not occur when you move to another cell.

Using Code

The following code illustrates the validation error when a string of more than 5 bytes is entered in the cell.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

Dim textBoxCell11 As New TextBoxCell()
Dim textLengthValidator1 As New TextLengthValidator()
textLengthValidator1.LengthUnit = LengthUnit.Byte
textLengthValidator1.MaximumLength = 5
textBoxCell11.Validators.Add(New LineNotify())
textBoxCell11.Validators.Add(textLengthValidator1)

Dim cells As Cell() = {textBoxCell11}
GcMultiRow1.Template = Template.CreateGridTemplate(cells)
GcMultiRow1.RowCount = 10
```

**[CS]**

```csharp
Dim textBoxCell11 As New TextBoxCell()
Dim textLengthValidator1 As New TextLengthValidator()
textLengthValidator1.LengthUnit = System.Text.Encoding.GetEncoding("Unicode")
textLengthValidator1.MaximumLength = 5
textBoxCell11.Validators.Add(new LineNotify())
textBoxCell11.Validators.Add(textLengthValidator1)

Dim cells As Cell() = {textBoxCell11}
GcMultiRow1.Template = Template.CreateGridTemplate(cells)
GcMultiRow1.RowCount = 10
```
using GrapeCity.Win.MultiRow;

TextBoxCell textBoxCell1 = new TextBoxCell();
TextLengthValidator textLengthValidator1 = new TextLengthValidator();
textLengthValidator1.LengthUnit = LengthUnit.Byte;
textLengthValidator1.MaximumLength = 5;
textLengthValidator1.Actions.Add(new LineNotify());
textBoxCell1.Validators.Add(textLengthValidator1);

Cell[] cells = { textBoxCell1 };
gcMultiRow1.Template = Template.CreateGridTemplate(cells);

cells[0].Text = "Hello World";
Built-in Validation Actions

MultiRow provides six types of built-in validation actions. The actions have default behaviors that notify the user in case of a validation error.

- Notification using Underline (LineNotify)
- Notification using CellStyle (CellStyleNotify)
- Notification using Icon (IconNotify)
- Notification using Sound (SoundNotify)
- TipNotify
- Notification using Focus Change (FocusProcess)
- ValueProcess
- Notification using Message Box (MessageBoxNotify)
- Notification using Three-State Icon

Notification using Underline (LineNotify)

You can use LineNotify ('LineNotify Class' in the on-line documentation) to notify the user of a validation error by underlining the value entered. The underline is drawn for each cell, so it might not match the string of the cell. You can also change the color of the underline. LineNotify is not affected by the GcMultiRow.ShowWaveLineInEditingStatus ('ShowWaveLineInEditingStatus Property' in the on-line documentation) property.

Using Designer

Complete the following steps to use LineNotify for cell validation.

1. Select a cell for which to validate the value (for example, textBoxCell1).
2. From the Properties window, select the Validators property and click the ... button.
3. From the displayed CellValidator collection editor, select RequiredFieldValidator from the top-left combo box and click the Add button.
4. Select the Actions property from the property grid at the right of the screen and click the ... button.
5. Add LineNotify in the displayed CellValidateAction collection editor.
6. Click the OK button and close the CellValidateAction collection editor.
7. Click the OK button and close the CellValidator collection editor.
8. Change the document window tab of the designer to Runtime.
9. Move to the next cell without making any changes in the selected cell and you can confirm that the validation error is shown.
Using Code

The following code displays an underline as the validation error when the value in textBoxCell is blank.

[VB]

Imports GrapeCity.Win.MultiRow

Dim textBoxCell1 As New TextBoxCell()
Dim requiredFieldValidator1 As New RequiredFieldValidator()
requiredFieldValidator1.Actions.Add(New LineNotify())
textBoxCell1.Validators.Add(requiredFieldValidator1)

Dim cells As Cell() = {textBoxCell1}
gcMultiRow1.Template = Template.CreateGridTemplate(cells)
gcMultiRow1.RowCount = 10

[CS]

using GrapeCity.Win.MultiRow;

TextBoxCell textBoxCell1 = new TextBoxCell();
RequiredFieldValidator requiredFieldValidator1 = new RequiredFieldValidator();
requiredFieldValidator1.Actions.Add(new LineNotify());
textBoxCell1.Validators.Add(requiredFieldValidator1);

Cell[] cells = { textBoxCell1 };
gcMultiRow1.Template = Template.CreateGridTemplate(cells);
gcMultiRow1.RowCount = 10;

Notification using CellStyle (CellStyleNotify)

You can use the CellStyleNotify ('CellStyleNotify Class' in the on-line documentation) object to change the cell style to notify users of a validation error. For example, you can change the background color or foreground color of the cell to red.

Using Designer

Use the following steps to use CellStyleNotify for notification.

1. Select a cell for which to validate the value (for example, textBoxCell1).
2. From the Properties window, select Validators property and click the ... button.
3. From the displayed CellValidator collection editor, select RequiredFieldValidator from the top-left combo box and click the Add button.
4. Select the Actions property from the property grid at the right of screen and click the ... button.
5. Add CellStyleNotify in the displayed CellValidateAction collection editor.
6. From the Members list, confirm that CellStyleNotify has been selected. From the property grid at the right of screen, set any style in the InvalidCellStyle property (for example, set the BackColor property to Red).
7. Click the OK button and close the CellValidateAction collection editor.
8. Click the OK button and close the CellValidator collection editor.
9. Change the document window tab of the designer to Runtime.
10. Move to the next cell without making any changes in your selected cell and you can confirm that the validation notification appears.
Using Code

The following code changes the background color of a cell to indicate a validation error when the textBoxCell is blank.

[VB]

```vbnet
Imports GrapeCity.Win.MultiRow

Dim textBoxCell1 As New TextBoxCell()
Dim requiredFieldValidator1 As New RequiredFieldValidator()
Dim cellStyleNotify1 As New CellStyleNotify()
    cellStyleNotify1.InvalidCellStyle.BackColor = Color.Red
    requiredFieldValidator1.Actions.Add(cellStyleNotify1)
    textBoxCell1.Validators.Add(requiredFieldValidator1)

Dim cells As Cell() = {textBoxCell1}
GcMultiRow1.Template = Template.CreateGridTemplate(cells)
GcMultiRow1.RowCount = 10
```

[CS]

```cs
using GrapeCity.Win.MultiRow;

TextBoxCell textBoxCell1 = new TextBoxCell();
RequiredFieldValidator requiredFieldValidator1 = new RequiredFieldValidator();
CellStyleNotify cellStyleNotify1 = new CellStyleNotify();
    cellStyleNotify1.InvalidCellStyle.BackColor = Color.Red;
    requiredFieldValidator1.Actions.Add(cellStyleNotify1);
    textBoxCell1.Validators.Add(requiredFieldValidator1);

Cell[] cells = { textBoxCell1 };
gcMultiRow1.Template = Template.CreateGridTemplate(cells);
gcMultiRow1.RowCount = 10;
```

Notification using Icon (IconNotify)

You can use IconNotify ('IconNotify Class' in the on-line documentation) to notify the user of a validation error by displaying an error icon. You can change the category of the error icon, its display position, and its tooltip. IconNotify is not affected by the GcMultiRow. ShowErrorIconInEditingStatus ('ShowErrorIconInEditingStatus Property' in the on-line documentation) property.

Using Designer

Complete the following steps to use IconNotify for cell validation.

1. Select a cell for which to validate the value (for example, textBoxCell1).
2. From the Properties window, select the Validators property and click the ... button.
3. From the displayed CellValidator collection editor, select RequiredFieldValidator from the top-left combo box and click the Add button.
4. Select the Actions property from the property grid at the right of the screen and click the ... button.
5. Add IconNotify in the displayed CellValidateAction collection editor.
6. Click the OK button and close the CellValidateAction collection editor.
7. Click the OK button and close the CellValidator collection editor.
8. Change the document window tab of the designer to Runtime.
9. Move to the next cell without making any changes in the selected cell and you can confirm that the validation error is displayed.

**Using Code**

The following code displays an error icon to depict a validation error when the value in `textBoxCell` is blank.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

Dim textBoxCell1 As New TextBoxCell()
Dim requiredFieldValidator1 As New RequiredFieldValidator()
requiredFieldValidator1.Actions.Add(New IconNotify())
textBoxCell1.Validators.Add(requiredFieldValidator1)

Dim cells As Cell() = {textBoxCell1}
GcMultiRow1.Template = Template.CreateGridTemplate(cells)
GcMultiRow1.RowCount = 10
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;

TextBoxCell textBoxCell1 = new TextBoxCell();
RequiredFieldValidator requiredFieldValidator1 = new RequiredFieldValidator();
requiredFieldValidator1.Actions.Add(new IconNotify());
textBoxCell1.Validators.Add(requiredFieldValidator1);

Cell[] cells = { textBoxCell1 };
gcMultiRow1.Template = Template.CreateGridTemplate(cells);
gcMultiRow1.RowCount = 10;
```

**Notification using Sound (SoundNotify)**

You can use **SoundNotify** ([`SoundNotify Class in the on-line documentation`](https://www.grapewebsite.com)) to notify the user of a validation error using a sound. This generated sound can be any of the following system sounds. In addition, you can use the `SoundNotify.SoundLocation` ([`SoundLocation Property in the on-line documentation`](https://www.grapewebsite.com)) property to notify of a validation error by using a specified sound.

- Asterisk
- Beep
- Exclamation
- Hand
- Question

- If the value of the `SoundNotify.SoundLocation` property is empty, the sound set in the `SoundNotify.SoundType` ([`SoundType Property in the on-line documentation`](https://www.grapewebsite.com)) property is used.
- Only a .wav file can be used for the `SoundNotify.SoundLocation` property.
- If the file specified in `SoundNotify.SoundLocation` property does not exist, the sound is not played.
Complete the following steps to use SoundNotify for cell validation.

1. Select a cell for which to validate the value (for example, textBoxCell1).
2. From the Properties window, select the Validators property and click the ... button.
3. From the displayed CellValidator collection editor, select RequiredFieldValidator from the top-left combo box and click the Add button.
4. Select the Actions property from the property grid at the right of the screen and click the ... button.
5. Add SoundNotify in the displayed CellValidateAction collection editor.
6. Click the OK button and close the CellValidateAction collection editor.
7. Click the OK button and close the CellValidator collection editor.
8. Change the document window tab of the designer to Runtime.
9. Move to the next cell without making any changes in the selected cell and you can confirm that the validation error is displayed.

**Using Code**

The following code generates a sound as the validation error when the value in textBoxCell is blank.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

Dim textBoxCell1 As New TextBoxCell()
Dim requiredFieldValidator1 As New RequiredFieldValidator()
requiredFieldValidator1.Actions.Add(New SoundNotify())
textBoxCell1.Validators.Add(requiredFieldValidator1)

Dim cells As Cell() = {textBoxCell1}
GcMultiRow1.Template = Template.CreateGridTemplate(cells)
GcMultiRow1.RowCount = 10
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;

TextBoxCell textBoxCell1 = new TextBoxCell();
RequiredFieldValidator requiredFieldValidator1 = new RequiredFieldValidator();
requiredFieldValidator1.Actions.Add(new SoundNotify());
textBoxCell1.Validators.Add(requiredFieldValidator1);

Cell[] cells = { textBoxCell1 };
gcMultiRow1.Template = Template.CreateGridTemplate(cells);
gcMultiRow1.RowCount = 10;
```

**Trouble Shooting**

If the sound is not generated, check the following:

- Whether the system sound has been assigned in the Control Panel.
- Whether sound device is installed in Windows.
- Whether the sound in Windows is set to Mute.
- Whether the volume of sound in Windows is too low.
- Whether the speakers are connected to the computer.
- Whether the speakers (hardware) are set to Mute.
- Whether volume of the speaker is too low.
TipNotify

You can use **TipNotify (TipNotify Class in the on-line documentation)** to notify the user of a validation error using a balloon tip.

**Using Designer**

Complete the following steps to use TipNotify for cell validation.

1. Select a cell for which to validate the value (for example, textBoxCell1).
2. From the **Properties** window, select the **Validators** property and click the ... button.
3. From the displayed CellValidator collection editor, select **RequiredFieldValidator** from the top-left combo box and click the **Add** button.
4. Select the **Actions** property from the property grid at the right of the screen and click the ... button.
5. Add **TipNotify** in the displayed CellValidateAction collection editor.
6. From the **Members** list, confirm that **TipNotify** has been selected. From the property grid at the right of the screen, type the title in the **ToolTipTitle** property and the description in the **ToolTipText** property.
7. Click the **OK** button and close the CellValidateAction collection editor.
8. Click the **OK** button and close the CellValidator collection editor.
9. Change the document window tab of the designer to **Runtime**.
10. Move to the next cell without making any changes in the selected cell and you can confirm that the validation error is displayed.

**Using Code**

The following code displays a balloon tooltip to indicate a validation error when the value in textBoxCell is blank.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow

Dim textBoxCell1 As New TextBoxCell()
Dim requiredFieldValidator1 As New RequiredFieldValidator()
Dim tipNotify1 As New TipNotify()

tipNotify1.ToolTipTitle = "Title"
tipNotify1.ToolTipText = "Error"
requiredFieldValidator1.Actions.Add(tipNotify1)
textBoxCell1.Validators.Add(requiredFieldValidator1)

Dim cells As Cell() = {textBoxCell1}
GcMultiRow1.Template = Template.CreateGridTemplate(cells)
GcMultiRow1.RowCount = 10
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;

TextBoxCell textBoxCell1 = new TextBoxCell();
RequiredFieldValidator requiredFieldValidator1 = new RequiredFieldValidator();
TipNotify tipNotify1 = new TipNotify();
tipNotify1.ToolTipTitle = "Title";
tipNotify1.ToolTipText = "Error";
requiredFieldValidator1.Actions.Add(tipNotify1);
textBoxCell1.Validators.Add(requiredFieldValidator1);
```
Notification using Focus Change (FocusProcess)

You can use FocusProcess ('FocusProcess Class' in the on-line documentation) to control the focus movement and keep it in the current cell when a validation error occurs.

Using Designer

Complete the following steps to use FocusProcess for cell validation.

1. Select a cell for which to validate the value (for example, textBoxCell1).
2. From the Properties window, select the Validators property and click the ... button.
3. From the displayed CellValidator collection editor, select the RequiredFieldValidator from the left-top combo box and click the Add button.
4. Select the Actions property from the property grid at the right of the screen and click the ... button.
5. Add FocusProcess in the displayed CellValidateAction collection editor.
6. Click OK button and close the CellValidateAction collection editor.
7. Click OK button and close the CellValidator collection editor.
8. Change the document window tab of the designer to Runtime.
9. Move to the next cell without making any changes in the selected cell and you can confirm that the focus does not move.

Using Code

The following code keeps the focus in the textBoxCell when the cell’s value is blank.

[VB]

Imports GrapeCity.Win.MultiRow
Dim textBoxCell11 As New TextBoxCell()
Dim requiredFieldValidator1 As New RequiredFieldValidator()
Dim focusProcess1 As New FocusProcess()
requiredFieldValidator1.Actions.Add(focusProcess1)
textBoxCell11.Validators.Add(requiredFieldValidator1)

Dim cells As Cell() = {textBoxCell11}
GcMultiRow1.Template = Template.CreateGridTemplate(cells)
GcMultiRow1.RowCount = 10

[CS]

using GrapeCity.Win.MultiRow;

TextBoxCell textBoxCell11 = new TextBoxCell();
RequiredFieldValidator requiredFieldValidator1 = new RequiredFieldValidator();
FocusProcess focusProcess1 = new FocusProcess();
requiredFieldValidator1.Actions.Add(focusProcess1);
textBoxCell11.Validators.Add(requiredFieldValidator1);

Cell[] cells = { textBoxCell11 };
ValueProcess

You can use the **ValueProcess** (*ValueProcess Class’ in the on-line documentation) class to perform the processes of retaining, deleting, or restoring the previous value when a validation error occurs. ValueProcess is executed when the cell validation or cell editing state ends.

**Using the Designer**

Use the following steps to use the value process for cell validation.

1. Select a cell value to validate (for example, textBoxCell1).
2. From the **Properties** window, select the **Validators** property and click the ... button.
3. From the displayed **CellValue** collection editor, select **IncludeListValidator** from the left-top combo box and click the Add button.
4. From the **Members** list, confirm that **IncludeListValidator** has been selected.
5. Select the **Candidates** property from the property grid and click the ... button.
6. Set the required string in the displayed **String Collection Editor**. For example:
   - New York
   - California
7. Click the OK button to close the window.
8. Select the **Actions** property from the property grid and click the ... button.
9. Add **ValueProcess** in the displayed **CellValueAction** collection editor.
10. From the **Members** list, confirm that **ValueProcess** has been selected. In the property grid, set the **DoActionReason** property to CellValidating, and **ValueProcessOption** property to Clear.
11. Click the OK button to close the **CellValueAction** collection editor.
12. Click the OK button to close the **CellValue** collection editor.
13. Change the document window tab of the designer to **Runtime**.
14. Enter **Washington** in textBoxCell1 to confirm that the validation error occurs and the entered value gets deleted, when you move to another cell.

**Using Code**

The following code illustrates the validation error, and deletes the entered value when a string other than "New York" or "California" is entered in the cell.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow
Dim textBoxCell1 As New TextBoxCell()
Dim includeListValidator1 As New IncludeListValidator()
Dim valueProcess1 As New ValueProcess()
valueProcess1.DoActionReason = ValidateReasons.CellValidating
includeListValidator1.Candidates = New String() {"New York", "California"}
includeListValidator1.Actions.Add(valueProcess1)
textBoxCell1.Validators.Add(includeListValidator1)
Dim cells As Cell() = {textBoxCell1}
GcMultiRow1.Template = Template.CreateGridTemplate(cells)
GcMultiRow1.RowCount = 10;
```
using GrapeCity.Win.MultiRow;
TextBoxCell textBoxCell1 = new TextBoxCell();
IncludeListValidator includeListValidator1 = new IncludeListValidator();
ValueProcess valueProcess1 = new ValueProcess();
includeListValidator1.Candidates = new string[] { "New York", "California" };
includeListValidator1.Actions.Add(valueProcess1);
textBoxCell1.Validators.Add(includeListValidator1);
Cell[] cells = { textBoxCell1 };
gcMultiRow1.Template = Template.CreateGridTemplate(cells);

diff

If the DoActionReason ('DoActionReason Property' in the on-line documentation) property is set to EndEdit, then only the timing at which the value process is executed differs from when it is set to CellValidating, but the end result is the same.

Notification using Message Box (MessageBoxNotify)

You can use the MessageBoxNotify ('MessageBoxNotify Class' in the on-line documentation) class to notify the user of a validation error by displaying a message box. You can change the title, text, and icon of the message box.

Using the Designer

Use the following steps to use a message box for cell validation.

1. Select a cell value to validate (for example, textBoxCell1).
2. From the Properties window, select the Validators property and click the ... button.
3. From the displayed CellValidator collection editor, select RequiredFieldValidator from the left-top combo box and click the Add button.
4. Select the Actions property from the property grid and click the ... button.
5. Add MessageBoxNotify in the displayed CellValidateAction collection editor.
6. From the Members list, confirm that MessageBoxNotify has been selected.
7. Select the Caption property from the property grid and enter the title.
8. Set the DoActionReason property in the property grid to CellValidating.
9. Set the Icon property in the property grid to Hand.
10. Select the Message property from the property grid and type the description.
11. Click the OK button to close the CellValidateAction collection editor.
12. Click the OK button to close the CellValidator collection editor.
13. Change the document window tab of the designer to Runtime.
14. Confirm that the validation error is displayed when you move to the next cell without making changes in textBoxCell1.

Using Code

The following code displays the validation error in a message box when a String type cell value is blank.

[VB]
Imports GrapeCity.Win.MultiRow
Dim textBoxCell1 As New TextBoxCell()
Dim requiredFieldValidator1 As New RequiredFieldValidator()
Dim messageBoxNotify1 As New MessageBoxNotify()
messageBoxNotify1.Caption = "Title"
messageBoxNotify1.DoActionReason = ValidateReasons.CellValidating
messageBoxNotify1.Icon = MessageBoxIcon.Hand
messageBoxNotify1.Message = "Error"
requiredFieldValidator1.Actions.Add(messageBoxNotify1)
textBoxCell1.Validators.Add(requiredFieldValidator1)
Dim cells As Cell() = {textBoxCell1}
GcMultiRow1.Template = Template.CreateGridTemplate(cells)
GcMultiRow1.RowCount = 10

[CS]

using GrapeCity.Win.MultiRow;
TextBoxCell textBoxCell1 = new TextBoxCell();
RequiredFieldValidator requiredFieldValidator1 = new RequiredFieldValidator();
MessageBoxNotify messageBoxNotify1 = new MessageBoxNotify();
messageBoxNotify1.Caption = "Title";
messageBoxNotify1.DoActionReason = ValidateReasons.CellValidating;
messageBoxNotify1.Icon = MessageBoxIcon.Hand;
messageBoxNotify1.Message = "Error";
requiredFieldValidator1.Actions.Add(messageBoxNotify1);
textBoxCell1.Validators.Add(requiredFieldValidator1);
Cell[] cells = { textBoxCell1 };
gcMultiRow1.Template = Template.CreateGridTemplate(cells);
gcMultiRow1.RowCount = 10;

Notification using Three-State Icon

You can use the **ThreeStateIconNotify** (*ThreeStateIconNotify Class* in the on-line documentation) class to notify the user of an error by displaying an icon when the input value is valid and when it is invalid. In addition, the icon is not shown if nothing is entered.

**Using the Designer**

Use the following steps to use three-state notification for cell validation.

1. Select a cell value to validate (for example, numericUpDownCell1).
2. From the **Properties** window, select the **Validators** property and click the ... button.
3. From the **CellValidator** collection editor, select **RangeValidator** from the left-top combo box and click the **Add** button.
4. From the **Members** list, confirm that **RangeValidator** has been selected.
5. Select the **MaxValue** property from the property grid and type **10**.
6. Select the **MinValue** property from the property grid and type **0**.
7. Click the **OK** button and close the window.
8. Select the **Actions** property from the property grid and click the ... button.
9. Add **ThreeStateIconNotify** in the **CellValidateAction** collection editor.
10. From the **Members** list, confirm that **ThreeStateIconNotify** has been selected. Set the **DoActionReason** property in the property grid to EndEdit.
11. Click the **OK** button to close the **CellValidateAction** collection editor.
12. Click the **OK** button to close the **CellValidator** collection editor.
13. Change the document window tab of the designer to **Runtime**.
14. Enter **20** in the numericUpDownCell1 and check that the error icon is displayed when you move to another cell.
15. Enter **5** in the numericUpDownCell1 and check that the icon for the valid value is displayed when you move to another cell.

**Using Code**

Note: Replace the code in the code example with the code that you want to use in your application.
The following code displays the error icon when a value outside the range of 0 to 10 is entered in the cell, and displays the icon for a valid value when a value within the range of 0 to 10 is entered.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow
Dim numericUpDownCell1 As New NumericUpDownCell()
Dim rangeValidator1 As New RangeValidator()
rangeValidator1.RequiredType = GetType(Integer)
rangeValidator1.MaxValue = 10
rangeValidator1.MinValue = 0
rangeValidator1.Actions.Add(New ThreeStateIconNotify())
numericUpDownCell1.Validators.Add(rangeValidator1)
Dim cells As Cell() = {numericUpDownCell1} GcMultiRow1.Template = Template.CreateGridTemplate(cells)
GcMultiRow1.RowCount = 10
```

**[CS]**

```cs
using GrapeCity.Win.MultiRow;
NumericUpDownCell numericUpDownCell1 = new NumericUpDownCell();
RangeValidator rangeValidator1 = new RangeValidator();
rangeValidator1.RequiredType = typeof(int);
rangeValidator1.MaxValue = 10;
rangeValidator1.MinValue = 0;
rangeValidator1.Actions.Add(new ThreeStateIconNotify());
numericUpDownCell1.Validators.Add(rangeValidator1);
Cell[] cells = {numericUpDownCell1};
gcMultiRow1.Template = Template.CreateGridTemplate(cells);
gcMultiRow1.RowCount = 10;
```

-The BlinkRate, BlinkStyle, and IconPadding properties that are available in the ThreeStateIconNotify property of InputMan are not supported.

Customizing the Validation Information

The GcMultiRow.CellValidateInfoNeeded ('CellValidateInfoNeeded Event' in the on-line documentation) event occurs when the validation of a cell is performed. You can use this event to customize the information that is displayed as the validation result.

You can use the IsValid ('IsValid Property' in the on-line documentation) property to retrieve whether the validation was successful in the GcMultiRow.CellValidateInfoNeeded event. In addition, you can use the ValidateContext.ValidateInfo ('ValidateInfo Property' in the on-line documentation) property to set the validation information.

Using Code

The following code demonstrates when the validation is successful, and when there is an error when using the RangeValidator and Notification using Three-State Icon.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow
Private Sub Form1_Load(sender As System.Object, e As System.EventArgs) Handles MyBase.Load
Dim numericUpDownCell1 As New NumericUpDownCell()
Dim rangeValidator1 As New RangeValidator()
rangeValidator1.RequiredType = GetType(Integer)
rangeValidator1.MaxValue = 10
rangeValidator1.MinValue = 1
rangeValidator1.NullIsValid = False
rangeValidator1.Actions.Add(New ThreeStateIconNotify())
numericUpDownCell1.Validators.Add(rangeValidator1)
Dim cells As Cell() = {numericUpDownCell1} GcMultiRow1.Template = Template.CreateGridTemplate(cells)
GcMultiRow1.RowCount = 10
End Sub
Private Sub GcMultiRow1_CellValidateInfoNeeded(sender As Object, e As CellValidateInfoNeededEventArgs) Handles GcMultiRow1.CellValidateInfoNeeded
If TypeOf e.CellValidator Is RangeValidator
Then If e.IsValid
Then e.ValidateContext.ValidateInfo = "This value is correct."
Else
If e.ValidateContext.EditedFormattedValue > 10
Then e.ValidateContext.ValidateInfo = "Please enter a value which is less than or equal to 10."
Else
If e.ValidateContext.EditedFormattedValue < 1
Then e.ValidateContext.ValidateInfo = "Please enter a value which is greater than 1."
End If
End If
End If
End If
End If
End Sub
```

**[CS]**
using GrapeCity.Win.MultiRow;
private void Form1_Load(object sender, EventArgs e)
{
    NumericUpDownCell numericUpDownCell1 = new NumericUpDownCell();
    RangeValidator rangeValidator1 = new RangeValidator();
    rangeValidator1.RequiredType = typeof(int);
    rangeValidator1.MaxValue = 10;
    rangeValidator1.MinValue = 1;
    rangeValidator1.NullIsValid = false;
    rangeValidator1.Actions.Add(new ThreeStateIconNotify());
    numericUpDownCell1.Validators.Add(rangeValidator1);
    Cell[] cells = { numericUpDownCell1 };
    gcMultiRow1.Template = Template.CreateGridTemplate(cells);
    gcMultiRow1.RowCount = 10;
}
private void gcMultiRow1_CellValidateInfoNeeded(object sender, CellValidateInfoNeededEventArgs e)
{
    if (e.CellValidator is RangeValidator)
    {
        if (e.IsValid)
        {
            e.ValidateContext.ValidateInfo = "This value is correct.";
        }
        else
        {
            if (Convert.ToInt32(e.ValidateContext.EdittedFormattedValue) > 10)
            {
                e.ValidateContext.ValidateInfo = "Please enter a value which is less than or equal to 10.";
            }
            if (Convert.ToInt32(e.ValidateContext.EdittedFormattedValue) < 1)
            {
                e.ValidateContext.ValidateInfo = "Please enter a value which is greater than 1.";
            }
        }
    }
}

User-Defined Cell Validator

You can create a user-defined cell validator by inheriting the CellValidator ('CellValidator Class' in the on-line documentation) class.

Using Code

The following code implements a cell validator to validate the maximum length of a cell's value.

[VB]
Imports GrapeCity.Win.MultiRow
Public Class MaxLengthValidator
    Inherits CellValidator

    Protected Overrides Function Validate(ByVal context As GrapeCity.Win.MultiRow.ValidateContext) As Boolean
        If context.EditedFormattedValue IsNot Nothing Then
            If context.EditedFormattedValue.ToString().Length > Me.MaxLength Then
                Return False
            End If
        End If
        Return True
    End Function

    Public MaxLength As Integer

        Dim validator As MaxLengthValidator = TryCast(MyBase.Clone(), MaxLengthValidator)
        validator.MaxLength = Me.MaxLength
        Return validator
    End Function

End Class

[CS]
using GrapeCity.Win.MultiRow;

public class MaxLengthValidator : CellValidator
{
protected override bool Validate(ValidateContext context)
{
    if (context.EditedFormattedValue != null)
    {
        if (context.EditedFormattedValue.ToString().Length > this.MaxLength)
            return false;
    }
    return true;
}

public int MaxLength { get; set; }

public override CellValidator Clone()
{
    MaxLengthValidator validator = base.Clone() as MaxLengthValidator;
    validator.MaxLength = this.MaxLength;
    return validator;
}
}

The following code shows an example for a user-defined MaxLengthValidator.

[VB]
Imports GrapeCity.Win.MultiRow
Dim textBoxCell1 As New TextBoxCell
Dim maxLengthValidator1 As New MaxLengthValidator()
maxLengthValidator1.MaxLength = 2
maxLengthValidator1.Actions.Add(New LineNotify())
textBoxCell1.Validators.Add(maxLengthValidator1)
Dim cells As Cell() = {textBoxCell1}
GcMultiRow1.Template = Template.CreateGridTemplate(cells)
GcMultiRow1.RowCount = 10

[CS]
TextBoxCell textBoxCell1 = new TextBoxCell();
MaxLengthValidator maxLengthValidator1 = new MaxLengthValidator();
maxLengthValidator1.MaxLength = 2;
maxLengthValidator1.Actions.Add(new LineNotify());
textBoxCell1.Validators.Add(maxLengthValidator1);
Cell[] cells = { textBoxCell1 };
gcMultiRow1.Template = Template.CreateGridTemplate(cells);
gcMultiRow1.RowCount = 10;

Designer and User-Defined Cell Validator

You can use the designer's CellValidator collection editor to add a user-defined cell validator.

Use the following steps to add the MaxLengthValidator.

1. Create a user-defined cell validator MaxLengthValidator and add it to the project.
2. Build the project.
3. Select the cell for which you wish to validate the value (for example: textBoxCell1).
4. Select the Validators property from the property window, and click the ellipses ... button.
5. Select MaxLengthValidator from the combobox on the top-left of the displayed CellValidator collection editor, and click Add.
6. Select the MaxLength property from the property grid on the right of the screen, and enter 5.
7. Select the Actions property from the property grid on the right of the screen, and click the ellipses ... button.
9. Click the OK button and close the CellValidateAction collection editor.
10. Click the OK button and close the CellValidator collection editor.

User Defined Validation Actions
You can create a user-defined validation action by inheriting the `CellValidateAction` class.

### Using Code

The following code implements a validation action to change the font color to red if there is a validation error.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow
Public Class CustomAction Inherits CellValidateAction
    Protected Overrides Sub DoAction(context As ValidateActionContext)
        If Not context.IsValid Then
        Else
        End If
    End Sub

    Public ForeColor As Color
    Public Overrides Function Clone() As CellValidateAction
        Dim action As CustomAction = TryCast(MyBase.Clone(), CustomAction)
        action.ForeColor = Me.ForeColor
        Return action
    End Function
End Class
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;
public class CustomAction : CellValidateAction
{
    protected override void DoAction(ValidateActionContext context)
    {
        if (!context.IsValid)
        {
        }
        else
        {
        }

    }

    public Color ForeColor
    {
        get;
        set;
    }

    public override CellValidateAction Clone()
    {
        CustomAction action = base.Clone() as CustomAction;
        action.ForeColor = this.ForeColor;
        return action;
    }
}
```

The following code shows an example of a user-defined custom action.
Designer and User Defined Validation Actions

You can use the CellValidateAction Collection Editor of the designer to add a user-defined validation action.
Use the following steps to add the above CustomAction.

1. Create a user-defined validation action that inherits the CellValidateAction class, and add it to the project.
2. Build the project.
3. Select a cell value to validate (for example, numericUpDownCell1).
4. From the **Properties** window, select the **Validators** property and click the **...** button.
5. From the displayed **CellValidator** collection editor, select **RangeValidator** from the left-top combo box and click the **Add** button.
6. From the **Members** list, confirm that RangeValidator has been selected.
7. Select the **MaxValue** property from the property grid and type **10**.
8. Select the **MinValue** property from the property grid and type **0**.
9. Select the **Actions** property from the property grid and click the **...** button.
10. Add **CustomAction** in the displayed **CellValidateAction** collection editor.
11. Set the **ForeColor** property in the property grid to Red.
12. Click the **OK** button to close the **CellValidateAction** collection editor.
13. Click the **OK** button to close the **CellValidator** collection editor.

**Connecting to Database**

The MultiRow control connects to the database using ADO.NET in the same way as the standard .NET Framework controls. The following example uses database products and samples.

- Microsoft SQL Server 2005 or 2008
- Northwind and pubs Sample Databases

**Connection to the Database**

Register the SQL Server database in the project. This is independent of MultiRow.

2. Click **Data - Add New Data Source** in Visual Studio.
3. On the **Data Source Configuration Wizard** page, select **Database** and click **Next**.
4. On the **Choose Your Data Connection** page, click **New Connection** and add a connection with the following configuration.
   - Data source - Microsoft SQL Server (SqlClient)
• Server name (Example) - (local)\SQLEXPRESS
• Select or enter a database name - Northwind

5. After adding the connection, click Next.
6. On the Save the Connection String to the Application Configuration File page, click Next.
7. On the Choose your Database Objects page, check the following tables:
   • Customers
   • Orders

8. Click Finish.
9. Check that the Orders and Customers tables have been added to the Data Sources window of Visual Studio.

Designing Templates
Create the template for MultiRow and assign the fields to the table.

1. Click Project - Add New Item in Visual Studio.
2. On the Add New Item page, select the MultiRow 7.0 Templates and click the Add button.
3. Check that the Template1.vb design page is displayed.
4. Create the headers with the following structure.

   a. Drag ColumnHeaderCell to columnHeaderSection1 from the Toolbox window (repeat this 4 times).
   b. In the Value property of the dragged cells, set ID, CustomerID, OrderDate, and Customer.
   c. Adjust the height of columnHeaderSection1 to be same as the cell height.

5. Create rows with the following structure.

   a. Drag RowHeaderCell to row from the Toolbox window.
   b. Drag TextBoxCell to row from the Toolbox window (repeat twice).
   c. Drag ComboBoxCell to row from the Toolbox window.
   d. In the DataField property of the dragged cells, set OrderID, CustomerID, and OrderDate (the DataField property of ComboBoxCell stays as it is).
   e. Adjust the row height to the same as the cell height.

6. Match the template width to the cell width.
7. Save the project.

Connecting to the Grid
Place the GcMultiRow control on the form and assign the data source and template to the grid.

1. Open the design view of the form (Form1).
2. Select the Orders table from the Data Sources window and click Customize..., from the drop-down list.
3. On the Options page that is displayed, check GcMultiRow in the list of Associated controls.
4. Close the dialog by clicking on the OK button.
5. Select the Orders table in the Data Sources window, and click on GcMultiRow in the drop-down list.
6. Drag and drop the **Orders** table onto the design view of the form (Form1).
7. If GcMultiRow displays a trial version dialog, close it by clicking on the **OK** button.
8. Verify that the BindingNavigator (ordersBindingNavigator) and GcMultiRow (ordersGcMultiRow) have been placed on the form.

9. Click on **Build - Build Solution** in Visual Studio.
10. Drag and drop **Template1** to the form from the Toolbox window.
11. Select the ordersGcMultiRow control on the form, and select **Choose Template** from the smart tag. Select **template11** from the drop-down.

This completes assigning the data source and template to the grid. If you run the project, the datasource values will be read into the grid.

**Setting the Lookup columns**

The CustomerID column values in the Orders table are same as the values of the CustomerID column in the Customers table. Use the values of this CustomerID column as the base and enumerate the values in the CompanyName column of the Customers table.

1. Select the **Customers** table from the **Data Sources** window, and drag and drop to the design screen of the form (Form1).
2. Delete the DataGridView placed on the form.
3. Open the code editor of the form and paste the following code.

```vbnet
Imports GrapeCity.Win.MultiRow
```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Dim myTemplate As Template1 = New Template1()
    Dim comboBoxCell As ComboBoxCell = DirectCast(myTemplate.Row("comboBoxCell1"), ComboBoxCell)
    comboBoxCell.DataSource = Me.customersBindingSource
    comboBoxCell.DisplayMember = "CompanyName"
    comboBoxCell.ValueMember = "CustomerID"
    comboBoxCell.DataField = "CustomerID"
    Me.ordersGcMultiRow.Template = myTemplate

    ' TODO: This code reads the data into 'northwindDataSet.Customers' table. Please move or delete as required.
    Me.customersTableAdapter.Fill(Me.northwindDataSet.Customers)
    ' TODO: This code reads the data into 'northwindDataSet.Orders' table. Please move or delete as required.
    Me.ordersTableAdapter.Fill(Me.northwindDataSet.Orders)
End Sub

using GrapeCity.Win.MultiRow;
private void Form1_Load(object sender, EventArgs e)
{
    Template1 myTemplate = new Template1();
    ComboBoxCell comboBoxCell = myTemplate.Row["comboBoxCell1"] as ComboBoxCell;
    comboBoxCell.DataSource = this.customersBindingSource;
    comboBoxCell.DisplayMember = "CompanyName";
    comboBoxCell.ValueMember = "CustomerID";
    comboBoxCell.DataField = "CustomerID";
    this.ordersGcMultiRow.Template = myTemplate;

    // TODO: This code reads the data into 'northwindDataSet.Customers' table. Please move or delete as required.
    this.customersTableAdapter.Fill(this.northwindDataSet.Customers);
    // TODO: This code reads the data into 'northwindDataSet.Orders' table. Please move or delete as required.
    this.ordersTableAdapter.Fill(this.northwindDataSet.Orders);
}

If you run the project, you can verify that the CompanyName column values have been enumerated into the Customer cells.

You can also verify that changing this value will change the values in the CustomerID cells bound to the same column.

Print and Print Preview

This sections explains the print and print preview features of the control.

- Simple Printing
- Custom Printing
- Print Style
- Paging Mode
- Specify Print Range
- Specifying the Page Break Position
- Specifying Empty Rows for Print
Simple Printing

If you choose to use simple printing, you can easily check printing results without using the .NET Framework print control.

Note that with simple printing, you will not be able to get the result of the user's choice from the dialog. Use the .NET Framework print control to get the dialog result or for customization.

Simple Print

If you execute the GcMultiRow.Print ('Print Method' in the on-line documentation) method without any arguments, the system will use its current settings and print the grid.

Using Code

The code below displays the Print dialog and allows the user to select a printer and then print. You can cancel the print in the middle of the operation.

[VB]

GcMultiRow1.Print()

[CS]

gcMultiRow1.Print();

If you wish to print without displaying a dialog, set False in the first argument.

Using Code

This example prints without displaying a dialog.

[VB]

GcMultiRow1.Print(False)

[CS]

gcMultiRow1.Print(false);

Page Setup

You can use the GcMultiRow.PageSetup ('PageSetup Method' in the on-line documentation) method to display the Page Setup dialog.

Using Code

This example displays the Page Setup dialog.

[VB]

GcMultiRow1.PageSetup()
Display Print Preview Dialog

Use the GcMultiRow.PrintPreview (in the on-line documentation) method to display the grid's print preview dialog based on the system settings.

Using Code

This example uses the PrintPreview method.

[VB]

GcMultiRow1.PrintPreview()

[CS]

gcMultiRow1.PrintPreview();
Custom Printing

You can use the GcMultiRow.Document property to customize printing while using the print control of the .NET Framework.

Page Setup

You can get the user's result if you display the print preview using the PageSetupDialog control.

Using Code

This example uses the PageSetupDialog control.

[VB]

Imports System.Drawing.Printing

Dim document As PrintDocument = New PrintDocument()
GcMultiRow1.Document = document
Using pageSetupDialog As PageSetupDialog = New PageSetupDialog()
    pageSetupDialog.Document = document
    If pageSetupDialog.ShowDialog(Me) = Windows.Forms.DialogResult.OK Then
        Console.WriteLine("OK was clicked.")
    Else
        Console.WriteLine("Cancel was clicked.")
    End If
End Using

[CS]

using System.Drawing.Printing;

PrintDocument document = new PrintDocument();
gcMultiRow1.Document = document;
using (PageSetupDialog pageSetupDialog = new PageSetupDialog())
{
    pageSetupDialog.Document = document;
    if (pageSetupDialog.ShowDialog(this) == DialogResult.OK)
        Console.WriteLine("OK was clicked.");
    else
        Console.WriteLine("Cancel was clicked.");
}

Print Style

MultiRow offers three types of print styles. The styles are rich, compact, and content.

Rich

If you set the PrintSettings.PrintStyle property to Rich, the grid is printed the same as it is displayed on the screen. This is the default style.

Using Code

This example sets the PrintStyle property to Rich.
Imports GrapeCity.Win.MultiRow
GcMultiRow1.PrintSettings.PrintStyle = PrintStyle.Rich
GcMultiRow1.PrintPreview()

using GrapeCity.Win.MultiRow;
gcMultiRow1.PrintSettings.PrintStyle = PrintStyle.Rich;
gcMultiRow1.PrintPreview();

Compact
If you set the PrintSettings.PrintStyle property to Compact, it excludes the screen dependent factors while printing the grid. Specifically, it excludes drop-down buttons or visual styles and creates a form that is applicable for printing.

Using Code
This example sets the PrintStyle property to Compact.

[VB]
Imports GrapeCity.Win.MultiRow
GcMultiRow1.PrintSettings.PrintStyle = PrintStyle.Compact
GcMultiRow1.PrintPreview()

[CS]
using GrapeCity.Win.MultiRow;
gcMultiRow1.PrintSettings.PrintStyle = PrintStyle.Compact;
gcMultiRow1.PrintPreview();

Content
If you set the PrintSettings.PrintStyle property to Content, it only prints strings and images. Cell borders, backcolors, and back gradation are not printed.

Using Code
This example sets the PrintStyle property to Content.

[VB]
Imports GrapeCity.Win.MultiRow
GcMultiRow1.PrintSettings.PrintStyle = PrintStyle.Content
GcMultiRow1.PrintPreview()

[CS]
using GrapeCity.Win.MultiRow;

gcMultiRow1.PrintSettings.PrintStyle = PrintStyle.Content;
gcMultiRow1.PrintPreview();

## Differences with Print Styles

<table>
<thead>
<tr>
<th><strong>Cell Type</strong></th>
<th><strong>Rich</strong></th>
<th><strong>Compact and Content</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>ColumnHeaderCell</td>
<td>Drop-down buttons for built-in sorting and built-in filtering are not printed (common to both).</td>
<td></td>
</tr>
<tr>
<td>RowHeaderCell</td>
<td>Indicator is not printed (common to both).</td>
<td></td>
</tr>
<tr>
<td>LinkLabelCell</td>
<td>-</td>
<td>Link underline is not printed.</td>
</tr>
<tr>
<td>NumericUpDownCell</td>
<td>-</td>
<td>Spin button is not printed.</td>
</tr>
<tr>
<td>ComboBoxCell</td>
<td>-</td>
<td>Drop-down button is not printed.</td>
</tr>
<tr>
<td>DateTimePickerCell</td>
<td>-</td>
<td>Spin and drop-down buttons are not printed.</td>
</tr>
<tr>
<td>CheckBoxCell</td>
<td>-</td>
<td>Visual Style is not applied.</td>
</tr>
<tr>
<td>RadioGroupCell</td>
<td>-</td>
<td>Visual Style is not applied.</td>
</tr>
<tr>
<td>ProgressBarCell</td>
<td>-</td>
<td>Visual Style is not applied.</td>
</tr>
</tbody>
</table>

## Paging Mode

In the MultiRow control, you can allocate rows to pages when printing.

### Flow Mode

Rows are arranged based on the page height and are printed sequentially. This is the default mode. You can enable flow mode by setting the PrintSettings.**PagingMode (PagingMode Property' in the on-line documentation)** property to Flow.

Using Code

This example sets the PagingMode property to Flow.

**[VB]**

```
```

**[CS]**

```
```

## Single Row Mode

This mode prints only one row per page and is useful for treating one row as one detail. You can enable single row mode...
by setting the PrintSettings.PagingMode property to SingleRow.

![Image of SingleRow paging mode]

**Using Code**

This example sets the PagingMode property to SingleRow.

**[VB]**

```vbnet
```

**[CS]**

```csharp
```

**Multi Column Mode**

This mode prints rows horizontally with the same size. The rows are placed in the shape of an N character from top to bottom and left to right. This mode is applied to print the maximum number of rows in a single page. You can enable multiple column mode by setting the PrintSettings.PagingMode property to MultiColumns.

![Image of MultiColumn paging mode]

**Using Code**

This example sets the PagingMode property to MultiColumns.

**[VB]**

```vbnet
```

**[CS]**

```csharp
```

You can set the column margin with the PrintSettings.ColumnMargin (["ColumnMargin Property" in the on-line documentation]) property.

**Specify Print Range**

In MultiRow, you can specify a print range for every single row.

**Print All Rows**

You can print all the displayed rows using the MultiRowPrintRange.AllRows option in the PrintSettings.PrintRange (["PrintRange Property" in the on-line documentation]) property.
Using Code

This example sets the PrintRange (PrintRange Property' in the on-line documentation) property to AllRows.

[VB]

[CS]


Print Rows Selected by the User

You can print all the rows selected by the user with the MultiRowPrintRange.SelectedRows option in the PrintSettings.PrintRange (PrintRange Property' in the on-line documentation) property.

Print Selected Rows

You can print the range of rows specified by the StartRow and EndRow options by setting the PrintSettings.PrintRange (PrintRange Property' in the on-line documentation) property to MultiRowPrintRange.SomeRows. The StartRow option can be set using PrintSettings.FromRow (FromRow Property' in the on-line documentation) and the EndRow option can be set with PrintSettings.ToRow (ToRow Property' in the on-line documentation).

Print Current Row

You can print the current row (GcMultiRow.CurrentCellPosition.RowIndex) only by setting the PrintSettings.PrintRange (PrintRange Property' in the on-line documentation) property to MultiRowPrintRange.CurrentRow.

Specifying the Page Break Position

In MultiRow, there are two ways of setting a page break at any position, one is using the automatic page break and the other is with the Cell.PageBreak (PageBreak Property' in the on-line documentation) property. You can specify a page break position for every single row in MultiRow.

Setting a Page Break Position in the Horizontal Direction

You can set a page break position in the horizontal direction with the the Cell.PageBreak property.

Using Code

This example sets the PageBreak property.

[VB]
GcMultiRow1.Rows(0).Cells(2).PageBreak = True

[CS]

gcMultiRow1.Rows[0].Cells[2].PageBreak = true;

Setting a Page Break Position in the Vertical Direction

You can set a page break position in the vertical direction with the Row.PageBreak (PageBreak Property' in the on-line documentation) property.

Using Code

This example sets the PageBreak property.

[VB]
GcMultiRow1.Rows(2).PageBreak = True

[CS]

gcMultiRow1.Rows[2].PageBreak = true;
Displaying the Page Break Preview

You can use the GcMultiRow.ShowPageBreaks ('ShowPageBreaks Property' in the on-line documentation) property to display the page break preview.

Using Code

This example sets the ShowPageBreaks property.

[VB]
GcMultiRow1.ShowPageBreaks = True

[CS]
gcMultiRow1.ShowPageBreaks = true;

Setting the Border Style of the Page Break Preview

You can set the border style of each page break with the GcMultiRow.PageBreakLines ('PageBreakLines Property' in the on-line documentation) property.

Using Code

This example sets the PageBreakLines property.

[VB]
Imports GrapeCity.Win.MultiRow

[CS]
using GrapeCity.Win.MultiRow;

Specifying Empty Rows for Print

MultiRow can print empty rows for the blank spaces in the page that you would like to print.

If the PrintSettings.PagingMode ('PagingMode Property' in the on-line documentation) property is set to SingleRow, only one row is printed on each page, regardless of the PrintSettings.RepeatToFill ('RepeatToFill Property' in the on-line documentation) property setting.

Printing Empty Rows

You can use the PrintSettings.RepeatToFill ('RepeatToFill Property' in the on-line documentation) property to print empty rows.

Using Code

This example sets the RepeatToFill ('RepeatToFill Property' in the on-line documentation) property.

[VB]
GcMultiRow1.PrintSettings.RepeatToFill = True

[CS]
gcMultiRow1.PrintSettings.RepeatToFill = true;

Printing Watermarks

You can print data with a watermark inserted in its background in MultiRow.

Setting the Watermark
You can set the `PrintSettings.Watermark` ("Watermark Property' in the on-line documentation) property to image type data for setting the watermark.

**Using Code**

This example sets the Watermark property.

**[VB]**

```vbnet
GcMultiRow1.PrintSettings.Watermark = New Bitmap("test.bmp")
```

**[CS]**

```csharp
gcMultiRow1.PrintSettings.Watermark = new Bitmap(@"test.bmp");
```

**Aligning the Watermark**

You can use the `PrintSettings.WatermarkAlignment` ("WatermarkAlignment Property' in the on-line documentation) property to set the location of the watermark.

**Using Code**

This example sets the WatermarkAlignment property.

**[VB]**

```vbnet
GcMultiRow1.PrintSettings.WatermarkAlignment = ContentAlignment.BottomRight
```

**[CS]**

```csharp
gcMultiRow1.PrintSettings.WatermarkAlignment = ContentAlignment.BottomRight;
```

**Size of the Watermark**

You can use the `PrintSettings.WatermarkSizeMode` ("WatermarkSizeMode Property’ in the on-line documentation) property to set the size of the watermark.

**Using Code**

This example sets the WatermarkSizeMode property.

**[VB]**

```vbnet
```

**[CS]**

```csharp
```

**Specifying Pages for Printing the Watermark**

You can use the `PrintSettings.WatermarkPrintOnPages` ("WatermarkPrintOnPages Property' in the on-line documentation) property to set the page(s) on which you want to print the watermark.

**Using Code**

This example sets the WatermarkPrintOnPages property.

**[VB]**
```csharp
GcMultiRow1.PrintSettings.WatermarkPrintOnPages = "3-5";
```

**Opacity of the Watermark**

You can use the `PrintSettings.WatermarkOpacity` property to set the opacity of the watermark.

**Using Code**

This example sets the `WatermarkOpacity` property.

```vbnet
GcMultiRow1.PrintSettings.WatermarkOpacity = 0.5
```

```csharp
gcMultiRow1.PrintSettings.WatermarkOpacity = 0.5;
```

**Displaying the Watermark in the Foreground**

By default, the watermark that overlaps the grid is not printed, but if the `Section.PrintBackground` property is set to False, you can print with the watermark displayed in the foreground.

```
<table>
<thead>
<tr>
<th>OrderID</th>
<th>CustomerID</th>
<th>OrderDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>10248</td>
<td>VINET</td>
<td>7/4/1996</td>
</tr>
<tr>
<td>10249</td>
<td>TOMSP</td>
<td>7/5/1996</td>
</tr>
<tr>
<td>10250</td>
<td>HANAR</td>
<td>7/8/1996</td>
</tr>
<tr>
<td>10251</td>
<td>VICTE</td>
<td>7/8/1996</td>
</tr>
<tr>
<td>10252</td>
<td>SUPRO</td>
<td>7/8/1996</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>OrderID</th>
<th>CustomerID</th>
<th>OrderDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>10248</td>
<td>VINET</td>
<td>7/4/1996</td>
</tr>
<tr>
<td>10249</td>
<td>TOMSP</td>
<td>7/5/1996</td>
</tr>
<tr>
<td>10250</td>
<td>HANAR</td>
<td>7/8/1996</td>
</tr>
<tr>
<td>10251</td>
<td>VICTE</td>
<td>7/8/1996</td>
</tr>
<tr>
<td>10252</td>
<td>SUPRO</td>
<td>7/8/1996</td>
</tr>
</tbody>
</table>
```

**Using Code**

This example prints the watermark.

```vbnet
Imports GrapeCity.Win.MultiRow
Dim textBoxCell1 As New TextBoxCell() textBoxCell1.Name = "textBoxCell1"
Dim textBoxCell2 As New TextBoxCell() textBoxCell2.Name = "textBoxCell2"
Dim template As Template = template.CreateGridTemplate(New Cell() {textBoxCell1, textBoxCell2})
template.Row.PrintBackground = False
GcMultiRow1.Template = template GcMultiRow1.RowCount = 10
```

```csharp
MultiRow Windows Forms Developer's Guide 429
```

Copyright © GrapeCity, inc. All rights reserved.
Print Settings for each Column Header Section

In MultiRow, you can set the printing method for each section of the column footer and the column header.

**Specifying the Print Pages of the Column Header Section**

You can use the ColumnHeaderSection.PrintPagesMode (‘PrintPagesMode Property' in the on-line documentation) property to set the printing method for each column header section.

**Using Code**

This example sets properties for column headers.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow
Dim template = New Template1()
template.ColumnHeader(0).PrintPagesMode = PrintHeader.AllPages
template.ColumnHeader(1).PrintPagesMode = PrintHeader.FirstPage
gcMultiRow1.Template = template
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;
Template template = new Template1();
template.ColumnHeader[0].PrintPagesMode = PrintHeader.AllPages;
template.ColumnHeader[1].PrintPagesMode = PrintHeader.FirstPage;
gcMultiRow1.Template = template;
```

**Specifying the Print Pages of the Column Footer Section**

You can use the ColumnFooterSection.PrintPagesMode (‘PrintPagesMode Property' in the on-line documentation) property to set the printing method for each column footer section.

**Using Code**

This example sets properties for column footers.

**[VB]**

```vbnet
Imports GrapeCity.Win.MultiRow
Dim template = New Template1()
template.ColumnHeader(0).PrintPagesMode = PrintFooter.AllPages
template.ColumnHeader(1).PrintPagesMode = PrintFooter.LastPage
gcMultiRow1.Template = template
```

**[CS]**

```csharp
using GrapeCity.Win.MultiRow;
Template template = new Template1();
template.ColumnHeader[0].PrintPagesMode = PrintFooter.AllPages;
template.ColumnHeader[1].PrintPagesMode = PrintFooter.LastPage;
gcMultiRow1.Template = template;
```
[CS]

```csharp
using GrapeCity.Win.MultiRow;
Template template = new Template1();
template.ColumnFooters[0].PrintPagesMode = PrintFooter.AllPages;
template.ColumnFooters[1].PrintPagesMode = PrintFooter.LastPage;
gcMultiRow1.Template = template;
```

File Input and Output

You can store and load templates as XML files.

- **Input and Output of Template XML**

Input and Output of Template XML

MultiRow can store and load templates as XML files. Since the XML file is a text file, it can be viewed using a text editor or an XML editor and the changes or differences can be checked directly.

- Loading an XML file created or modified using a tool other than MultiRow is not supported.
- Information other than cell layout is not stored in the XML file.

Designer and XML File

The template can be saved to an XML file while it is in an edit state in the designer. You can load an XML file that was saved with the Template.**Save ('Save Method' in the on-line documentation)** method. Use the following steps to save templates being edited in the designer as XML files.

- Select menu **Template - Save Layout**.

Use the following steps to load a template from an XML file in the designer.

- Select menu **Template - Load Layout**.

Output of XML File

Use the Template.**Save ('Save Method' in the on-line documentation)** method to save the displayed contents of the GcMultiRow control as an XML file.

> If there is a file with the same name as that of the file specified with the Template.**Save ('Save Method' in the on-line documentation)** method, the existing file is overwritten.

Using Code

This example saves a file.

-[VB]

```vb
GcMultiRow1.Template.Save("filename.xml")
```

-[CS]

```csharp
gcMultiRow1.Template.Save(@"filename.xml");
```

Input in XML File
The Template.Load ('Load Method' in the on-line documentation) method is used to load an XML file as a template of the GcMultiRow control.

The System.FormatException exception is thrown if an XML file with an incorrect format is specified.

Using Code

This example loads a file.

[VB]

GcMultiRow1.Template.Load("filename.xml")

[CS]

gcMultiRow1.Template.Load(@"filename.xml");

Multi-touch Features

MultiRow supports multi-touch features on touch devices. Basic touch operations such as input using the touch keyboard and cell selection, spin, drop-down, scrolling the grid, and so on, are possible.

Additional features and user interfaces suitable for touch are available in MultiRow, as described in the following sections.

- Zooming
- Scrolling
- Cell Selection
- Resizing the Column Width
- Touch Toolbar
- Touch Keyboard
- Gripper during Cell Editing
- Zoomed Display of the Drop-down List
- Retrieving the Input Device

Zooming

In MultiRow, you can zoom the contents of the grid using the stretch and pinch gesture. This section describes the feature to zoom the grid, using touch.
Setting the Minimum Zoom Percentage

You can use the `MinZoomFactor` property to set the minimum zoom percentage. If you zoom to less than the minimum zoom percentage using the pinch gesture, the screen reverts back to the zoom percentage that is set in the `MinZoomFactor` property.

Using Code
The following code sets the minimum zoom percentage to 100%.

**[VB]**

```vbnet
GcMultiRow1.MinZoomFactor = 1.0F
```

**[CS]**

```csharp
gcMultiRow1.MinZoomFactor = 1.0F;
```

### Adding a Snap Point

You can use the `TouchZoomSnapPoints` (‘TouchZoomSnapPoints Property’ in the on-line documentation) property to set the snap point of the zoom. If the snap point is set, the zoom percentage is automatically set to the nearest snap point when you interrupt the stretch or pinch operation by removing your finger from the screen.

### Using Designer

Use the following steps to add a snap point.

1. Select the GcMultiRow control (for example, gcMultiRow1).
2. From the Properties window, select the TouchZoomSnapPoints property and click the ... button.
3. In the Single Collection Editor, click the Add button.
4. Select the item added to the members, and add any value in the range of 0-4 to the Value property in the property grid.
5. Click the OK button and close the Single Collection Editor.

You can also add a snap point with code. You can use the Add or AddRange method to add a snap point.

### Using Code

This example uses the Add or AddRange method to add a snap point.

**[VB]**
'Add a snap point of zoom percentage, using the Add method.
GcMultiRow1.TouchZoomSnapPoints.Add(1.5F)
GcMultiRow1.TouchZoomSnapPoints.Add(2.0F)
GcMultiRow1.TouchZoomSnapPoints.Add(2.5F)
'Add a zoom percentage snap point, using the AddRange method.
GcMultiRow1.TouchZoomSnapPoints.AddRange(New Single() {3.0F, 3.5F})

[CS]

// Add a snap point of zoom percentage, using the Add method.
gcMultiRow1.TouchZoomSnapPoints.Add(1.5F);
gcMultiRow1.TouchZoomSnapPoints.Add(2F);
gcMultiRow1.TouchZoomSnapPoints.Add(2.5F);
// Add a zoom percentage snap point, using the AddRange method.
gcMultiRow1.TouchZoomSnapPoints.AddRange(new float[] { 3F, 3.5F });

You need to zoom to a value greater than the value set in the TouchZoomSnapDistance (‘TouchZoomSnapDistance Property’ in the on-line documentation) property, so that the zoom percentage is automatically set and uses the snap point. For example, if the TouchZoomSnapDistance property is set to 1, the zoom percentage is not set automatically to the snap point, unless you change the zoom percentage to 100% or more, by a single stretch or pinch operation.

Using Code

This example sets the TouchZoomSnapDistance property.

[VB]

' The zoom percentage is automatically set when the 'zoom percentage is changed to 50% or more.
GcMultiRow1.TouchZoomSnapDistance = 0.5F

[CS]

// The zoom percentage is automatically set when the //zoom percentage is changed to 50% or more.
gcMultiRow1.TouchZoomSnapDistance = 0.5F;

Disabling Zoom

Setting the AllowUserToTouchZoom ('AllowUserToTouchZoom Property' in the on-line documentation) property to False disables zooming using the touch operation. By disabling the zoom operation using touch, you can prevent accidental zooming while scrolling or other operations.

Using Code

This example sets the AllowUserToTouchZoom property.
Scrolling

In MultiRow, you can scroll the screen by sliding your finger on the grid. This section describes the feature to scroll the grid using touch gestures.

Setting the Scroll Direction

You can set the direction in which you can scroll using touch with the **PanningMode** (‘PanningMode Property’ in the on-line documentation) property.

### PanningMode

<table>
<thead>
<tr>
<th>PanningMode property</th>
<th>None</th>
<th>HorizontalOnly</th>
<th>VerticalOnly</th>
<th>HorizontalOrVertical</th>
<th>Both</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scroll direction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Using Code

This example sets the PanningMode property.

### [VB]

```vbnet
```

### [CS]

```csharp
```

Boundary Behavior

The **BoundaryFeedbackMode** (‘BoundaryFeedbackMode Property’ in the on-line documentation) property sets the operation when the scroll has reached the edge of the screen. **Split** shows a split visual feedback of the row section region (region except the column header section and the column footer section), and **Standard** causes the parent window containing the GcMultiRow control to have a visual feedback.

### BoundaryFeedbackMode property

<table>
<thead>
<tr>
<th>Boundary Feedback Mode property</th>
<th>Standard</th>
<th>Split</th>
</tr>
</thead>
</table>

Using Code

This example demonstrates the BoundaryFeedbackMode property.

### [VB]

```vbnet
```

### [CS]

```csharp
```
This example sets the BoundaryFeedbackMode property.

[VB]
```
```

[CS]
```
```

**Scrolling Behavior**

If the `HorizontalScrollMode` (`'HorizontalScrollMode Property' in the on-line documentation`) property is set to Cell, the scroll by touch operation moves in pixels, even if the `VerticalScrollMode` (`'VerticalScrollMode Property' in the on-line documentation`) property is set to Row. In this case, when you release your finger from the screen, the scroll position moves such that all the cells displayed in the top-left are displayed.

**Scrolling using the Scroll Button**

In MultiRow, you can scroll the screen by pressing the scroll button. In addition, you can also scroll by sliding the scroll bar.

**Cell Selection**

MultiRow displays a frame border around the selected cells and a gripper (round selection handle) when you select cells using touch. You can use the gripper to change the selected cell range. This section describes the way to select cells using touch gestures.

**Selection Method**

MultiRow displays a frame border around the selected cells, along with a gripper, when you select cells using touch.

You can change the range of selected cells by sliding (move your finger while touching) the gripper after selecting the cells.

Depending on the layout of the template, unselected cells may be included in the frame of the selected range; however, if a cell which is not in the selected state is included in the frame of the selected range, the cell is still treated as an unselected cell.

For example, in the figure below, Cell8 and Cell15 in Case1 are unselected, and Cell8 is unselected in Case2.
In addition, you can select individual columns by tapping the column header cell. In this case, the gripper is displayed in the center of the display area.

Style of the Frame Border and Gripper

You can set the color and line type of the gripper and the frame border, respectively. You can set the style using the `TouchSelectionBorderLine` ('TouchSelectionBorderLine Property' in the on-line documentation) property for the frame border, and the `TouchSelectionGripperLine` ('TouchSelectionGripperLine Property' in the on-line documentation) property for the gripper.

In addition, the inside color of the circle for the gripper can be set using the `TouchSelectionGripperBackColor` ('TouchSelectionGripperBackColor Property' in the on-line documentation) property.

If nothing is set in the `TouchSelectionGripperBackColor` property, the color set in the `BackColor` property of the `DefaultCellStyle` ('DefaultCellStyle Property' in the on-line documentation) property is applied as the background color of the gripper.

Using Code

This example adds and sets colors for the selection border and gripper line.

[VB]

GcMultiRow1.TouchSelectionGripperBackColor = Color.Yellow

[CS]

gcMultiRow1.TouchSelectionGripperBackColor = Color.Yellow;

Hiding the Gripper and the Frame Border

You can hide the gripper and the frame border by setting the `UseOptimizedSelectionForTouch` ('UseOptimizedSelectionForTouch Property' in the on-line documentation) property.
documentation) property to False. After choosing this setting, you can no longer change the selected range of cells by sliding the gripper.

Using Code

This example sets the UseOptimizedSelectionForTouch ('UseOptimizedSelectionForTouch Property' in the on-line documentation) property.

[VB]
GcMultiRow1.UseOptimizedSelectionForTouch = False

[CS]
gcMultiRow1.UseOptimizedSelectionForTouch = false

Resizing the Column Width

MultiRow allows you to resize the column width using touch. This section describes how to resize the column width using the touch gesture.

Mark for Resizing

The resizing mark is displayed at the right or bottom of the cell when you tap on it, if the Cell.ResizeMode ('ResizeMode Property' in the on-line documentation) property for the cells is set to a value other than None.

The following figure shows the difference between the resizing mark based on the value set in the Cell.ResizeMode ('ResizeMode Property' in the on-line documentation) property.

<table>
<thead>
<tr>
<th>ResizeMode property</th>
<th>Horizontal</th>
<th>Vertical</th>
<th>Both</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display of the Resize Mark</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The following is a description about the column width, but the same feature is also available for resizing the row height.

Resize by Sliding

You can change the column width by sliding the mark.

Resizing by Double-tap

Double-tap the resizing mark to perform automatic adjustment of the cell width to fit the contents of the corresponding cell.

In addition, you can perform automatic adjustment of the width for all the cells collectively by tapping the CornerHeaderCell, after double-tapping the resizing mark.

Disable Resizing

You can disable resizing with touch, by setting the AllowUserToTouchResize ('AllowUserToTouchResize Property' in the on-line documentation) property to False. After this setting, the mark for resizing does not appear even if you tap the cell.

Using Code

This example sets the AllowUserToTouchResize ('AllowUserToTouchResize Property' in the on-line documentation) property.

[VB]
GcMultiRow1.AllowUserToTouchResize = False

[CS]
gcMultiRow1.AllowUserToTouchResize = false
Touch Toolbar

MultiRow provides a **Touch Toolbar** for performing operations such as copying and pasting of selected cells using touch gestures. This section describes how to operate and customize the touch toolbar.

**Touch Toolbar**

The touch toolbar has a structure where the **TouchToolBarButton** (**TouchToolBarButton** Class’ in the on-line documentation) object is placed in the **MultiRowTouchToolBar** (**MultiRowTouchToolBar** Class’ in the on-line documentation) object, and you can assign an action to TouchToolBarButton. Since it is possible to customize the touch toolbar, you can perform various operations using the touch toolbar on the MultiRow control.

In MultiRow, a default touch toolbar allows you to perform operations such as cut, copy, paste, and delete, so you can use the touch toolbar without any configuration.

---

**How to Display**

The touch toolbar can be displayed by tapping, or pressing the selected cells.

- The touch toolbar is only displayed on a touch-enabled device.
- The touch toolbar is displayed only when a touch operation is performed. It is not displayed at the time of a mouse operation.

**Tapping Selected Cells**

1. **Display of Touch Toolbar**
   
   In a state when the gripper and the frame border surrounding the selected cells are displayed, the touch toolbar is displayed if you tap inside the frame border.

   ![Diagram of touch toolbar display](image)

   When there is a blank area included in the frame border of selected cells, the touch toolbar is displayed even if you tap on that blank area.
2. Hiding the touch toolbar
In a state where some cells are selected, and the touch toolbar is displayed, the touch toolbar can be hidden by tapping inside the grid. At that time, the state of the selected cells is retained.

If you tap inside the grid in a state where all the rows of the grid are selected and the touch toolbar is being displayed, the touch toolbar is hidden and the tapped cell is selected.

Display by Long Press
In MultiRow, if you press a cell, the cell is selected and the touch toolbar is displayed.

1. Selected Cells
If you press inside the frame border of the selected cells, the touch toolbar is displayed, and the state of the selected cells is retained.
2. Cells Outside the Selected Range
If you press a cell outside the frame border of the selected cells, the touch toolbar is displayed, and the pressed cell is selected.

Disabling the Touch Toolbar
Since, the default touch toolbar is set in the GcMultiRow.TouchToolBar (TouchToolBar Property in the on-line documentation) property of the GcMultiRow control, the default touch toolbar is displayed if no settings are performed.

If you do not want the default touch toolbar to be displayed, you can set the TouchToolBar property to Null.

Using Code
This example sets the TouchToolBar property.

[VB]
GcMultiRow1.TouchToolBar = Nothing

[CS]
gcMultiRow1.TouchToolBar = null;

Customizing the Default Touch Toolbar
The default touch toolbar has four items (cut, copy, paste, and delete) set in it, but you can hide these items, and also add new items to it.

Using Code
The following code creates an instance of the default touch toolbar, and assigns a touch toolbar to the GcMultiRow control.

[VB]
Dim defaultTouchToolBar1 = New DefaultTouchToolBar(GcMultiRow1)
' Hide Cut of the default touch toolbar
defaultTouchToolBar1.Items(1).Visible = False
' Set the touch toolbar to the GcMultiRow control
GcMultiRow1.TouchToolBar = defaultTouchToolBar1

[CS]
DefaultTouchToolBar defaultTouchToolBar1 = new DefaultTouchToolBar(gcMultiRow1);
// Hide Cut of the default touch toolbar
defaultTouchToolBar1.Items[1].Visible = false;
// Set the touch toolbar to the GcMultiRow control
gcMultiRow1.TouchToolBar = defaultTouchToolBar1;

The following codes uses the RemoveAt method to remove an item from the default touch toolbar.

[VB]
Dim defaultTouchToolBar1 = New DefaultTouchToolBar(GcMultiRow1)
' Remove Copy of the default toolbar
defaultTouchToolBar1.Items.RemoveAt(2)
' Set the touch toolbar to the GcMultiRow control
GcMultiRow1.TouchToolBar = defaultTouchToolBar1

In addition, you can use the Insert method to add a new item to the default touch toolbar. This example adds a new item to the toolbar.

[C#]
DefaultTouchToolBar defaultTouchToolBar1 = new DefaultTouchToolBar(gcMultiRow1);
// Add an item to the default toolbar
DefaultTouchToolBarItems.Add(selectallBtn);
// Set the touch toolbar to the GcMultiRow control
GcMultiRow1.TouchToolBar = defaultTouchToolBar1;

Customizing the Touch Toolbar

You can use the MultiRowTouchToolBar ('MultiRowTouchToolBar Class' in the on-line documentation) class to create a new touch toolbar, without using the default toolbar.

Using Code

The following code creates a touch toolbar on which Copy and Paste are placed.

[VB]
Imports GrapeCity.Win.MultiRow
GcMultiRow1.Template = Template.CreateGridTemplate(5)
GcMultiRow1.RowCount = 6
Dim touchToolBar As MultiRowTouchToolBar = New MultiRowTouchToolBar(GcMultiRow1)
.Image = New Bitmap("Touch_Copy.png")
.Image = New Bitmap("Touch_Paste.png")
} touchToolBar.Items.AddRange(New ToolStripItem() {copyBtn, pasteBtn})
GcMultiRow1.TouchToolBar = touchToolBar

[CS]
using GrapeCity.Win.MultiRow;
GcMultiRow1.Template = Template.CreateGridTemplate(5);
You can process operations of your own using the Click event of TouchToolBarButton.

Using Code

The following code performs sorting in the touch toolbar.

[VB]

Imports GrapeCity.Win.MultiRow
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    GcMultiRow1.Template = Template.CreateGridTemplate(3)
    GcMultiRow1.AllowUserToAddRows = False
    GcMultiRow1.RowCount = 5
    ' Set the sample data
    GcMultiRow1.Rows(0).SetValues(New String(2) {"E", "1", "VB")
    GcMultiRow1.Rows(1).SetValues(New String(2) {"D", "2", "VB")
    GcMultiRow1.Rows(2).SetValues(New String(2) {"C", "3", "CS")
    GcMultiRow1.Rows(3).SetValues(New String(2) {"B", "4", "VB")
    GcMultiRow1.Rows(4).SetValues(New String(2) {"A", "5", "CS")
    ' Set the touch toolbar
    Dim touchToolBar = New MultiRowTouchToolBar(GcMultiRow1)
    ' Set the button click event of the touch toolbar
    AddHandler ascSortBtn.Click, AddressOf ascSortBtn_Click
    AddHandler desSortBtn.Click, AddressOf desSortBtn_Click
    touchToolBar.Items.AddRange(New ToolStripItem() {ascSortBtn, desSortBtn})
    GcMultiRow1.TouchToolBar = touchToolBar
End Sub

Private Sub ascSortBtn_Click(sender As Object, e As EventArgs)
    ' Sort in ascending order, with reference to the current cell.
End Sub

Private Sub desSortBtn_Click(sender As Object, e As EventArgs)
    ' Sort in descending order, with reference to the current cell.
End Sub

[CS]

using GrapeCity.Win.MultiRow;
private void Form1_Load(object sender, EventArgs e)
{
    gcMultiRow1.Template = Template.CreateGridTemplate(3);
    gcMultiRow1.AllowUserToAddRows = false;
    gcMultiRow1.RowCount = 5;
    // Set the sample data
    gcMultiRow1.Rows[0].SetValues(new string[3] {"E", "1", "VB")
    // Set the touch toolbar
    MultiRowTouchToolBar touchToolBar = new MultiRowTouchToolBar(gcMultiRow1);
    TouchToolBarButton ascSortBtn = new TouchToolBarButton() { Text = "Sort in Ascending Order", Name = "Sort", Image = new Bitmap("Touch_AscSort.png")};
    // Set the button click event of the touch toolbar
    ascSortBtn.Click += ascSortBtn_Click;
    desSortBtn.Click += desSortBtn_Click;
    touchToolBar.Items.AddRange(new ToolStripItem[] {ascSortBtn, desSortBtn});
    gcMultiRow1.TouchToolBar = touchToolBar;
}
private void ascSortBtn_Click(object sender, EventArgs e)
{
    // Sort in ascending order, with reference to the current cell.
    gcMultiRow1.Sort(gcMultiRow1.CurrentCell.CellIndex, SortOrder.Ascending);
}
private void desSortBtn_Click(object sender, EventArgs e)
{
    // Sort in descending order, with reference to the current cell
    gcMultiRow1.Sort(gcMultiRow1.CurrentCell.CellIndex, SortOrder.Descending);
}

Using Context Menu in the Touch Toolbar

You can use a context menu in the touch toolbar, using the `ContextMenuStrip` class and `ToolStripDropDownButton` class of the .NET Framework.

Using Code

The following code sets a context menu in the touch toolbar.

```
Imports GrapeCity.Win.MultiRow
Dim touchToolBar = New MultiRowTouchToolBar(gcMultiRow1)
Dim dropDownMenu = New ToolStripDropDownButton()
dropDownMenu.Image = New Bitmap("Touch_dropdown.png")
dropDownMenu.ShowDropDownArrow = False
dropDownMenu.ImageScaling = ToolStripItemImageScaling.None
' Set the context menu
menu.Items.Add("Add Row") menu.Items.Add("Delete Row")
menu.Items.Add("Select All")
dropDownMenu.DropDown = menu
touchToolBar.Items.AddRange(New ToolStripItem[] {dropDownMenu})
gcMultiRow1.TouchToolBar = touchToolBar
```

You can set the space between items in the context menu, by setting the `ContextMenuStrip.Items.AutoSize` property to False and setting the `Height` property, so that it is easy to operate through touch.

Using Code

This example sets the menu height.

```
' Set the context menu
menu.Items.Add("Add Row")
menu.Items.Add("Delete Row")
menu.Items.Add("Select All")
dropDownMenu.DropDown = menu;
touchToolBar.Items.AddRange(new ToolStripItem[] { dropDownMenu });
gcMultiRow1.TouchToolBar = touchToolBar;
```
Because the context menu used in the touch toolbar uses the features of ContextMenuStrip and ToolStripDropDownButton of the .NET Framework, it is not a feature provided by MultiRow.

Using the Touch Toolbar in Cell Notes

The touch toolbar can also be used in Cell Notes.

In a cell note, you can display the touch toolbar by any of the following actions:

- Press the selected cell note when it is not in edit mode.
- Press a cell note which is not selected.

If you want to use the same touch toolbar in every cell note, you can use the TouchToolBar (‘TouchToolBar Property’ in the on-line documentation) property of the GcMultiRow.DefaultCellNoteStyle (‘DefaultCellNoteStyle Property’ in the on-line documentation) property.

Using Code

This example adds a button to the touch toolbar.

[VB]

Imports GrapeCity.Win.MultiRow
Dim touchToolBar = New MultiRowTouchToolBar(GcMultiRow1)
' Set a touch toolbar button
Dim noteDeleteBtn = New TouchToolBarButton() With {.Text = "Delete", .Name = "NoteDelete",
.Image = New Bitmap("Touch_NoteDel.png")}
' Add the button to the touch toolbar
touchToolBar.Items.AddRange(new ToolStripItem[] {noteDeleteBtn})
GcMultiRow1.DefaultCellNoteStyle.TouchToolBar = touchToolBar

[CS]

using GrapeCity.Win.MultiRow;
MultiRowTouchToolBar touchToolBar = new MultiRowTouchToolBar(gcMultiRow1);
// Set a touch toolbar button
TouchToolBarButton noteDeleteBtn = new TouchToolBarButton() { Text = "Delete", Name = "NoteDelete",
.Image = new Bitmap("Touch_NoteDel.png") };
// Add the button to the touch toolbar
touchToolBar.Items.AddRange(new ToolStripItem[] {noteDeleteBtn});
gcMultiRow1.DefaultCellNoteStyle.TouchToolBar = touchToolBar;

If you want to use the touch toolbar in a specific cell note, you can use the TouchToolBar (‘TouchToolBar Property’ in the on-line documentation) property of the CellNote.Style (‘Style Property’ in the on-line documentation) property as in this example.

[VB]

Imports GrapeCity.Win.MultiRow
Dim touchToolBar = New MultiRowTouchToolBar(GcMultiRow1)
' Set a touch toolbar button
.Image = New Bitmap("Touch_NoteEdit.png")}
' Add the button to the touch toolbar
TouchToolBar.Items.AddRange(New ToolStripItem() {notestyleBtn})
' Assign the touch toolbar to a specific cell note
GcMultiRow1(5, 0).Note.Style.TouchToolBar = touchToolBar

using GrapeCity.Win.MultiRow;
MultiRowTouchToolBar touchToolBar = new MultiRowTouchToolBar(gcMultiRow1);
// Set a touch toolbar button
TouchToolBarButton notestyleBtn = new TouchToolBarButton() { Text = "Edit", Name = "NoteEdit",
Image = new Bitmap(@"Touch_NoteEdit.png")};
// Add the button to the touch toolbar
touchToolBar.Items.AddRange(new ToolStripItem[] { notestyleBtn });
// Assign the touch toolbar to a specific cell note
GcMultiRow1[5, 0].Note.Style.TouchToolBar = touchToolBar;

Touch Keyboard

In MultiRow, you can show or hide the touch keyboard and also specify the type of keyboard to be displayed. This section describes how you can control the touch keyboard.

Showing or Hiding the Touch Keyboard

You can show or hide the touch keyboard using the ShowTouchKeyboard ('ShowTouchKeyboard Method' in the on-line documentation) and HideTouchKeyboard ('HideTouchKeyboard Method' in the on-line documentation) methods.

For example, if you use the ShowTouchKeyboard and HideTouchKeyboard methods in the CellBeginEdit ('CellBeginEdit Event' in the on-line documentation) and CellEndEdit ('CellEndEdit Event' in the on-line documentation) events respectively, you can show or hide the touch keyboard in accordance with the start or end of cell editing.

Using Code

This example shows and hides the keyboard.

[VB]
Handles GcMultiRow1.CellBeginEdit
' Show the touch keyboard when the editing of the cell starts.
GcMultiRow1.ShowTouchKeyboard()
End Sub

Handles GcMultiRow1.CellEndEdit
' Hide the touch keyboard when the editing of the cell ends.
GcMultiRow1.HideTouchKeyboard()
End Sub

[CS]
{
    // Show the touch keyboard when the editing of the cell starts.
    gcMultiRow1.ShowTouchKeyboard();
}
{
    // Hide the touch keyboard when the editing of the cell ends.
    gcMultiRow1.HideTouchKeyboard();
}

Setting the Type of the Keyboard

You can use the CellStyle.InputScope ('InputScope Property' in the on-line documentation) property to set the type of touch keyboard to be displayed.

Using Code

The following code displays a different touch keyboard for each cell.
Imports GrapeCity.Win.MultiRow
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
Dim textBoxCell1 = New TextBoxCell() textBoxCell1.Style.InputScope = InputScopeNameValue.Default
Dim textBoxCell2 = New TextBoxCell() textBoxCell2.Style.InputScope = InputScopeNameValue.AlphanumericHalfWidth
Dim numericUpDownCell1 = New NumericUpDownCell()
numericUpDownCell1.Style.InputScope = InputScopeNameValue.Number
GcMultiRow1.Template = Template.CreateGridTemplate(New Cell() {textBoxCell1, textBoxCell2, numericUpDownCell1})
GcMultiRow1.RowCount = 10
End Sub
Handles GcMultiRow1.CellBeginEdit
' Display the touch keyboard when the editing of the cell starts.
GcMultiRow1.ShowTouchKeyboard()
End Sub
Handles GcMultiRow1.CellEndEdit
' Hide the touch keyboard when the editing of the cell ends.
GcMultiRow1.HideTouchKeyboard()
End Sub

Using Code
This example sets the AutoScrollWhenKeyboardShowing property to true.

[VB]
GcMultiRow1.AutoScrollWhenKeyboardShowing = True

[CS]
gcMultiRow1.AutoScrollWhenKeyboardShowing = true;

Gripper during Cell Editing

You can display a gripper for selecting the entered text in an editable cell. This section provides information about the gripper displayed during editing.
Cell Types Displaying the Gripper during Editing

The gripper for selecting the entered text during editing is displayed for the following common cell types.

- TextBoxCell
- MaskedTextBoxCell
- FilteringTextBoxCell

Hiding the Gripper

By default, the gripper is displayed during editing. If you want to hide the gripper during editing, you can set the `ShowGrippersInEditingStatus` property to False. Since the `ShowGrippersInEditingStatus` property setting is enabled for all the cells placed on the template, you can show or hide the gripper for all the cells at once.

Using Code

This example sets the `ShowGrippersInEditingStatus` property.

[VB]
```vbnet
GcMultiRow1.ShowGrippersInEditingStatus = False
```

[CS]
```csharp
gcMultiRow1.ShowGrippersInEditingStatus = false;
```

- During mouse operation, the gripper is always hidden, regardless of the setting of the `ShowGrippersInEditingStatus` property.
- The style of the gripper during editing, cannot be changed using the `TouchSelectionGripperLine` property and `ShowGrippersInEditingStatus` properties.

Zoomed Display of the Drop-down List

In column header cells, it is possible to have a zoomed-in view of the drop-down list displayed during the touch operation. This section provides information about the zoomed-in display of the drop-down list.

Using Code

The following code enlarges the size of the drop-down list of the `ColumnHeaderCell` which is a common cell, to 200% of its default size.
[VB]
Imports System.Windows.Forms
Imports GrapeCity.Win.MultiRow
Dim textBoxCell1 = New TextBoxCell()
textBoxCell1.Name = "textBoxCell1"
textBoxCell1.Size = New Size(200, 20)
Dim template As Template = template.CreateGridTemplate(New Cell() {textBoxCell1})
Dim columnHeaderCell = DirectCast(template.ColumnHeaders(0).Cells(0), ColumnHeaderCell)
columnHeaderCell.DropDownList = New HeaderDropDownList("textBoxCell1", False, False)
Dim headerDropDownContextMenu1 = New HeaderDropDownContextMenu()
headerDropDownContextMenu1.Items.AddRange(New ToolStripItem() { New SortToolStripItem(), New SortToolStripItem(),
New ToolStripSeparator(), New ShowAllToolStripItem(), New ToolStripSeparator(), New AutoFilterToolStripItem()})
' Enlarge the size of the drop-down list to 200% of its default size.
headerDropDownContextMenu1.TouchDropDownScale = 2.0F
columnHeaderCell.DropDownContextMenuStrip = headerDropDownContextMenu1
GcMultiRow1.Template = template

[CS]
using System.Windows.Forms;
using GrapeCity.Win.MultiRow;
TextBoxCell textBoxCell1 = new TextBoxCell();
textBoxCell1.Name = "textBoxCell1";
textBoxCell1.Size = new Size(200, 20);
Template template = Template.CreateGridTemplate(new Cell[] { textBoxCell1 });
ColumnHeaderCell columnHeaderCell = template.ColumnHeaders[0].Cells[0] as ColumnHeaderCell;
columnHeaderCell.DropDownList = new HeaderDropDownList("textBoxCell1", false, false);
HeaderDropDownContextMenu headerDropDownContextMenu1 = new HeaderDropDownContextMenu();
headerDropDownContextMenu1.Items.AddRange(new ToolStripItem[] { new SortToolStripItem(), new SortToolStripItem(),
new ToolStripSeparator(), new ShowAllToolStripItem(), new ToolStripSeparator(), new AutoFilterToolStripItem()});
// Enlarge the size of the drop-down list to 200% of its default size.
headerDropDownContextMenu1.TouchDropDownScale = 2.0F;
columnHeaderCell.DropDownContextMenuStrip = headerDropDownContextMenu1;
gcMultiRow1.Template = template;

Retrieving the Input Device

You can operate a touch device using touch or a touch pen. In addition, you can also operate it using the mouse and keyboard. In MultiRow, you can retrieve information to specify which input device was used for operating the GcMultiRow control. This section explains how to get the information related to the input device.

Retrieving the Input Device

You can use the `InputDeviceType` (`InputDeviceType Property` in the on-line documentation) property to retrieve information about the input device that is operating GcMultiRow.
Using Code

The following code retrieves the information related to the input device operating the MultiRow control.

[VB]

```vbnet
Console.WriteLine("Operated by {0}.", GcMultiRow1.InputDeviceType)
```

[CS]

```csharp
Console.WriteLine("Operated by {0}.", gcMultiRow1.InputDeviceType);
```

Controlling the Display of Touch Keyboard

You can use the `InputDeviceType` (InputDeviceType Property in the on-line documentation) property and the `CellEnter` (CellEnter Event in the on-line documentation) event, to display the touch keyboard only in case of touch operations.

Using Code

This example displays the touch keyboard.

[VB]

```vbnet
    If GcMultiRow1.InputDeviceType = GrapeCity.Win.MultiRow.InputDeviceType.Touch
    Then ' Display the touch keyboard in case of touch operation.
        GcMultiRow1.ShowTouchKeyboard()
    End If
End Sub
```

[CS]

```csharp
private void gcMultiRow1_CellEnter(object sender, GrapeCity.Win.MultiRow.CellEventArgs e)
{
    if (gcMultiRow1.InputDeviceType == GrapeCity.Win.MultiRow.InputDeviceType.Touch)
    {
        // Display the touch keyboard in case of touch operation.
        gcMultiRow1.ShowTouchKeyboard();
    }
}
```

Troubleshooting

- Unexpected License Dialog Gets Displayed
- Template Added in Project Does Not Display in Smart Tag
- Unable to Select MathStatistics Object in SummaryCell
- Unable to Modify BackColor of Button and Header
- Unable to Edit Cell Placed in Column Header
- Value in EditedFormattedValue Property and the Value Displayed Are Different
- User-Defined Cell Added in Project Does Not Appear in Toolbox
- Vertical Alignment and Equal Distribution Do Not Get Set for the Cell Text
- Cannot Find Table Information in Template Source Code
- Cell Validation Result Is Not Visible
- Cell.DesignTimeValue Property Gives a Compile Error
- Error "LC.exe" Exited with Code -1 Occurs While Building
- Flickering Display of Help
- Unable to Sort (Filter) Last Row
- Displaying Bound Data in a Cell Placed in the Column Header Section

Unexpected License Dialog Gets Displayed

If you have installed the licensed version of the product and you are still getting the Trial version dialog, review the information in the following sections.

Project was created using the trial version

Project created using trial version may have trial license information still remaining in its temporary file. You will need to rebuild the project using the following steps to update this information.
Open the target project.
From Build menu of Visual Studio, run Rebuild (ProjectName) or Rebuild Solution.

Project does not have a licenses.licx file
This happens when either the licenses.licx file does not exist, or if you have generated the control dynamically using code and not pasted it from the toolbox. If this is the case, you will have to manually create a licenses.licx file. To do so, complete the following steps:

1. Create any Windows Application Project in Visual Studio. (Visual Basic or C#).
2. In Form Designer of Visual Studio, place GcMultiRow control on the Form from the Toolbox.
3. Open Solution Explorer of Visual Studio, open Licenses.licx file under MyProject folder or Properties folder, and copy details similar to the following to the clipboard.

   \[Licenses.licx\]
   

4. Create a blank file with the name Licenses.licx under My Project or Properties folder. If a file with same name already exists, then open that file.
5. Open the created Licenses.licx file and paste the above information into Licenses.licx file.

When using MultiRow in a UserControl
If you have created your own control inherited from a product control or you have created a UserControl by combining it with some other control, you will have to set the LicenseProvider attribute in the control. You can set LicenseProvider in the control by putting LicenseProvider attribute before class declaration as shown in the following sample:

\[VB\]

Imports System.Windows.Forms
Imports System.ComponentModel

' In case of inherited control
&LicenseProviderAttribute(GetType(LicenseProvider))\)_  
Public Class MyControl
   Inherits GrapeCity.Win.MultiRow.GcMultiRow
   ...
End Class

' In case of user control
&LicenseProviderAttribute(GetType(LicenseProvider))\) _
Public Class UserControl1
   ...
End Class

\[CS\]

using System.Windows.Forms;
using System.ComponentModel;

// In case of inherited control

[LicenseProviderAttribute(typeof(LicenseProvider))]
public partial class MyControl : GrapeCity.Win.MultiRow.GcMultiRow
{
   ...
}

// In case of user control

[LicenseProviderAttribute(typeof(LicenseProvider))]
public partial class UserControl1 : UserControl
{
   ...
}

When placed in a class library and called from another assembly
If you are using GcMultiRow control in a class library and trying to reuse (which includes calling dynamically using reflection) it from assembly through the class library, then the calling project will also require a MultiRow license. If this is the case, add license information in the licenses.licx file of the calling project, similar to the case when the licenses.licx file does not exist in the project.

Hardware configuration has been changed after installation
Refer to the Order FAQ section available on our official website.

Template Added in Project Does Not Display in Smart Tag
If the template from the project does not appear in the smart tag of the GcMultiRow placed on the form, build your project and make the template object referable.

Unable to Select MathStatistics Object in SummaryCell
You cannot select the **MathStatistics** (‘MathStatistics Class’ in the on-line documentation) object in a SummaryCell. **Calculation** (‘Calculation Property’ in the on-line documentation) property if you have placed a SummaryCell in a row in the designer. The MathStatistics option is available only when SummaryCell is placed in the ColumnHeaderSection or ColumnFooterSection. When working in a row, you can use the **Expression** (‘Expression Class’ in the on-line documentation) object to get the sum of cell values.

### Unable to Modify BackColor of Button and Header

By default for **ButtonCell** (‘ButtonCell Class’ in the on-line documentation) and **HeaderCell** (‘HeaderCell Class’ in the on-line documentation), the background specified in the Visual Style is given preference; however, you can disable this setting and enable the background color by completing the following steps.

1. Set ButtonCell.FlatStyle (‘FlatStyle Property’ in the on-line documentation) property to Standard.

You can set the same properties for **ColumnHeaderCell** (‘ColumnHeaderCell Class’ in the on-line documentation) and **HeaderCell** (‘HeaderCell Class’ in the on-line documentation) also. You can do the same for using BackColor Gradation and BackColor Pattern.

For more information, see the following topics:

- [Change BackColor (HeaderCell)](#)
- [Change BackColor (ButtonCell)](#)
- [BackColor and ForeColor](#)

### Unable to Edit Cell Placed in Column Header

By default, you cannot edit cells in column headers or column footers. For example, if you place a textBoxCell in the column header and run your project, the cell will be neither selectable nor editable. If you want to allow users to edit cells in the column header or footer, use the settings below.

- Set Section.ReadOnly (‘ReadOnly Property’ in the on-line documentation) property to False.
- Set Section.Selectable (‘Selectable Property’ in the on-line documentation) property to True.

Also, if you are unable to select or edit a cell, check the following settings.

- Set Cell.ReadOnly (‘ReadOnly Property’ in the on-line documentation) property to False.
- Set Cell.Selectable (‘Selectable Property’ in the on-line documentation) property to True.

Finally, if the cell type by default does not support user edits, then you cannot provide editing even when you place the cell in a column header or column footer.

### Value in EditedFormattedValue Property and the Value Displayed Are Different

Because the Cell.EditedFormattedValue (‘EditedFormattedValue Property’ in the on-line documentation) property returns the value stored in the cell after it has been converted to a string (the result of Object.ToString), the value may not be always be the same as the value displayed. For example, **DateTimePickerCell** (‘DateTimePickerCell Class’ in the on-line documentation) stores values as DateTime values, so even if the cell displays only date information, the cell still contains the time information. You can use the Cell.DisplayText (‘DisplayText Property’ in the on-line documentation) property to return the displayed cell value as a string.

### User-Defined Cell Added in Project Does Not Appear in Toolbox
If a user-defined cell is not displayed in the toolbox, perform the following actions.

- Build the project.
- Set the Document window tab of the Template designer to active.

**Vertical Alignment and Equal Distribution Do Not Get Set for the Cell Text**

The Vertical Alignment or Equal Distribution option for a cell can be enabled only when GDI+ Compatibility Mode is set. For example, you can use the following settings and enable Equal Distribution for a cell.

- Set Equal Distribution or any text in TextBoxCell.Value ('Value Property' in the on-line documentation) property.
- Set TextBoxCell.Style.TextAdjustment ('TextAdjustment Property' in the on-line documentation) property to Distribute.
- Set TextBoxCell.Style.UseCompatibleTextRendering ('UseCompatibleTextRendering Property' in the on-line documentation) property to True.

**Cannot Find Table Information in Template Source Code**

Because the Table is meant to be design information it is written in Resource and not in Source Code. For example, if the template file name is Template1.vb, the resource file will be Template1.resx.

**Cell Validation Result Is Not Visible**

To provide validation for each cell (Cell.Validators ('Validators Property' in the on-line documentation) and to display a validation result, you need to explicitly specify validation actions. Add LineNotify or IconNotify in the Actions property for each cell validator.

**Cell.DesignTimeValue Property Gives a Compile Error**

Because the Cell.DesignTimeValue property is displayed only at design time, you cannot use it in the code editor. Use the Cell.Value ('Value Property' in the on-line documentation) property to specify cell values in code.

**Error "LC.exe" Exited with Code -1 Occurs While Building**

While building the application, "LC.exe" exited with code -1 error might occur, at which time the build stops. You can avoid this issue by opening the licenses.licx file for the project and deleting the license information for the controls that are not being used.

- In Visual Basic, licenses.licx gets placed inside the My Project folder in the tree on selecting Show All Files from toolbar of Solution Explorer window.
- In C#, licenses.licx is placed inside the Properties folder found under the Solution Explorer window.

**Flickering Display of Help**

In case Microsoft Document Explorer 9.0 is used to display MultiRow help in Windows Vista or Windows 7, and Visual Studio 2008 environment, the display flickers when the screen is clicked or when page transition occurs. This is a common issue of Document Explorer 9.0 and not specific to MultiRow only.
You can reduce the occurrence of this issue by doing disabling visual themes of Microsoft Document Explorer 9.0. To do so, right-click on the MultiRow help shortcut from the Start menu, select Compatibility tab from MultiRow Windows Forms Help Properties screen and check the Disable Visual Themes option.

Unable to Sort (Filter) Last Row

When the GcMultiRow.AllowUserToAddRows ('AllowUserToAddRows Property' in the online documentation) property is set to True, the last row is treated as a new row and because editing has not been confirmed in the new row, sorting and filtering cannot be carried out for that particular row. You should place a RowHeaderCell in this row and display an indicator to make the user aware that it is a new row.

Displaying Bound Data in a Cell Placed in the Column Header Section

Data binding in the column header section is now supported in MultiRow for Windows Forms 7.0. Because data binding in the column header section was not supported in MultiRow for Windows Forms 6.0, if a project is created in 6.0 and if you have specified the name of a bound datasource field in the DataField ('DataField Property' in the online documentation) property of a cell placed in the column header section, unnecessary data is displayed after migration to 7.0.

As a workaround for this behavior, manually delete the set value of the DataField ('DataField Property' in the online documentation) property.
Limitations

The following limitations are applicable to all the assembly files of MultiRow Windows Forms (referred to as MultiRow).

Installation

- Make sure Visual Studio is not open when you run the installer.
- Since the installation takes some time to integrate the designer in Visual Studio, please do not terminate the installer forcibly during that time.
- In case of upgrading the OS, please uninstall MultiRow before upgrading the OS and re-install it after upgrading by using the installer that supports the new OS.
- The MultiRow designer does not get installed properly if the product is installed before installing Visual Studio. Uninstall MultiRow and re-install it.

General

- When both the CellIndex and the CellName can be assigned to the object, the later settings will overwrite the former.
- The result of the visual style depends on the OS settings.
- Animation GIF is not supported.
- There is no counterpart of Share Row functionality of the DataGridView control.
- In Windows XP/Server 2003/7/Server 2008 R2, the Extend support of advanced text services to all programs option is not supported when input is done in non-editable mode.
- The BorderBase class cannot be inherited. Use owner draw for a user-defined border.
- By default, the appearance of border styles "Dashed" and "Hair" is the same.
- UI automation is not supported.

Designer

- The designer cannot be used in the Visual Studio Express Edition. For details, refer to System Requirements.
- Templates not derived from the Template class cannot be edited in the designer. For instance, a template created by inheriting Template1 which is derived from Template class cannot be edited in the designer.
- Multiple cells selected across sections cannot be copied. In this case, the cells that are selected out of the active region get de-selected automatically.
- The following cell types cannot be placed in the row section.
  - ColumnHeaderCell
  - CornerHeaderCell
  - FilteringTextBoxCell
- RowHeaderCell cannot be placed in the ColumnHeaderSection or ColumnFooterSection.
- The column name of the Property List Window 7.0 is always in English. This is a specification in order to match with the property name of the cell.
- The position or size of the cell can be set in Pixel units only.
- The ImageCell value cannot be edited in Property List Window 7.0.
- When the designer is in run mode, the GcMultiRow control shortcut keys and cell types are disabled but the Visual Studio shortcut keys work.
- Information of objects placed in the component tray are not displayed in the Template Explorer 7.0.
- In the Template Named Cell Style 7.0 window, the usability status of the NamedCellStyle, which is used in templates, sections, combined style, and conditional style cannot be identified.

GcMultiRow Control

- A key assigned in Windows cannot be assigned as a control key.
• Zooming of borders, scroll bars, and tooltips is not supported.
• The GcMultiRow.WheelMouseCount property does not support horizontal scrolling with the tilt wheel.
• A hidden cell cannot be selected by the user. For instance, even if a row with a hidden cell is selected, the hidden cell remains unselected.
• The product does not have any functionality similar to the CellStateChanged and RowStateChanged events of the System.Windows.Forms.DataGridView control.
• The following styles are disabled when the CellStyle.UseCompatibleTextRendering property is set to False.
  • Text rotation (CellStyle.TextAngle)
  • Line adjustment (CellStyle.LineAdjustment)
  • Text adjustment (CellStyle.TextAdjustment)
  • Vertical text (CellStyle.TextVertical)
• Please refer to GDI+ Compatibility Mode for limitations when the CellStyle.UseCompatibleTextRendering property is set to True.
• The scroll bar and background images do not get printed.
• Cell or row selections are not printed. The selected cell or row is printed with the usual backcolor and font color.
• The user cannot resize a cell with size "0" at runtime.
• The display of the grid and the printed result may be different. This happens because of differences in rendering GDI and GDI+.
• The changed value is not highlighted in bold in the GcMultiRow.SplitStyle property dialog. This is because the values of SplitStyle are stored in structures.

Template

• Cells with the same name (Cell.name property) cannot be used in the templates. The cell name has to be unique so that it can be identified.

Cell Type

• The selection mode of ColumnHeaderCell and built-in sorting cannot be used simultaneously. Only one of them can be assigned to the click event of ColumnHeaderCell.
• ColumnHeaderCell cannot be added to a row. RowHeaderCell or HeaderCell can be used.
• ColumnHeaderCell cannot be added to a column footer. HeaderCell can be used.
• The object of the RowHeaderCell class cannot be added to a column header or a column footer section.
• Other images cannot be displayed in a column header while you are customizing the glyph that indicates the ascending or descending sorting direction in the ColumnHeaderCell class object. A custom rendering process has to be added to the column header in order to display the images together with the customization of the glyph.
• If HeaderCell is placed in a column footer, cells cannot be selected using this HeaderCell. Selection can be done only when it is placed in a column header or a row.
• Displaying error value icons has a higher priority than displaying cell type contents.
• The text alignment does not apply when the ComboBoxCell is in edit mode. This happens because of the ComboBox control behavior.
• The Checkbox cannot be used in a DateTimePickerCell.
• The style background color does not apply when the DateTimePickerCell is in edit mode. This happens because of the DateTimePicker control behavior.
• Cell types do not support the animation effect of visual styles.
• Cell types do not support access keys.
• In the CheckBoxCell, RadioGroupCell, LinkLabelCell, and ButtonCell, the focus frame doesn't get displayed in caption.
- In HeaderCell, when the value of HeaderCell.FlatStyle is System and visual effects provided by OS are enabled, OS visual effects are given preference during rendering of the header border. Thus the value of the HeaderStyle.GutterStyles property and cell appearance may not match.
- ProgressBarCell.ProgressBarStyle = Blocks is equivalent to the Continues effect of Windows Vista/7.
- The SummaryCell does not support databinding.
- Cells in ColumnHeaderSection and ColumnFooterSection do not support databinding.

Help file

- Comments and sample data in code samples in the assembly reference are provided in English.
2  Index

About MultiRow, 20
Adding Cells, 138-139
Adding Cells in the Designer, 104-105
Adding Column Footers, 140-142
Adding Column Headers, 139-140
Adding Columns, 326-329
Adding MultiRow Component and Template to the Project, 57-61
Adding Rows, 305-307
Adding the MultiRow Component to a Visual Studio 2013 Project, 61-65
Adjust Character Size with Placeholders, 126-127
Advanced Designer Features, 115-116
Assigning and Allocating a Template, 149-150
Automatic Adjustment of the Cell Width, 182-183
Automatic Merging of Cells, 178-180
Automatically Confirm Initial Value as Input Value (DateTimePickerCell), 242
BackColor and ForeColor, 350-352
Background Gradation, 354-356
Background Pattern, 352-354
Base Cell, 200-201
Basic Concepts, 45-48
Borders, 347-350
Breaking Changes, 40
Breaking Changes from 6.0, 40-42
Built-in Cell Validators, 384
Built-in Validation Actions, 401
ButtonCell, 215-218
Cannot Find Table Information in Template Source Code, 454
Cell Display Modes in the Designer, 113-114
Cell Note, 321-325
Cell Selection, 437-439
Cell Structure, 51-56
Cell Styles, 345
Cell Types, 199-200
Cell Validation Result Is Not Visible, 454
Cell.DesignTimeValue Property Gives a Compile Error, 454
Change BackColor (ButtonCell), 218-219
Change BackColor (HeaderCell), 203-204
Changing Cell Index in the Designer, 109-110
Changing Cell Styles based on Conditions, 374-376
Changing Height and Width of a Row in Designer, 103
Changing Tab Order of Cells in the Designer, 111-112
MultiRow Windows Forms Developer's Guide

Changing the Header Appearance, 197-198
CheckBoxCell, 222-224
Co-existence and Migration from the Previous Version, 42
Co-existence with the Previous Version, 42-43
Column Mode, 325
Column Mode Template, 144-148
Column Operation, 332-334
ColumnHeaderCell, 204-206
Combining Multiple Styles, 376-377
ComboBoxCell, 225-229
CompareCellValidator, 392-393
CompareStringValidator, 398-399
CompareValueValidator, 393-394
Compatibility with the Previous Version, 40
Conditional Cell Styles, 369
Connecting to Database, 416-419
Connecting to SQL Server Database, 66-73
Context Menu, 169-171
Conversion from DataGridView, 132-136
Copying Cells in the Designer, 106-107
CornerHeaderCell, 208
Create Statement of Account (SummaryCell), 291-294
Create Template from Data Sources Window, 118-120
Create Template from Table, 120-125
Create Template using Wizard, 116-118
Create Templates Using Tracing Mode, 125-126
Creating an Application using GcMultiRow, 65-66
Creating an Application Using the Designer (Grid View), 73-81
Creating an Application Using the Designer (Single Template View), 81-89
Current Cell, 304-305
Current Column, 325-326
Current Row, 303-304
Custom Printing, 422
Customizing the Justification of Single-byte Characters, 368-369
Customizing the Validation Information, 411-412
Data Binding Modes, 338-340
DateTimePickerCell, 235-239
Deleting Columns, 329-332
Deleting Rows, 307-309
Designer Options, 99-102
Designer Window, 89
Detect Value Change (CheckBoxCell), 225
Detect Value Changed (RadioGroupCell), 272-273
Developer's Guide, 1
Disabled Cells, 315
Display Alternate Image for Empty Value (ImageCell), 261-262
Display Alternate Text for Null Values (DateTimePickerCell), 239-240
Display Image (ImageCell), 263
Display Modes of the Designer, 112-113
Display String (ButtonCell), 220-221
Display String (LabelCell), 256-257
Display Vertical Scroll Bar (TextBoxCell), 214-215
Displaying a New Row at the Top, 180-181
Displaying an Error Icon for Data Errors (SummaryCell), 290-291
Displaying Bound Data in a Cell Placed in the Column Header Section, 455
Displaying Error Messages, 319-320
Displaying Negative Values in Red (NumericUpDownCell), 246-248
Displaying Numeric Value Separator (NumericUpDownCell), 249-250
Document Window, 90-91
DomainUpDownCell, 250-253
Edit Modes, 150-153
Editing Cells, 317-319
EncodingValidator, 400-401
End-User License Agreement, 19
Error "LC.exe" Exited with Code -1 Occurs While Building, 454
ExcludeListValidator, 391-392
Expanding Or Collapsing Columns, 181-182
Feature Comparison, 24-26
File Input and Output, 431
Filtering Rows using the Column Headers, 190-194
FilteringTextBoxCell, 299
Flickering Display of Help, 454-455
Format String, 56
Freezing Rows, 310-312
GDI+ Compatibility Mode, 363-366
Get and Set Data, 335-338
Get Formatted Values (DateTimePickerCell), 241-242
Get Index Of Selected Value (ComboBoxCell), 229-230
Get or Set Cell Properties in the Designer, 108-109
Getting and Setting the Row Count, 309-310
Grid, 148-149
Gripper during Cell Editing, 448-449
Handle User Click (ImageCell), 262-263
Handling User Click (ButtonCell), 219-220
Handling User Click (LabelCell), 255-256
HeaderCell, 201-203
NumericUpDownCell, 243-246
Open Link in Associated Browser (LabelCell), 259
Paging Mode, 424-425
PairCharValidator, 394-396
Points to Note with Default Settings, 183-185
Pop-upCell, 282-284
Print and Print Preview, 419-420
Print Settings for each Column Header Section, 430-431
Print Style, 422-424
PrintInfoCell, 294-296
Printing Watermarks, 427-430
Product Overview, 20-21
Product Structure, 21
ProgressBarCell, 263-266
Properties Window, 94
Quick Start, 56-57
RadioGroupCell, 269-272
RangeValidator, 384-387
Read-Only Cells, 320-321
Refer to Editing Value when Modifying TrueValue and FalseValue (CheckBoxCell), 224-225
Register User Defined Cell in ToolBox, 302
RegularExpressionValidator, 397-398
Replacing Cells in the Designer, 105-106
RequiredFieldValidator, 390-391
RequiredTypeValidator, 388-390
Resizing, 176-177
Resizing Cells in the Designer, 108
Resizing the Column Width, 439-440
Restricting Cell Selection, 314-315
Retrieving the Input Device, 450-451
Retrieving the Method Used to Move Cells, 177-178
RichTextBoxCell, 266-269
RowHeaderCell, 206-208
Rows and Cells, 302-303
Rules for Applying Styles, 345-347
Scrolling, 159-162, 436-437
Select Color (Pop-upCell), 284-285
Selected Rows and Cells, 312-314
Selecting Cells in the Designer, 107-108
Selecting Cells using Headers, 189-190
Selection Mode, 153-156
Set Default Value in New Row, 344-345
Setting Bullets (ListLabelCell), 277-278
Setting Shapes (ShapeCell), 280-281
Setting the Column Mode Template in the Designer, 115
ShapeCell, 278-280
Shortcut Keys, 171-176
Show Column Sum in Footer (SummaryCell), 288-290
Simple Printing, 420-421
Sorting Data, 340-343
Sorting Rows using Column Headers, 194-197
Specify Print Range, 425-426
Specifying Empty Rows for Print, 427
Specifying the Page Break Position, 426-427
Splitter Line and ViewPort, 166-169
Status Bar, 93-94
SummaryCell, 285-288
SurrogateCharValidator, 396-397
System Requirements, 12-13
Tag, 356-358
Technical Support, 17
Template, 138
Template - Named Cell Styles 7.0, 98-99
Template Added in Project Does Not Display in Smart Tag, 452
Template Explorer 7.0, 97-98
Template File, 50-51
Template Localization, 127-130
Template Property List 7.0, 94-97
Template Structure, 48-50
Template Width and Height, 142-144
Text Formatting, 361-362
Text Indent, 362-363
TextBoxCell, 208-212
TextLengthValidator, 399-400
TipNotify, 406-407
Tips on Improving the Performance, 185-189
Toolbar, 93
Toolbox, 91-92
Touch Keyboard, 447-448
Touch Toolbar, 440-447
TrackBarCell, 296-299
Troubleshooting, 451
Types of Cell Styles, 369-371
Unable to Edit Cell Placed in Column Header, 453
Unable to Modify BackColor of Button and Header, 453
Unable to Select MathStatistics Object in SummaryCell, 452-453
Unable to Sort (Filter) Last Row, 455
Undo Changes in Designer, 110-111
Unexpected License Dialog Gets Displayed, 451-452
Usage of ValueChanged Event (DateTimePickerCell), 242-243
User Defined Conditional Styles, 377-378
User Defined Validation Actions, 413-416
User Input Validation, 382
User-Defined Cell, 299-302
User-Defined Cell Added in Project Does Not Appear in Toolbox, 453-454
User-Defined Cell Validator, 412-413
User-defined Word Wrap, 367-368
Using Annotations, 136-138
Using Button Commands (ButtonCell), 221-222
Using MultiRow, 45
Using Row Headers, 198-199
Using the Designer, 102-103
Validation, 383-384
Validation using Control Events, 382-383
Value in EditedFormattedValue Property and the Value Displayed Are Different, 453
Value Validation by Cell, 383
ValueProcess, 408-409
Vertical Alignment and Equal Distribution Do Not Get Set for the Cell Text, 454
View Modes, 156-159
Visual Style, 162-163
What's New, 26
What's New in the Designer, 27-33
What's New in the Grid Control, 33-38
Working with Clipboard, 343-344
Working with Column Footers in the Designer, 104
Working with Column Headers in the Designer, 103-104
Zoomed Display of the Drop-down List, 449-450
Zooming, 163-166, 432-436