10Tec iGrid for .NET 5.0
What's New in the Control

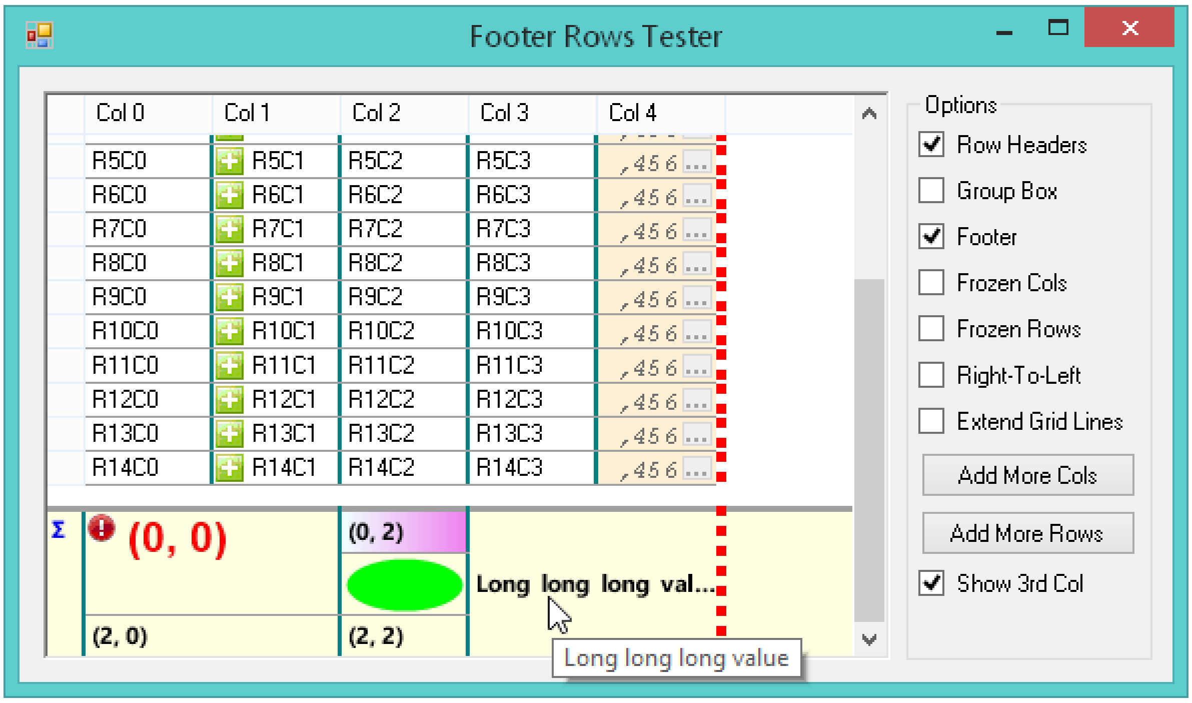
Keywords used to classify changes:

- [Added] – a brand new feature was implemented;
- [Changed] – a change in a member functionality or interactive behavior;
- [Fixed] – a fixed bug or solved problem;
- [Improved] – something is implemented better than in previous versions;
- [Removed] – a feature was removed;
- [Renamed] – a name of the member was changed so it is enough to rename it in your code.

New footer section

General features of footer

This release of iGrid.NET introduces a new footer section. The footer section is a special area always displayed at the bottom; it has the same set of columns defined in the grid and may have several rows. From this point of view the footer is similar to the iGrid header, but it contains footer cells. Footer cells cannot be edited interactively and implement almost all features of normal grid cells. They also provide you with some specific features like automatic calculation of totals and cell merging:



Col 0	Col 1	Col 2	Col 3	Col 4
R5C0	R5C1	R5C2	R5C3	,456 ...
R6C0	R6C1	R6C2	R6C3	,456 ...
R7C0	R7C1	R7C2	R7C3	,456 ...
R8C0	R8C1	R8C2	R8C3	,456 ...
R9C0	R9C1	R9C2	R9C3	,456 ...
R10C0	R10C1	R10C2	R10C3	,456 ...
R11C0	R11C1	R11C2	R11C3	,456 ...
R12C0	R12C1	R12C2	R12C3	,456 ...
R13C0	R13C1	R13C2	R13C3	,456 ...
R14C0	R14C1	R14C2	R14C3	,456 ...

Σ (0, 0) | (0, 2) | Long long long val...
(2, 0) | (2, 2)

Long long long value

- [✓] Row Headers
- [ ] Group Box
- [✓] Footer
- [ ] Frozen Cols
- [ ] Frozen Rows
- [ ] Right-To-Left
- [ ] Extend Grid Lines
- [✓] Show 3rd Col

The footer area is controlled through the new **Footer** property of iGrid. It is an object of the new **iGFooter** type, and its properties and methods allow you to configure the footer section.

Like the header, the footer contains one footer row by default. But in contrast to the header, the footer section is hidden for every new instance of iGrid. To make it visible, set its **Visible** property to True.

The main footer formatting properties are **BackColor**, **ForeColor**, **Font**. By default the footer uses the colors of the system Windows tooltip. The **Font** property is not set by default, which means the grid's font is used. The **Footer.SeparatingLine** property of the **iGPenStyle** type specifies the thickness, color and style of the special line at the top of the footer area that separates it from the main grid contents.

To access footer cells and change their properties, use the **Cells** collection of the **Footer** object property. Footer rows are indexed from top to bottom, and the top (or only) row always has the index of 0. Thus, if you want to display the value of 100 in the footer cell for the 5th column, you need code like this:

```
iGrid1.Footer.Cells[0, 4].Value = 100;
```

Every footer cell is represented with an instance of the new **iGFooterCell** class. It implements almost all properties of normal cells (i.e. the **iGCell** class) – **Value**, **ImageIndex**, **BackColor**/**ForeColor**, **Style**, **TextTrimming**, an so on. There is no difference in using these properties for normal cells and footer cells. The only difference is that a footer cell style is an object of the new **iGFooterCellStyle** class that contains the corresponding properties of the footer cell class.

One of the main ideas that works for an iGrid footer cell is the inheritance of the column cell style (the **iGCol.CellStyle** property). If you do not redefine formatting properties of a footer cell, it inherits the look of the cells in the same column (the same format string, alignment, etc.) You can redefine any of these properties if required – for instance, highlight special values with a bold font and/or the red color.

Footer rows can be accessed using the indexed **Row** property of the **Footer** object. For instance, you can access the 3rd row as **iGrid.Footer[2]**. A call like this returns an object of the new **iGFooterRow** class that represents individual footer rows. The two main properties of **iGFooterRow** are **Height** and **Visible**, which can be changed by you.

Note that by default the height of every footer row is calculated automatically when you change the contents of its cells or related formatting properties like font, so assignments to the **iGFooterRow.Height** property have no effect. To change this mode, use the **iGrid.Footer.AutoHeightEvents** property.

Some additional points related to the iGrid footer functionality are the following:

- Footer cells support the classic set of mouse events: **FooterCellMouseDown**, **FooterCellMouseMove**, **FooterCellMouseUp**, **FooterCellMouseEnter**, **FooterCellMouseLeave**. They are raised not only for normal footer cells, but for the footer row header cell and the footer extra cell.
- Footer cells can be merged using the **iGFooterCell.SpanRows** and **iGFooterCell.SpanCols** properties (the same functionality is used for column headers).
- Footer cells have built-in tooltips, which can be redefined on the fly using the **RequestFooterCellToolTipText** event like we can do that for normal cells or column headers.
- Footer cells support custom drawing like normal cells. The row header area in the footer section also supports custom drawing. The **CustomDrawFooterRowHdr** event allows you to draw your custom contents over the standard color fill of the **iGrid.Footer.BackColor** color.

Automatic calculation of totals

Footer cells can be used to automatically calculate and display grid totals. Every footer cell has the **AggregateFunction** property of the **iGAggregateFunction** enumeration type that allows you to specify what aggregate function is calculated for the cells of a column: Sum, Average, Minimum, Maximum, or Count. By default the **AggregateFunction** property is set to **iGAggregateFunction.None**, but if you change this value, iGrid calculates the corresponding function on any appropriate event. These are:

1) Adding or removing grid rows (using the **Rows.Count** property or methods like **Rows.Add**, **Rows.RemoveRange**, and so on).
2) Changing row visibility (setting the **iGRow.Visible** property, but not while collapsing/expanding group rows or tree nodes).
3) Changing cell values (interactively or from code with the **iGCell.Value** property).
4) Populating iGrid from a data source using the **FillWithData** method.

Totals are not re-calculated between **BeginUpdate**/**EndUpdate** or if the new **CalculateTotals** Boolean property is set to False (its default value is True). The **CalculateTotals** property allows you to turn automatic recalculation of totals off when you definitely know that your changes to the grid in the cases listed above will not affect the specified totals.

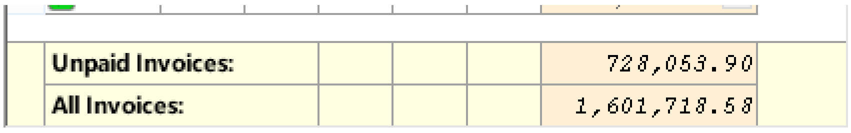
iGrid also raises the new **RecalculateTotals** event for every case listed above, and this can be useful if you use your own algorithm to calculate specific totals. In fact, this event tells you when you need to recalculate your totals if anything related to totals has been changed in the grid. This event is raised regardless of the state of the **CalculateTotals** property.

Totals are calculated only for numeric cell values (except the Count function) and are stored as **System.Double** values. In fact, iGrid tries to convert a cell value to the corresponding value of the **Double** type, and use this value in the calculation if the conversion was successful. The result of the calculation is stored in the **Value** property of the corresponding footer cell after every recalculation of totals, and it can be read or even rewritten in code using this property.

Pay attention to the fact that you can define more than one aggregate function for the same column if you have several rows in the footer. For example, the first footer cell in the 4th column can display the row counter and the second footer cell can display the sum:

```
iGrid1.Footer.Cells[0, 3].AggregateFunction = iGAggregateFunction.Count;
iGrid1.Footer.Cells[1, 3].AggregateFunction = iGAggregateFunction.Sum;
```

iGrid extends this idea even more and allows you to define the same aggregate function for the same column several times, but filter rows included into calculations of a particular aggregate function. A practical example is the following: you may have a list of invoices in your grid and calculate two sums – the sum of all invoices and the sum of unpaid ones. It may look like the following picture:



Unpaid Invoices:				728,053.90	
All Invoices:				1,601,718.58	

Row filtering for totals calculation is performed with the new **IncludeRowInTotalsCalculation** event. Its parameters indicate what cell (**RowIndex**, **ColIndex**) is about to be used for totals calculation in a footer row (**FooterRowIndex**) and whether to do that (**DoDefault**). The footer section on the above picture was created with this code:

```
iGrid1.Footer.Rows.Count = 2;
iGrid1.Footer.Cells[0, 0].Value = "Unpaid Invoices:";
iGrid1.Footer.Cells[0, 0].SpanCols = 3;
iGrid1.Footer.Cells[1, 0].Value = "All Invoices:";
iGrid1.Footer.Cells[1, 0].SpanCols = 3;
iGrid1.Footer.Cells[0, 6].AggregateFunction = iGAggregateFunction.Sum;
iGrid1.Footer.Cells[1, 6].AggregateFunction = iGAggregateFunction.Sum;
iGrid1.IncludeRowInTotalsCalculation += iGrid1_IncludeRowInTotalsCalculation;

void iGrid1_IncludeRowInTotalsCalculation(object sender,
iGIncludeRowInTotalsCalculationEventArgs e)
{
   if (e.FooterRowIndex == 0)
   {
      e.DoDefault = (bool)iGrid1.Cells[e.RowIndex,"unpaid"].Value;
   }
}
```

We also imply that all values in the last column (both cell values and footer values) use the same format string, alignment and background color defined in the column's cell style:

```
iGCellStyle colCellStyle = iGrid1.Cols[6].CellStyle;
colCellStyle.BackColor = Color.PapayaWhip;
colCellStyle.TextAlign = iGContentAlignment.MiddleRight;
colCellStyle.FormatString = "{0:#,##.00}";
```

Other changes

1. [Added] The new **IsPointOverFooter** method was added to iGrid to test whether a point with the specified coordinates belong to the footer area. The method has the following syntax:

   ```
   public bool IsPointOverFooter(int x, int y)
   ```

2. [Added] The **backgroundY** parameter was added to the **DrawCellContents** method. It allows you to specify the y-coordinate of the cell background rectangle that may differ from the y-coordinate of the cell contents area (in the previous versions of iGrid, the value of the **y** parameter was used for this purpose inside the method).

3. [Added] The arguments of the **ColHdrMouseMove** and **ColHdrMouseUp** events were supplemented with the **ElemControl** property that indicates whether there is a control under the mouse pointer (the combo button).

4. [Added][Removed][Changed] In the previous builds, the **iGColHdr.SpanRows** and **iGColHdr.SpanCols** properties returned 1 for header cells covered by a bigger merged cell. Starting from this build, these properties return 0 to indicate this special case. If it is the case, use the new **iGColHdr.SpanRoot** property to know the root of the merged cell that spans across the specified cell.

   The value returned by the **iGColHdr.SpanRoot** property has the same **iGColHdr** type and represents a column header cell. If a header cell is not covered by a merged cell, **iGColHdr.SpanRoot** returns the same cell.

   The **GetSpanColsOnLeft**, **GetSpanColsOnRight**, **GetSpanRowsOnTop**, **GetSpanRowsOnBottom** methods of the **iGHeader** class were removed as the new universal **iGColHdr.SpanRoot** property can be used instead. For instance, if we need to know the value returned by the **GetSpanRowsOnBottom** method in the previous builds, we can use a code snippet like the following one:

   ```
   iGColHdr rootCell = fGrid.Header.Cells[rowIndex, colIndex].SpanRoot;
   int spanRowsOnBottom = rowIndex - rootCell.RowIndex + 1;
   ```

   For the **GetSpanColsOnRight** method, we can use the following equivalent:

   ```
   iGColHdr rootCell = iGrid1.Header.Cells[rowIndex, colIndex].SpanRoot;
   int spanColsOnRight = rootCell.SpanCols - (iGrid1.Cols[colIndex].Order -
   iGrid1.Cols[rootCell.ColIndex].Order);
   ```

   The new **iGFooterCell** object implements the same **SpanRows**, **SpanCols** and **SpanRoot** properties as a part of the unified cell merging functionality.

5. [Added] The **iGCell** object implements the new property named **ImageBounds** that returns the rectangle occupied by the cell image (if any). This rectangle can be used in any scenarios – for instance, to check whether the cell image is clicked in the **CellMouseDown** event.

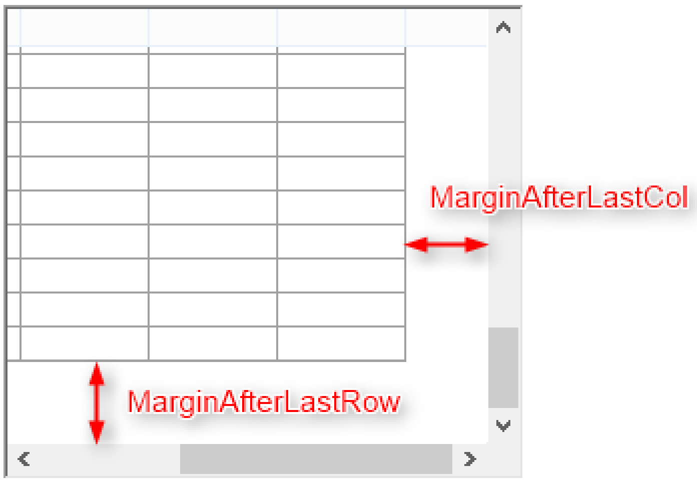
   The **iGColHdr** object was also supplemented with the **TextBounds** and **ImageBounds** properties.

6. [Added] The two new properties, **CurCellForeColorNoFocus** and **CurCellBackColorNoFocus**, were implemented. They are used to specify the colors of the current cell when iGrid is not focused, which is very useful if the current cell is highlighted using special colors in row mode.

7. [Added] The **iGCell** class implements the new property **EffectiveContentIndent** that returns the effective content indent for a normal cell.

8. [Added] The two new properties, **MarginAfterLastRow** and **MarginAfterLastCol**, were added to iGrid. The **MarginAfterLastRow** property allows you to create empty space between the last row

and the next iGrid item at the bottom (the grid's bottom edge, the footer's separating line, etc), which enhances the visual representation of the cells area:



The **MarginAfterLastCol** property specifies the width of the empty space between the last visible column and the next iGrid item at the right (the vertical scroll bar, etc).

The default value of both properties is 0 for backward compatibility.

9. [Added] The two new events that allow you to control the new value for every scroll bar, **HScrollBarValueChanging** and **VScrollBarValueChanging**, were implemented. The event arguments of both events have the new **iGScrollBarValueChangingEventArgs** type. It contains the only integer property named **Value**, which is passed by reference to your event handler and can be changed.

   One of the examples of the usage of the new functionality is so-called integral scrolling, when the user is allowed to scroll the grid only by whole rows so the top edge of the first visible row is always placed beneath the header. This is done enough easily with an event handler of the **VScrollbarValueChanging** event:

```
void iGrid1_VScrollBarValueChanging(object sender,
iGScrollBarValueChangingEventArgs e)
{
   e.Value = (e.Value / iGrid1.DefaultRow.Height) *
iGrid1.DefaultRow.Height;
}
```

   (We imply that all your rows have the same height set in the **DefaultRow.Height** property. Note also that the result of the division operator is always integer as it is C#.)

   To avoid the problem when the very last row is partially hidden by the bottom edge of iGrid as its scrollable height may not be equal to **DefaultRow.Height** multiplied by an integer value, set the **MarginAfterLastRow** property to **DefaultRow.Height** when initializing your grid:

```
iGrid1.MarginAfterLastRow = iGrid1.DefaultRow.Height;
```

10. [Added] The **iGIndent** structure has a new constructor with one argument that allows you to specify the uniform length for all 4 indents. The **ToString()** method of this structure was also redefined to return the string representation of this structure as "{Left=…,Top=…,Right=…,Bottom=…}".

11. [Added] The **CellDynamicContents** event is raised for check box cells now, and the new **CheckState** property of the event arguments allows you to specify the state of the cell check box control dynamically.

12. [Added] The key for the row text column (the **iGrid.RowTextCol.Key** property) can be set at design time in the Property Grid.

13. [Added] The previous versions of iGrid did not allow you to remove header rows because of merged column headers. This restriction was removed in this version, and now header rows that may even span across merged column headers can be removed with the new **iGrid.Header.Rows.RemoveAt()** and **iGrid.Header.Rows.RemoveRange()** methods. The total number of header rows can be also decreased using the **iGrid.Header.Rows.Count** property.

14. [Added][Renamed] The name of the **iGHdrAutoHeightFlags** enumeration was changed to a more appropriate and common name, **iGAutoHeightEvents**, as it is also used now to specify the events when auto-height operation is performed for footer rows (see the **iGrid.Footer.AutoHeightEvents** property). The **iGrid.Header.AutoHeightFlags** property was also renamed to **iGrid.Header.AutoHeightEvents** accordingly.

    The **iGAutoHeightEvents** enumeration was supplemented with the new **OnRemoveRow** element, which is used to auto-height the corresponding grid area when it contains merged cells.

15. [Added][Removed] The iGrid **Header** object implements the new **Bounds** property that returns the **Rectangle** occupied by the header area (the group box is also included if it is visible).

    The **RowHeader** object also implements the similar property now. The **RowHeader.X** property was removed to not duplicate the existing functionality - now it can be retrieved as **RowHeader.Bounds.X**. Note that **RowHeader.Bounds** the coordinates of the area used to draw row header for real rows – though they may be absent in some bottom positions if the grid does not contain enough rows.

    The new **Footer** object implements the same **Bounds** property as a part of this unified functionality.

16. [Removed] To keep the member set clear, the two useless column header properties, **Type** and **TypeFlags** that can accept only one effective value, were removed. They were removed from the **iGColHdr** and **iGColHdrStyle** classes; the related enumerations **iGColHdrType** and **iGColHdrTypeFlags** were also removed.

17. [Changed] The **Header.Reset()** method now also clears the contents of column headers.

18. [Improved] The AutoFilterManager add-on now can be used with .NET Framework 2.0 (the minimal required version of .NET was 3.5 in previous builds).

19. [Fixed] When AutoFilterManager was attached to iGrid, the filter box did not open for columns that contained values of different numeric types within the same column.

20. [Fixed] If the user clicked any part of the current text cell being edited outside of the text rectangle, the editing process finished. In some earlier versions, the situation was even worse: the editing temporarily finished with the **BeforeCommitEdit** event and started again with the **RequestEdit** event. The fix implemented in this build eliminates all these problems: iGrid simply remains in edit mode without raising any events if the user clicks any part of the current text cell.

21. [Fixed] The column headers in the disabled control were drawn as pressed.

22. [Fixed] Double-clicking the column header divider of a merged column header that span several columns may have generated an exception.

23. [Fixed] Reading custom draw flags for a column header with the **iGColHdr.CustomDrawFlags** property gave improper results or may have generated an exception.

24. [Fixed] The **ModifierKeys** property of the **ColHdrMouseMove** event arguments was not filled properly.