10Tec iGrid.NET 12.0 What's New

Contents

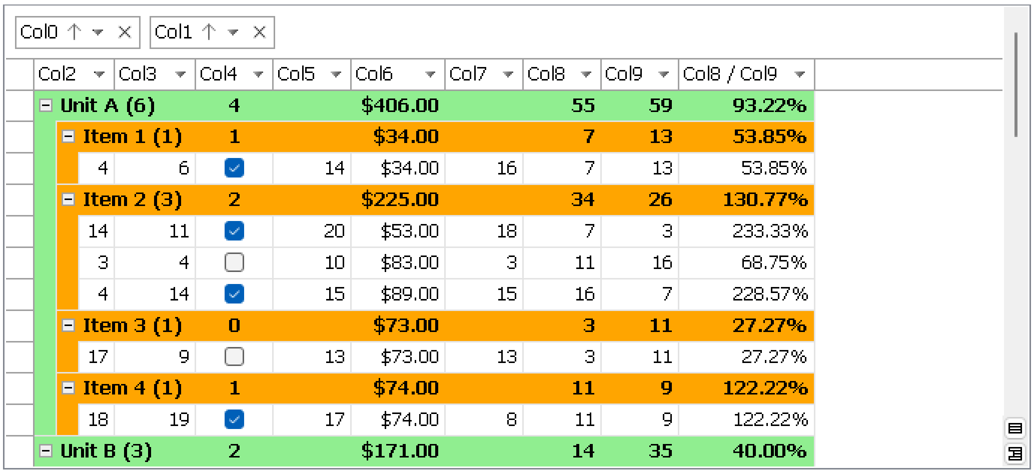
Most Important Improvements Briefly

- Two built-in group summary types: within columns and in group titles.
- Customizable item count in group titles.
- Optimized support for LINQ extension methods in iGrid collections.
- Twice as fast for-each loops on iGrid collections.
- Universal infrastructure for resetting properties of iGrid and other core components.
- Enhanced automatic column width and row height adjustment with smart handling of merged cells.

New Summary Feature

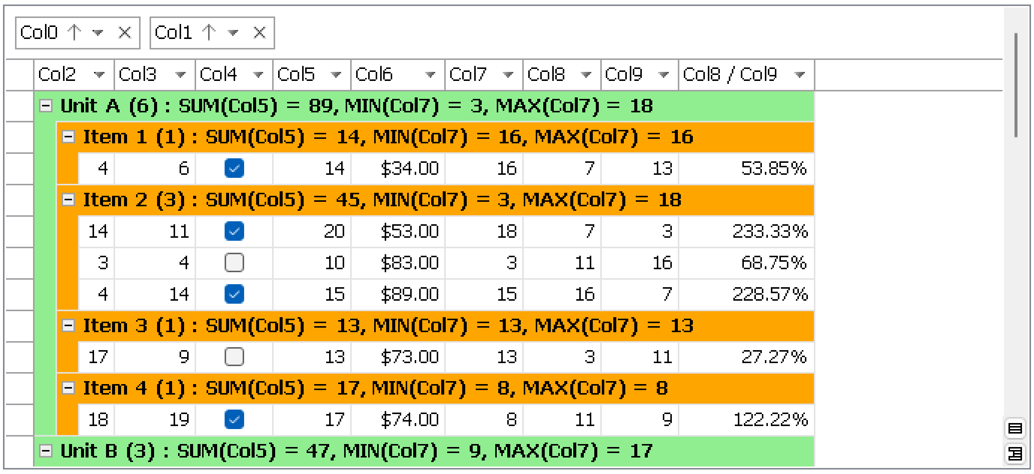
Introduction

The most important improvement in this major update of iGrid is the ability to calculate group summaries and display them in group rows. There are two kinds of group summaries: in-column group summaries and group header summaries. In-column group summaries are displayed in the cells of the columns for which they are calculated:


Col0 ↑ × Col1 ↑ ×

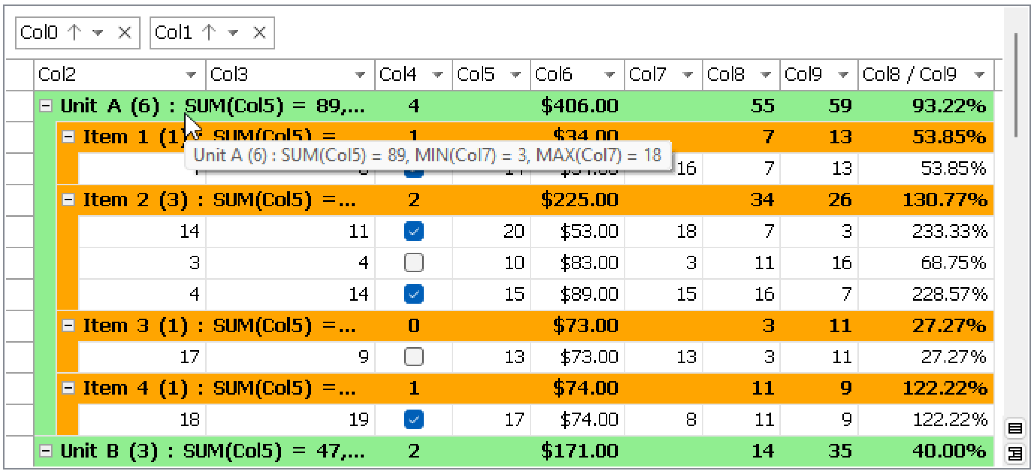
Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9	Col8 / Col9
⊟ Unit A (6)		4		$406.00		55	59	93.22%
⊟ Item 1 (1)		1		$34.00		7	13	53.85%
4	6	☑	14	$34.00	16	7	13	53.85%
⊟ Item 2 (3)		2		$225.00		34	26	130.77%
14	11	☑	20	$53.00	18	7	3	233.33%
3	4	☐	10	$83.00	3	11	16	68.75%
4	14	☑	15	$89.00	15	16	7	228.57%
⊟ Item 3 (1)		0		$73.00		3	11	27.27%
17	9	☐	13	$73.00	13	3	11	27.27%
⊟ Item 4 (1)		1		$74.00		11	9	122.22%
18	19	☑	17	$74.00	8	11	9	122.22%
⊟ Unit B (3)		2		$171.00		14	35	40.00%

Group title summaries are stored in a group row and are automatically displayed as part of group title unless you have hidden them:


Col0 ↑ × Col1 ↑ ×

Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9	Col8 / Col9
⊟ Unit A (6) : SUM(Col5) = 89, MIN(Col7) = 3, MAX(Col7) = 18								
⊟ Item 1 (1) : SUM(Col5) = 14, MIN(Col7) = 16, MAX(Col7) = 16								
4	6	☑	14	$34.00	16	7	13	53.85%
⊟ Item 2 (3) : SUM(Col5) = 45, MIN(Col7) = 3, MAX(Col7) = 18								
14	11	☑	20	$53.00	18	7	3	233.33%
3	4	☐	10	$83.00	3	11	16	68.75%
4	14	☑	15	$89.00	15	16	7	228.57%
⊟ Item 3 (1) : SUM(Col5) = 13, MIN(Col7) = 13, MAX(Col7) = 13								
17	9	☐	13	$73.00	13	3	11	27.27%
⊟ Item 4 (1) : SUM(Col5) = 17, MIN(Col7) = 8, MAX(Col7) = 8								
18	19	☑	17	$74.00	8	11	9	122.22%
⊟ Unit B (3) : SUM(Col5) = 47, MIN(Col7) = 9, MAX(Col7) = 17								

Both kinds of group summaries can be used within the same grid:


Col0 ↑ × Col1 ↑ ×

Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9	Col8 / Col9
⊟ Unit A (6) : SUM(Col5) = 89,...		4		$406.00		55	59	93.22%
⊟ Item 1 (1) : SUM(Col5) =		1		$34.00		7	13	53.85%
					16	7	13	53.85%
⊟ Item 2 (3) : SUM(Col5) =...		2		$225.00		34	26	130.77%
14	11	☑	20	$53.00	18	7	3	233.33%
3	4	☐	10	$83.00	3	11	16	68.75%
4	14	☑	15	$89.00	15	16	7	228.57%
⊟ Item 3 (1) : SUM(Col5) =...		0		$73.00		3	11	27.27%
17	9	☐	13	$73.00	13	3	11	27.27%
⊟ Item 4 (1) : SUM(Col5) =...		1		$74.00		11	9	122.22%
18	19	☑	17	$74.00	8	11	9	122.22%
⊟ Unit B (3) : SUM(Col5) = 47,...		2		$171.00		14	35	40.00%

Unit A (6) : SUM(Col5) = 89, MIN(Col7) = 3, MAX(Col7) = 18

The screenshots above demonstrate one more important enhancement in this release of iGrid: group titles now include item counts. The item counts in group titles are adjustable: you can change their format or even disable using the corresponding new properties introduced in this release.

The new group summary functionality is related to the automatic footer totals feature implemented in the previous versions of iGrid. Both infrastructures are based now on the same underlying types to ensure consistency in member names and expectations in functioning. Existing iGrid members associated with footer totals have been largely retained, though their names have been modified to align with the new infrastructure. The corresponding sections below describe these changes, which will assist in quickly upgrading the existing code.

In-column Group Summaries

1. [New] The new **GroupSummaryType** property of an iGrid column (the **iGCol** class) is used to define in-column group summaries. This property holds a value from the new **iGSummaryType** enumeration. The items of this enumeration are used to specify the aggregate function to be applied to column cells to calculate summary values or to configure iGrid for custom summary calculation. The **iGSummaryType** enumeration corresponds to the **iGAggregateFunctions** enumeration from the previous versions of iGrid in which it was used to define totals for the whole grid in the footer section. **iGSummaryType** includes all items from **iGAggregateFunctions** and adds a new member for custom summary values.

   If the **GroupSummaryType** property of a column is set to a value other than **iGSummaryType.None** (the default value), iGrid calculates the corresponding aggregate function for the column cells and displays the result in group rows. Below is an example of how to show totals for the cells in the 5th column in group rows:

   ```
   iGrid1.Cols[4].GroupSummaryType = iGSummaryType.Sum;
   ```

   The corresponding **GroupSummaryType** property was also added to the **iGColPattern** class used to define column patterns.

2. [New] In-column group summaries are calculated based on the cell values in the corresponding columns using an algorithm similar to totals calculation in the footer. The key points in this process are:

   - iGrid calculates summary values only for visible rows. These are rows with the **Visible** and **VisibleFiltered** properties set to True.
   - To get the numeric value of a cell for calculation, iGrid tries to interpret a cell value as a value of the **System.Decimal** type. The result of summary value calculation is also a value of the **System.Decimal** type.

   The summary value for a group is stored as the value of the corresponding cell in the group row. This means that you can read the calculated value like the traditional value of iGrid cells. For example, if you have configured iGrid to count the sum of the cell values in the 5th column and display these values in groups, you can retrieve the calculated sum for the very first group using the code below:

   ```
   decimal mySum = (decimal)iGrid1.CellValues[0, 4];
   ```

   Note that this approach allows you to calculate summary values for check box cells. In most cases check box states are stored as **Boolean**, **CheckState** or numeric values, and they can be treated as 1 for checked cells and 0 for unchecked cells. Thus, the summary type **Sum** for a column with check box cells will count the number of checked cells in the corresponding group.

3. [New] If a group row displays an in-column group summary, iGrid outputs the summary value inside the group row using almost the same algorithm used to render data cells. This allows you to use almost

all standard cell formatting features, such as color/font formatting, text alignment, etc. Column cell style inheritance also works for such cells, allowing you to apply the same format used for the column's data cells to the group summary values automatically.

The functionality of cells used to output in-column summaries inside group rows differs from data cells in the following aspects:

- The cell vertical grid line is not drawn but occupies the corresponding space as if it were visible.
- The **Type** and **TypeFlags** properties are not inherited from the column and row cell styles. The cell type is always considered text, and any type flags are not applied.

The foreground part of empty cells between cells with in-column summaries inside group rows is never rendered, but the background part is rendered like for data cells. This means you can specify their background color with the **BackColor** property of the corresponding cell or even use custom background drawing.

Group Title Summaries

1. [New] In contrast to in-column group summaries, you can define group summaries designated for displaying in group titles. This is done with the help of the new **GroupTitleSummaries** property of iGrid. **iGrid.GroupTitleSummaries** is a collection containing one or more summary definitions. Each summary definition is an instance of the new **iGSummaryDef** class with the following properties:

   - **Key** – a unique string to reference this summary in other iGrid members.
   - **ColIndex** – the grid column for which summary is calculated.
   - **SummaryType** – the summary type (a member of **iGSummaryType**).
   - **DisplayFormat** – the format string to get the text representation of summary value.

   Every summary definition must have a key. Keys are case-sensitive and cannot be null. The target column and the summary type must also be specified. The display format is optional and is used only when you want to display the summary calculation result in group titles.

   Group title summary definitions are created with the **Add()** method of the **iGrid.GroupTitleSummaries** collection. Below is an example demonstrating how to calculate totals for the column with the key "Data" and display them as "Amount = <value>" at the end of group titles:

   ```
   iGrid1.GroupTitleSummaries.Add(
     "data_total", "Data", iGSummaryType.Sum, "Amount = {0:#,##.00}");
   ```

   The first argument of the **Add()** call above, the string "data_total", is the key that can be used to retrieve the values of this summary for groups.

   iGrid.GroupTitleSummaries is an instance of the new **iGSummaryDefCollection** class. It includes properties and methods similar to traditional members in any collection of objects, such as the **Count** property or the **Clear()** method. In addition, it also provides a few properties defining how the string representing the list of calculated summary values is built. The members of the **iGSummaryDefCollection** class are described in greater detail below.

2. [New] To define a summary for group title, use one of the overloaded versions of the **Add()** method of the **iGrid.GroupTitleSummaries** collection:

```
Add(string key, int  colIndex, iGSummaryType summaryType)

Add(string key, string colKey, iGSummaryType summaryType)

Add(string key, int  colIndex, iGSummaryType summaryType,
    string displayFormat)

Add(string key, string colKey, iGSummaryType summaryType,
    string displayFormat)
```

The parameters of these overloads are used to initialize the properties of the created summary definition objects:

- The **key** parameter defines the summary key used to reference it. The value of this parameter cannot be null, and the keys of all summary definitions in the collection must be unique. Summary keys are case-sensitive.
- The **colIndex** parameter is used to define the column by its numeric index. An alternative way is to define the column by its string key with the **colKey** parameter. Note that the resulting created summary definition object (**iGSummaryDef**) will store only the corresponding numeric index of the column in its **ColIndex** property.
- The **displayFormat** parameter is used to specify the format applied to the raw calculated summary value to create its string representation on the screen. You can use the "{0}" placeholder for the summary value in such a format string as if you used the .NET **String.Format()** method to get the text representation of the object. If **displayFormat** is set to null, empty string or is not specified by calling the first 2 overloaded versions of the **Add()** method, the result of the calculation of the corresponding summary will not be displayed in group titles. It will be only stored in group rows for further use (building custom group titles with this summary value, calculations based on this summary, etc.)

Every **Add()** method returns an **iGSummaryDef** object that represents the created summary definition. This object is an informational object allowing you only to read the properties of the summary. You cannot create **iGSummaryDef** objects in your code using the New operator because these objects are tightly linked to the internal iGrid data and can be managed correctly only by iGrid.

The **Add()** methods create new summary definitions and add them to the end of the **GroupTitleSummaries** collection. The **iGrid.GroupTitleSummaries** collection also provides the set of **Insert()** methods with the corresponding overloads like the **Add()** method to create summary definitions and insert them before existing ones.

3. **iGrid.GroupTitleSummaries** offers the following collection-oriented features:

   - Retrieve the number of created summary definitions with the **Count** property.
   - Remove a summary definition by its key using the **RemoveByKey()** method.
   - Clear all summary definitions at once with the **Clear()** method.
   - Check if a summary with the specified key exists with the **ContainsKey()** method or find its index with the **IndexOfKey()** method.
   - Access a summary definition with a numeric index or string key using the [] indexer in C# or the **Item()** property in VB.
   - Enumerate summary definitions with a for-each loop.
   - Apply LINQ methods like **Select()** or **Any()** as the collection implements the generic **IEnumerable<T>** interface.

4. If the **DisplayFormat** property of a group title summary defined in **iGrid.GroupTitleSummaries** is not null or an empty string, the result of its calculation is automatically displayed at the end of group title. The string representation of the result is retrieved by applying the specified format to the raw summary value. iGrid constructs the resulting group title representation for all such summaries by combining their string representations into one string. The following string properties of the **iGrid.GroupTitleSummaries** object are used for building this resulting string:
    - **JoinValuesDelimiter** – the separator string between summary values (", " by default).
    - **JoinValuesPrefix** – the string added before the resulting string (null by default).
    - **JoinValuesSuffix** – the string added after the resulting string (null by default).
5. The calculated values of group title summaries are stored in the **SummaryValues** property of the new **GroupInfo** property of an **iGRow** row object. **SummaryValues** is an instance of the .NET **OrderedDictionary** class in which you can access items by their numeric indexes or string keys. These numeric indexes correspond to the indexes of summary definitions in the **iGrid.GroupTitleSummaries** collection. The strings keys are the keys of the summary definitions, and this is one of the reasons why they must be defined while creating summary definitions with the **Add()** or **Insert()** methods of the **GroupTitleSummaries** collection. A more detailed description of the **iGRow.GroupInfo** property and an example of its usage together with group title summaries can be found below in the section dedicated to changes in iGrid group rows.

Changes in Group Rows

1. [New][Change] If a group row displays group column summaries after setting the **GroupSummaryType** property of iGrid columns, the structure of the group row changes compared to the case when the group row does not show these new elements. If they are visible, the row text cell used to display the group title does not occupy all the space allocated for the group row: it becomes narrower and occupies the space up to the first cell containing the first group summary. The remaining space of the row is used to output group summary cells in columns for which the **GroupSummaryType** property is set and empty cells for other columns.

    Several important implications arise from these changes:

    - iGrid members that return cell boundaries and determine which cell is in the specified coordinates (**iGCell.Bounds**, **iGCellCollection.FromPoint()**, etc.) also behave accordingly. For example, the **Bounds** property for the row text cell returns the rectangle occupied by the group row title, which is less than the whole group row rectangle. The **iGrid.Cells.FromPoint()** method can be used to determine whether a group summary cell is displayed in the specified point, and so on.
    - Traditional mouse events for cells (**CellMouseEnter**, **CellMouseLeave**, **CellMouseDown**, **CellMouseUp** and the like) are raised for the row text cell and other individual cells inside group rows.
    - The **RequestCellToolTipText** event of iGrid is also raised for all cells inside group rows, allowing you to redefine texts of built-in tooltips for all those cells.
2. [New][Change][Code-Upgrade] In the previous versions of iGrid you could retrieve the bounds of a group row by reading the **Bounds** property or by calling the equivalent **GetBounds()** method of the row text cell belonging to the group row, for example:

    ```
    Rectangle myGroupRowBounds = iGrid1.Rows[2].RowTextCell.Bounds;
    ```

    In this version of iGrid a group row may display column summaries. In this case the bounds of the row text cell will be less than the row bounds, and the above approach will not work. To retrieve the correct bounds of the whole group row, two new members of the **iGRow** class were implemented:

```
Rectangle Bounds;
Rectangle GetBounds(bool includeLevel);
```

They work like the similar members of the **iGCell** class and allow you to retrieve the bounds of any row including group rows regardless of their contents. Thus, now you can use the following call to retrieve the bounds of a group row:

```
Rectangle myGroupRowBounds = iGrid1.Rows[2].Bounds;
```

3. [New][Change] If the user clicks a group row displaying group column summaries, iGrid changes cell selection and sets the current cell to the clicked cell, which can now be the row text cell with the group header text, a cell with the column summary value, or even empty cell. However, from the user's point of view, a group row is considered as one visual element and thus we cannot draw only the current clicked cell as selected without making the whole row selected. As a result, iGrid draws the whole group row containing the clicked cell as selected. This is another difference between the new iGrid and the previous versions, in which a group row displayed only the row text cell and only this row text cell could be made current and selected.

4. [New][Removed][Code-Upgrade] This release of iGrid introduces the new **GroupTitleOptions** object property allowing you to configure the process of generation of group titles created during automatic grouping. This object property has the following 2 sub-properties:

   - **AddColTitlePrefix** - a Boolean property specifying whether column title is added before group value in group title. The default value is True.
   - **ItemCountFormatString** – a string property defining the format string used to create the text representation of item counter after the group value in group title. The default value is "({0})", where {0} is the placeholder for the item count value.

   The **iGrid.GroupTitleOptions.AddColTitlePrefix** property fully corresponds to the behavior of the **iGrid.PrefixGroupValue** property in the previous versions of iGrid and replaces it. Because of this, **iGrid.PrefixGroupValue** property was removed in this version of iGrid.

   The **iGrid.GroupTitleOptions.ItemCountFormatString** property allows you to remove the item counter in group rows created automatically. To do this, set it to null or an empty string. If you do this, the process of group title generation will fully correspond to what iGrid did in the previous versions.

5. [New] The **iGRow** class implements the new **GroupInfo** property that stores the information about the group row represented with an **iGRow** instance. The **iGRow.GroupInfo** property returns an instance of the new **iGGroupInfo** class with the following properties:

   - **GroupValue** – the group value used to create the group.
   - **GroupValueText** – the text representation of the group value in the group title.
   - **ItemCount** – the number of items in the group.
   - **SummaryValues** – the values of group title summaries.

   Below is an example demonstrating how to display these values for the very first group row created during automatic grouping:

```
private void buttonDisplayFirstGroupInfo_Click(object sender, EventArgs e)
{
  iGGroupInfo gi = iGrid1.Rows[0].GroupInfo;

  MessageBox.Show(
    $"Group Value = {gi.GroupValue} ({gi.GroupValue.GetType()})\n" +
    $"Group Value Text = '{gi.GroupValueText}'\n" +
    $"Item Count = {gi.ItemCount}");
}
```

Instances of the **iGGroupInfo** class can be also retrieved with the **GroupInfo** properties of the event arguments of the existing **AfterAutoGroupRowCreated** event and the new **CustomGroupTitle** event described below.

The **iGRow.GroupInfo** property is read-only and is populated by iGrid only for group rows created automatically during grouping. It contains a null reference for rows of other types.

6. [New] iGrid raises the new **CustomGroupTitle** event before it sets the group title in a group row created automatically and allows you to modify this title if required. The arguments of this event provide you with the main information about the created group row – its index (the **RowIndex** property), the information about group (the **GroupInfo** property of the **iGGroupInfo** class), and the grouped column (the **ColIndex** and **ColTitle** properties). The read-write **GroupTitle** property contains the default group title and is used to modify it if required.

   This event is raised after calculated summary values for the group row are set, which gives you an ability to read them in an event handler of this event. As an example, let's consider a grid in which the first column is a check box column used to select rows for group summary calculation. You can define the invisible group title summary to calculate the number of checked items in a group like this:

```
iGrid1.GroupTitleSummaries.Add("checked", 0, iGSummaryType.Sum);
```

   Having this group summary, you can output the number of selected items and the number of all items like "Unit 1 (selected 4 of 6)" in group rows created automatically using the following event handler of the **CustomGroupTitle** event:

```
private void iGrid1_CustomGroupTitle(
  object sender, iGCustomGroupTitleEventArgs e)
{
  e.GroupTitle = $"{e.GroupValue} (selected
{e.GroupInfo.SummaryValues["checked"]} of {e.GroupInfo.ItemCount})";
}
```

7. [Change][Code-Upgrade] In the previous versions of iGrid group row styles defined in the **iGrid.GroupRowLevelStyles** property were applied to row text cells that stored the contents of the whole group row. Starting from this release of iGrid **GroupRowLevelStyles** are applied to whole rows through the **iGRow.CellStyle** property. This change allows you to apply the same formatting to all cells inside group rows – the row text cell displaying the group title, group summary cells, etc.

   The values of all properties of the cell styles in **iGrid.GroupRowLevelStyles** except **BackColor** are set to the special **NotSet** value. This allows iGrid to inherit column cell formatting in group row summary cells unless it has not been changed for individual cells inside group rows.

   If you do not change individual properties of group rows or row text cells displayed in group rows in iGrid event handlers (such as **AfterAutoGroupRowCreated**), existing code will work like earlier without any changes.

8. [Removed] The **iGrid.ScrollGroupRows** property was removed because it caused compatibility problems with some iGrid options. The ability to make the contents of group rows non-scrollable was also incompatible with the new group summary functionality.
9. [Renaming][Code-Upgrade] The **GroupRowTextSelected** and **GroupRowTextNotSelected** properties of the **iGrid.UIStrings** object property were renamed to **GroupValueTextSelected** and **GroupValueTextNotSelected** respectively to correspond to the new terminology used in changed group row functionality.

Custom Group and Footer Summaries

1. [New] To implement custom summaries in group rows and footer cells, the **iGSummaryTypes** enumeration provides the **Custom** member. If you set the **GroupSummaryType** property of a column object (**iGCol**) or the **SummaryType** property a group title summary definition object (**iGSummaryDef**) to **iGSummaryTypes.Custom**, iGrid raises the new **CustomGroupSummary** event so that you can provide a custom summary value for a group row in all appropriate cases — creation of an automatic group row during grouping, group summary updates after editing a cell, etc.

   The **CustomGroupSummary** event fires after all defined group summaries other than **Custom** have been automatically calculated. This allows you to calculate custom summary values based on the summary values calculated by iGrid. As an example, let's consider a grid in which sums of cell values by groups are calculated for columns with indexes 4 and 5. We need to calculate a custom group summary showing the ratio of these two values and display it in column with the index 6.

   The definition of group summaries for our grid should look as follows:

```
iGrid1.Cols[4].GroupSummaryType = iGSummaryType.Sum;
iGrid1.Cols[5].GroupSummaryType = iGSummaryType.Sum;
iGrid1.Cols[6].GroupSummaryType = iGSummaryType.Custom;
```

   The event handler of the **CustomGroupSummary** event implementing our task may look like this:

```
private void iGrid1_CustomGroupSummary(
  object sender, iGCustomGroupSummaryEventArgs e)
{
  If (e.ColIndex == 6)
  {
    decimal myValue1 = Convert.ToDecimal(iGrid1.CellValues[e.RowIndex, 4]);
    decimal myValue2 = Convert.ToDecimal(iGrid1.CellValues[e.RowIndex, 5]);

    if (myValue2 != 0)
      e.Value = myValue1 / myValue2;
  }
}
```

   The similar new event for footer cells, **CustomFooterSummary,** was also introduced in this release of iGrid. It is raised for footer cells with the **SummaryType** property set to **iGSummaryTypes.Custom**. This event is also raised after all footer summaries other than **Custom** were automatically calculated by iGrid, allowing you to calculate custom footer summary values based on the summary values calculated by iGrid.

   Let's continue with our example. If we want to display total summaries for columns 4, 5, and 6 for the grid we described above, we will define the footer summaries as follows:

```
iGrid1.Footer.Cells[0, 4].SummaryType = iGSummaryType.Sum;
iGrid1.Footer.Cells[0, 5].SummaryType = iGSummaryType.Sum;
iGrid1.Footer.Cells[0, 6].SummaryType = iGSummaryType.Custom;
```

Then the following event handler of the **CustomFooterSummary** event will implement what we need:

```
private void iGrid1_CustomFooterSummary(
  object sender, iGCustomFooterSummaryEventArgs e)
{
  If (e.ColIndex == 6)
  {
    decimal myValue1 = iGrid1.Footer.Cells[e.RowIndex, 4].Total;
    decimal myValue2 = iGrid1.Footer.Cells[e.RowIndex, 5].Total;

    if (myValue2 != 0)
      e.Value = myValue1 / myValue2;
  }
}
```

The arguments of the **CustomGroupSummary** event are represented with an instance of the new **iGCustomGroupSummaryEventArgs** class. Its **RowIndex** and **ColIndex** fields allow you to know the index of the group row and the index of the grid column respectively for which the custom summary is calculated. The **SummaryKey** field of the event arguments contains the key of the group title summary or null if the event is raised for a custom in-column group summary. The read-write **Value** field is used to tell iGrid the custom summary value calculated by the developer.

The arguments of the **CustomFooterSummary** event are provided with an instance of the new similar **iGCustomFooterSummaryEventArgs** class, except it does not have the **SummaryKey** field used only for group summaries.

Other Improvements

In the previous versions of iGrid there were only footer summaries referred to simply as "totals" in all relevant control members. Now in iGrid there are 2 types of summaries - footer summaries and group summaries. Because of this we had to use the term "FooterSummary" and "GroupSummary" in the corresponding members. As a result, some existing members were renamed to avoid confusion and to match the enhanced iGrid summary infrastructure.

1. [New] The following new members related to group summaries were implemented in iGrid:
    - The Boolean **AutoUpdateGroupSummaries** property that controls whether iGrid automatically recalculates group summaries during relevant events such as changes in cell values or row visibility. The default value is True.
    - The **UpdateGroupSummaries()** method that recalculates all group summaries and displays them in group rows. This method is useful if you need to update group summaries in special cases when iGrid does not do that automatically, or if you turned off automatic group summaries updates with the **iGrid.AutoUpdateGroupSummaries** property and want to launch group summaries recalculation manually from code.
    - The **GroupSummariesUpdated** event that is raised after iGrid updated group summaries (when automatic group summaries updates are on or you called the **UpdateGroupSummaries()** method from code).
2. [New][Renaming][Code-Upgrade] To provide a similar set of members for footer summaries, the following changes were implemented in iGrid:
    - The existing **CalculateTotals** property was renamed to **AutoUpdateFooterSummaries**.

- The **UpdateFooterSummaries()** method that recalculates all footer summaries and displays them in the footer section was implemented.
- The **FooterTotalsUpdated** event raised after iGrid updated footer totals was renamed to **FooterSummariesUpdated**.

3. [New][Renaming][Code-Upgrade] By default, iGrid includes cells that are in visible rows in the calculation of footer summaries and group summaries. These are all rows with the **Visible** and **VisibleFiltered** properties set to True. You can further filter this set of cells and specify which cells to include in the calculation of summaries of each type. For group summaries, this is done with the help of the new **IncludeCellInGroupSummary** event. Below is an example demonstrating how to include cells from rows having the checked check box in the 5th column into group summary calculation:

```
private void iGrid1_IncludeCellInGroupSummary(
  object sender, iGIncludeCellInGroupSummaryEventArgs e)
{
  CheckState myCheckState = (CheckState)iGrid1.CellValues[e.RowIndex, 4];
  e.DoDefault = myCheckState == CheckState.Checked;
}
```

   The similar event for footer summaries called **IncludeRowInTotalsCalculation** already existed in the previous version of iGrid. It was renamed to **IncludeCellInFooterSummary** in this release of iGrid to correspond to the general scheme of member names for group summaries and footer summaries. The new event name using the term "cell" instead of "row" more accurately conveys the purpose of this event.

4. [New][Renaming][Code-Upgrade] The **RecalculateTotals** event, which was raised whenever iGrid needed to update footer totals in the previous versions, was renamed to **FooterSummariesNeedUpdate**. A similar event for group row summaries, **GroupSummariesNeedUpdate**, was introduced in this release of iGrid.

Recap: Renames of iGrid Members Related to Summaries

- The **iGAggregateFunction** enumeration is now **iGSummaryType**.
- **iGFooterCell**'s **AggregateFunction** property is now **SummaryType**.
- iGrid's **IncludeRowInTotalsCalculation** event is now **IncludeCellInFooterSummary**.
- iGrid's **CalculateTotals** property is now **AutoUpdateFooterSummaries**.
- iGrid's **RecalculateTotals** event is now **FooterSummariesNeedUpdate**.
- iGrid's **FooterTotalsUpdated** event is now **FooterSummariesUpdated**.

Better LINQ Support and For-each Loop Optimizations

1. [New][Enhancement] All iGrid collections now implement the generic **IEnumerable<T>** interface to provide support for LINQ extension methods out of the box. These are**: iGrid.Cols**, **iGrid.Rows**, **iGHeader.Rows**, **iGFooter.Rows**, **iGrid.Cells**, **iGCol.Cells**, **iGRow.Cells**, **iGHeader.Cells**, **iGHdrRow.Cells**, **iGFooter.Cells**, **iGFooterRow.Cells**, **iGrid.MergedCells**, **iGrid.SelectedCells**, **iGrid.SelectedRows**, **iGrid.HScrollBar.CustomButtons**, **iGrid.VScrollBar.CustomButtons**, **iGDropDownList.Items**, **iGrid.GroupTitleSummaries**.

   As a demonstration of how this feature can be used, look at the following example showing how to use the **Sum()** extension method for enumerable collections to calculate the total for the Data column containing decimal values:

```
decimal total = iGrid1.Cols["Data"].Cells.Sum(
  cell => Convert.ToDecimal(cell.Value));
```

2. [Optimization] The implementation of the generic **IEnumerable<T>** interface in iGrid collections results in faster execution of for-each loops. Our tests show that loops like

```
foreach (iGCell cell in iGrid1.Cells)
{
  // doing something with the cell
}
```

   can work 2 times faster now.

3. [New] All iGrid classes representing objects used in iGrid collections implement the generic **IEquatable<T>** interface. This allows objects of these classes to be compared correctly and is used by LINQ operators such as **Contains()**. These classes are: **iGCell**, **iGCol**, **iGRow**, **iGHdrRow**, **iGFooterRow**, **iGColHdr**, **iGFooterCell**, **iGDropDownListItem**, **iGScrollBarCustomButton**, **iGSummaryDef**.

4. [New] The following collection classes implement the generic **IList<T>** interface: **iGCellCollection**, **iGSelectedCellsCollection**, **iGSelectedRowsCollection**. The **IList<T>.Contains()** method in these classes was highly optimized to provide the best performance in LINQ queries. This means that the LINQ **Contains()** operator for the **iGrid.Cells**, **iGrid.MergedCells**, **iGrid.SelectedCells** and **iGrid.SelectedRows** collections is executed very fast now.

5. [New] All iGrid collections now implement **IEnumerable<T>**, so you can use the **Reverse()** LINQ method to enumerate items in reverse order as shown below:

```
foreach (iGRow row in iGrid1.SelectedRows.Reverse())
{
  // doing something with the row
}
```

   However, the standard implementation of this **Reverse()** extension method first creates a copy of the whole collection and then iterates it in reverse order instead of simply enumerating the input collection in reverse order. It may result in creating an abnormally large number of objects in a short moment of time and increase the code execution time due to this unnecessary copy operation.

   To solve this problem, iGrid provides the equivalent **ReverseOptimized()** method for all major collections that may contain a large number of objects. These are **iGrid.Cells**, **iGrid.Cols**, **iGrid.Rows**, **iGCol.Cells**, **iGRow.Cells**, **iGrid.MergedCells**, **iGrid.SelectedCells** and **iGrid.SelectedRows**. The iGrid implementation is free from the drawbacks described above and simply returns objects in reverse order ready for enumeration. According to tests performed, this optimized implementation can run 3 times faster than the standard one.

   Use iGrid's **ReverseOptimized()** just like the standard LINQ **Reverse()** method, as shown below:

```
foreach (iGRow row in iGrid1.SelectedRows.ReverseOptimized())
{
  // doing something with the row
}
```

New Universal Property Reset Infrastructure

This release of iGrid introduces new universal tools to reset properties to default values at run time. This can be done with the help of the static methods of the new **iGPropertyManager** class implemented in the core iGrid.NET assembly (TenTec.Windows.iGridLib.dll). The revised and enhanced internal infrastructure used to reset properties at design time is now also used for resetting properties at run time and works for the following components: **iGrid**, **iGDropDownList**, **iGCellStyle**, **iGColHdrStyle**, **iGAutoFilterManager**, **iGPrintManager**.

One good example of when these tools can be useful is given below. Many apps provide the user with the ability to change the look and behavior of some parts of iGrid and revert them to their original state. For example, you may provide your users with the option to adjust the width and color of the grid lines in iGrid and revert them back to the default values. If you have a check box for such a setting, the corresponding event handler may look like the following code snippet:

```
private void checkBoxThickGridLines_CheckedChanged(object sender, EventArgs e)
{
  if (checkBoxThickGridLines.Checked)
  {
    fGrid.GridLines.Horizontal.Width = 2;
    fGrid.GridLines.Horizontal.Color = Color.Red;
    fGrid.GridLines.HorizontalLastRow.Width = 3;
    fGrid.GridLines.HorizontalLastRow.Color = Color.DarkRed;
    fGrid.GridLines.Vertical.Width = 2;
    fGrid.GridLines.Vertical.Color = Color.Red;
    fGrid.GridLines.VerticalLastCol.Width = 3;
    fGrid.GridLines.VerticalLastCol.Color = Color.DarkRed;
  }
  else
  {
    fGrid.GridLines.Horizontal.Width = _savedHorizontalWidth;
    fGrid.GridLines.Horizontal.Color = _savedHorizontalColor;
    fGrid.GridLines.HorizontalLastRow.Width = _savedHorizontalLastRowWidth;
    fGrid.GridLines.HorizontalLastRow.Color = _savedHorizontalLastRowColor;
    fGrid.GridLines.Vertical.Width = _savedVerticalWidth;
    fGrid.GridLines.Vertical.Color = _savedVerticalColor;
    fGrid.GridLines.VerticalLastCol.Width = _savedVerticalLastColWidth;
    fGrid.GridLines.VerticalLastCol.Color = _savedVerticalLastColColor;
  }
  fGrid.Invalidate();
}
```

Needless to say that you also need to declare all these _saved* variables in your form and initialize them with the default values of the corresponding properties when your form starts. The new universal property reset tools allow you to avoid having such variables and to reset any iGrid property to its default state with one call to the **iGPropertyManager.ResetProperty()** method. Having this method, we can rewrite the else part in our event handler as follows:

```
private void checkBoxThickGridLines_CheckedChanged(object sender, EventArgs e)
{
  if (checkBoxThickGridLines.Checked)
  {
    fGrid.GridLines.Horizontal.Width = 2;
    fGrid.GridLines.Horizontal.Color = Color.Red;
    fGrid.GridLines.HorizontalLastRow.Width = 3;
    fGrid.GridLines.HorizontalLastRow.Color = Color.DarkRed;
    ... (other custom grid line settings)
  }
  else
  {
    iGPropertyManager.ResetProperty(fGrid, "GridLines");
  }
  fGrid.Invalidate();
}
```

The **iGPropertyManager** class implements the following static methods:

```
void ResetProperty(object obj, string property)

bool CanResetProperty(object obj, string property)

IEnumerable<string> GetResettableProperties(object obj)

bool? IsPropertyModified(object obj, string property)
```

These methods are described in greater detail below.

1. The **ResetProperty()** method can be used to reset any property to its default value, for example:

```
// Reset a property of an enumeration type (iGBorderStyle):
iGPropertyManager.ResetProperty(iGrid1, "BorderStyle");

// Reset a complex object property of the iGGridLines type:
iGPropertyManager.ResetProperty(iGrid1, "GridLines");

// Reset a property storing an array of iGCellStyle items:
iGPropertyManager.ResetProperty(iGrid1, "GroupRowLevelStyles");
```

This method accepts an object whose property is to be reset, along with the property's name specified as a string. The property name is case-sensitive. If the method can't find the specified property, it throws an **ArgumentException** with the message "Property <property> not found".

If you want to provide compile-time check for property names specified in **ResetProperty()** method calls, use the **nameof** operator in C# or **NameOf** in VB, for example:

```
iGPropertyManager.ResetProperty(iGrid1, nameof(iGrid1.BorderStyle));
iGPropertyManager.ResetProperty(iGrid1, nameof(iGrid1.GridLines));
iGPropertyManager.ResetProperty(iGrid1, nameof(iGrid1.GroupRowLevelStyles));
```

The **ResetProperty()** method can be called for complex object properties like **iGrid.GridLines**. In this case it resets the sub-properties of such properties. You can also call **ResetProperty()** for individual sub-properties of a complex property. For example, the following call will reset the **Vertical** property of the **iGrid1.GridLines** property to its default value:

```
iGPropertyManager.ResetProperty(iGrid1.GridLines, "Vertical");
```

The equivalent call with compile time check of the property name will be

```
iGPropertyManager.ResetProperty(
  iGrid1.GridLines, nameof(iGrid1.GridLines.Vertical));
```

2. The **CanResetProperty()** method can be called to know whether a property can be reset. For example, the **IsDocInPrintableArea** property of **iGPrintManager** is a read-only Boolean property indicating whether the document margins are in the area where the selected printer can print. Thus, the following expression will return False:

```
iGPropertyManager.CanResetProperty(iGPrintManager1, "IsDocInPrintableArea")
```

   Note that some read-only properties can still be reset. There are many object properties like **iGrid.GridLines** that are read-only itself (you cannot replace the whole object property value in the **iGrid.GridLines** property), but the sub-properties of these properties can be reset. The **CanResetProperty()** method returns True for such object properties indicating that their sub-properties can be reset.

3. The **GetResettableProperties()** method returns all properties of a component or object that can be reset. These properties are returned as an enumerable string collection that can be used in a for-each loop or with LINQ extension methods. This allows you to build simple loops to reset a whole component to its default state. Below is an example of how you can set an iGrid instance to the default state as if it had just been created:

```
foreach (var prop in iGPropertyManager.GetResettableProperties(iGrid1))
{
  iGPropertyManager.ResetProperty(iGrid1, prop);
}
```

   Pay attention to the fact that iGrid inherits many properties from the **System.Windows.Forms.Control** class, such as **Location** and **Size**. The loop above will reset these properties too because these properties are considered iGrid properties and can be reset. If you want to keep the position of iGrid on a form, you can use LINQ's **Except()** to filter the property list like in the following code:

```
string[] layoutProps = new string[] {
  "Anchor", "Dock", "Location", "Size", "Margin" };

foreach (var prop in
  iGPropertyManager.GetResettableProperties(iGrid1).Except(layoutProps))
{
  iGPropertyManager.ResetProperty(iGrid1, prop);
}
```

4. You can determine whether a resettable property has a non-default value with the help of the **IsPropertyModified()** method of the **iGPropertyManager** class. This method returns a Nullable Boolean value indicating whether a property was modified (True) or not (False). If the specified property does not provide the corresponding infrastructure tools to know its modification status, the method returns null (Nothing in VB). Note that the ability to know property modification status is based on the same internal tools as the ability to reset a property, and thus it can be also checked with the **iGPropertyManager.CanResetProperty()** method.

Other Changes and Enhancements

Automatic Adjustment of Column Widths and Row Heights

1. [New][Change][Code-Upgrade] This release of iGrid introduces three methods for automatically adjusting column widths and row heights when it contains merged cells. You can separately specify

the method used by iGrid for columns and rows with the new **AutoWidthColSpanHandling** and **AutoHeightRowSpanHandling** properties of iGrid. Both properties accept one of the values from the following **iGAutoSizeSpanHandling** enumeration introduced in this release:

```
public enum iGAutoSizeSpanHandling
{
  Ignore = 0,             // Ignore merged cells during auto-sizing
  SizeRootUsingSpan = 1,  // Size root using info from entire span
  SizeRootOnly = 2        // Size root using only root's own info
}
```

To demonstrate how these methods work, let's consider the following grid with several merged cells spanning over columns:

ID	Name	Age	Salary
0	This is a test	10	100
1	This is a test	11	110
2	This is a longer test		
3	This is a test	13	130
4	This is a test	14	140
5	This is also a longer test		
6	This is a test	16	160
7	This is a very very very long...		
8	This is a test	18	180
9	This is a test	19	190
10	This is a test	20	200

The default value for the **iGrid.AutoWidthColSpanHandling** property is **iGAutoSizeSpanHandling.Ignore**. In this case, if the user double-clicks the divider of the Name column or the developer calls the **iGCol.AutoWidth()** method to automatically adjust the column's width, iGrid adjusts the width of the column considering only non-merged cells in the column:

ID	Name	Age	Salary
0	This is a test	10	100
1	This is a test	11	110
2	This is a longer test		
3	This is a test	13	130
4	This is a test	14	140
5	This is also a longer test		
6	This is a test	16	160
7	This is a very very very l...		
8	This is a test	18	180
9	This is a test	19	190
10	This is a test	20	200

If you set the **AutoWidthColSpanHandling** property to **SizeRootUsingSpan**, iGrid sets the width of the Name column to the minimal value at which the texts of the merged cells become fully visible:

ID	Name	Age	Salary
0	This is a test	10	100
1	This is a test	11	110
2	This is a longer test		
3	This is a test	13	130
4	This is a test	14	140
5	This is also a longer test		
6	This is a test	16	160
7	This is a very very very long test		
8	This is a test	18	180
9	This is a test	19	190
10	This is a test	20	200

The last auto-sizing method enabled with the **SizeRootOnly** member causes iGrid to enlarge the Name column so that its width is enough to display the texts of merged cells without clipping:

ID	Name	Age	Salary
0	This is a test	10	100
1	This is a test	11	110
2	This is a longer test		
3	This is a test	13	130
4	This is a test	14	140
5	This is also a longer test		
6	This is a test	16	160
7	This is a very very very long test		
8	This is a test	18	180
9	This is a test	19	190
10	This is a test	20	200

The **Ignore** mode is selected by default because it mirrors behavior in widely used applications such as Microsoft Excel, aligning with user expectations. The **SizeRootOnly** mode corresponds to what iGrid did in the previous versions, and you should set it explicitly if you need the same behavior in this release of iGrid.

The **SizeRootUsingSpan** mode is the smartest mode that allows you to set the widths of all grid columns to some optimal values to display the texts of all cells without clipping. You can do this in one call to the **iGrid.Cols.AutoWidth()** method because it also respects the value of the **AutoWidthColSpanHandling** property of iGrid.

Everything written above also applies to grid rows, with the difference that the described methods work in the "vertical direction".

2. [Enhancement] The **AutoWidth()** method of the **iGCol** class, whose instances represent iGrid columns, was enhanced.

   When this method is called for a normal column (not the row text column), it considers in-column group summaries that can be drawn in group rows. The same enhancement works when the user double-clicks the column divider of a column to automatically adjust its width.

   When this method is called for the row text column, it performs automatic adjustment of the widths of normal columns so that row text cells and/or group titles are fully displayed using the minimal available widths of the corresponding columns. To specify whether to process only group rows or row text cells, you can use 2 new overloaded versions of the method that take the parameter of the new **iGRowTextKinds** enumeration:

```
public void AutoWidth(iGRowTextKinds rowTextKinds)

public void AutoWidth(int rowIndex, int rowCount,
  bool allowDecrease, iGRowTextKinds rowTextKinds)
```

The **iGRowTextKinds** enumeration is defined as follows:

```
[Flags]
public enum iGRowTextKinds
{
  None = 0,
  GroupRow = 1,
  RowTextCell = 2,
  All = GroupRow | RowTextCell
}
```

3. [Enhancement] The **AutoHeight()** method of the **iGRow** class, whose instances represent iGrid rows, was enhanced. It supports new group rows with summaries introduced in this release and allows you to adjust row height taking into account various formatting that can be applied to the row text cell containing group title and cells displaying in-column group summaries. The same enhancement works when the user double-clicks the row divider of a row to automatically adjust its height.

4. [New][Change] When the user double-clicks the divider of a column header or you call the **iGCol.AutoWidth()** method to automatically adjust the width of a column, iGrid decides whether to process a column's cell based on the visibility of the corresponding row. The new **AutoWidthColCheckRowVisibility** property of iGrid allows you to control how iGrid checks row visibility during automatic column width adjustment.

   The **AutoWidthColCheckRowVisibility** property accepts a combination of flags from the following new enumeration:

```
[Flags]
public enum iGRowVisibilityProperties
{
  None = 0,
  Visible = 1,
  VisibleFiltered = 2,
  VisibleParentExpanded = 4,
  All = 7
}
```

   The **Visible**, **VisibleFiltered** and **VisibleParentExpanded** flags correspond to the 3 row properties that define row visibility. If a flag is set, iGrid will check the value of the corresponding row property while adjusting the column width.

   In most cases a row should be skipped if it is currently not visible, except the case when the row belongs to a collapsed group row or tree node. The default value of the **AutoWidthColCheckRowVisibility** property is the combination of the **Visible** and **VisibleFiltered** flags, which corresponds to the described typical behavior. In other words, if a row is a child row of a group row or tree node, it will be processed during automatic adjustment of the column width regardless of the collapsed state of the row's parent because the **iGRow.VisibleParentExpanded** property will not be checked.

   The described default behavior differs from the behavior of iGrid in the previous versions when it always ignored child rows inside collapsed group rows or tree nodes while adjusting column width. To get the same behavior in the new release of iGrid, use the following setting:

```
iGrid1.AutoWidthColCheckRowVisibility = iGRowVisibilityProperties.All;
```

5. [New] The **GetPreferredRowHeight()** method of iGrid is often used to retrieve the optimal row height. iGrid implements the following new overloaded version of this method in which you can specify the number of text lines in cells for which iGrid should calculate the optimal row height:

```
public int GetPreferredRowHeight(int textLineCount, bool hasImage)
```

AutoFilterManager Add-on Enhancements

1. [New] Now you can check whether a filter is applied to a column using the new Boolean **IsFilterApplied** property of the **iGColAutoFilter** object returned by the **iGAutoFilterManager.ColAutoFilter()** method, for example:

```
bool isCol2Filtered = iGAutoFilterManager1.ColAutoFilter(2).IsFilterApplied;
```

2. [Enhancement] Changing the values of the sub-properties of the object property **iGAutoFilterManager.FilterButtonColors** leads to immediate update of the filter buttons inside the grid column headers.
3. [Removed] The **iGAutoFilterManager.FilterButtonColors.BackColor** property was removed because it is no longer used in drawing filter buttons.

PrintManager Add-on Enhancements

1. [New] The **iGPrintManager** component implements the new **RowPrinted** event. This event occurs after a row was printed. The properties of the event arguments object allow you to know the grid from which the row was printed (the **Grid** property), the row index (the **RowIndex** property) and the page number on which the row was printed (the **PageNumber** property).

Design Time Improvements

1. [Enhancement] iGrid now provides enhanced internal infrastructure that makes the Reset command in the Visual Studio Property Grid work for the following components and their object properties:
   - **iGrid: Cols, DefaultAutoGroupRow, Footer, FrozenArea, GridLines, GroupBox, GroupTitleOptions, Header, HScrollBar, RowHeader, Rows, RowTextCol, ScrollBarSettings, SearchAsType, TreeLines, VScrollBar, UIStrings**.
   - **iGAutoFilterManager**: **FilterButtonColors**, **FilterBoxColors**, **UIStrings**.
   - **iGPrintManager**: **DocumentHeader**, **DocumentFooter**, **PageHeader**, **PageFooter**, **PrintPreviewSettings**, **UIStrings**.
   - **iGDropDownList**: **Items**, **SearchAsType**.
2. [Enhancement] You can now view and change the **FilterKeepCurRow** and **AutoCancel** sub-properties of the **iGrid.SearchAsType** property in the Visual Studio property grid.
3. [Enhancement] You can now view and change the **iGrid.VScrollBar.WidthOverride** and **iGrid.HScrollBar.HeightOverride** properties in the Visual Studio property grid. The default values of these properties (null) are also no longer serialized to the code generated by the WinForms Designer.

Miscellaneous

1. [New] iGrid implements the new **CustomGroupValueText** event that can be used together with the **CustomGroupValue** event to provide custom text representations for custom group values. The **RowIndex**, **ColIndex**, and **Value** fields of the event arguments of the new event allow you to identify the group row, column and custom group value for which iGrid obtains the custom text representation.

The default text representation is passed in the **Text** argument and can be changed to any desired string.

2. [New] The following read-only cell properties for retrieving effective values of the corresponding cell properties were implemented: **EffectiveCustomEditor**, **EffectiveEmptyStringAs**, **EffectiveEnabled**, **EffectiveFlags**, **EffectiveFormatProvider**, **EffectiveFormatString**, **EffectiveMaxInputLength**, **EffectiveSelectable**, **EffectiveValueType**.

3. [New] The **iGrid.Cols** collection implements the new **GetKey(int colIndex)** method that can be used to retrieve the key of the column with the specified index. This method, in contrast to the expression **iGrid.Cols[int colIndex].Key**, does not create a new unneeded **iGCol** object. This results in less load on the Garbage Collector, especially if column keys are checked in event handlers executed very often (custom drawing, dynamic cell contents and the like), for example:

```
private void fGridOutlook_CellDynamicContents(
  object sender, iGCellDynamicContentsEventArgs e)
{
  if (fGridOutlook.Cols[e.ColIndex].Key == "Flag")
  {
    // provide dynamic contents
  }
}
```

   It is advised that you write such event handlers with the new **GetKey()** method like this:

```
private void fGridOutlook_CellDynamicContents(
  object sender, iGCellDynamicContentsEventArgs e)
{
  if (fGridOutlook.Cols.GetKey(e.ColIndex) == "Flag")
  {
    // provide dynamic contents
  }
}
```

   The similar **GetKey(string rowIndex)** method was implemented for rows in the **iGrid.Rows** collection.

4. [New] iGrid implements the new static **GetDefaultGroupRowLevelStyles()** method. You can use it to set the **iGrid.GroupRowLevelStyles** property to the default state like this:

```
iGrid1.GroupRowLevelStyles = iGrid.GetDefaultGroupRowLevelStyles();
```

5. [New] The **iGrid.Rows** collection property implements the **RemoveByKey(string key)** method that can be used to remove a row with the specified string key.

6. [Renaming][Code-Upgrade] The overloaded version of the **iGrid.Cols.RemoveAt()** method that accepts a string key of the column to remove was renamed to **RemoveByKey** for consistency with similar methods in all other iGrid collections.

7. [Code-Upgrade] The **Reset()** methods of the **iGrid.Footer** and **iGrid.Header** objects were moved to the **Rows** collection of these objects. For example, instead of the statement

```
iGrid1.Footer.Reset();
```

   , use the following one in this release:

```
iGrid1.Footer.Rows.Reset();
```

   There are two reasons for this change. First, the **Reset()** method does not reset the entire **iGrid.Footer** property, but only its rows (the **Rows** sub-property). Second, this is necessary for the new universal property reset infrastructure, which also works for the **iGrid.Footer.Rows** property. For example, calling the **Reset()** method for **iGrid.Footer** is equivalent to the following call

```
iGPropertyManager.ResetProperty(iGrid1.Footer, "Rows");
```

This procedure is also executed as part of the reset process of the entire **iGrid.Footer** property in

```
iGPropertyManager.ResetProperty(iGrid1, "Footer");
```

This all also applies to the **iGrid.Header** property.

8. [New][Code-Upgrade] The **DrawCellContents()** method of iGrid provides a new Boolean parameter **drawContents** that can be used to specify whether this method draws the main cell contents (text, image, combo button, etc.). If you specify False as the value of this parameter, only the cell background will be drawn. This ability is used by the PrintManager add-on to draw parts of group rows with totals on the paper.

   If you use this method in your code and upgrade it for this release of iGrid, specify True as the value of this parameter in all your calls to have the same behavior as in the previous versions.

9. [New] iGrid implements the **ColsAdding** and **RowsAdding** events, which are triggered when new columns or rows are about to be added to the grid, respectively.

10. [Enhancement] The **RowsRemoving** and **RowsRemoved** events occur when all rows are cleared by calling the **iGrid.Rows.Clear()** method or when the **iGrid.Rows.Count** property is set to zero.

11. [Enhancement] If the developer specified an incorrect column or row index when calling an iGrid member, iGrid threw an **ArgumentOutOfRangeException** with the default message "Specified argument was out of the range of valid values". Now iGrid throws an **ArgumentException** with a meaningful message "Invalid column index: <value>" or "Invalid row index: <value>" in these cases.

12. [Enhancement] The previous versions of iGrid did not allow you to create row text cells in the frozen column area. Now it is possible due to enhancements in the corresponding iGrid algorithms.

13. [Enhancement] Now iGrid always sorts the cells in a column with the **CustomGrouping** property set to True if iGrid is grouped by this column regardless of the position of this column in the group/sort criteria.

14. [Renaming][Code-Upgrade] The **colIndices** parameter of the **MergeCellsInCols** and **UnmergeCellsInCols** methods was renamed to **colIndexes** to use the plural form "indexes" for the word "index", which is the de facto standard in software development. The existing code needs changes only if you use named parameters in these methods calls.

Bug Fixes

1. [Fixed] Some problems with drawing of group row contents when iGrid had frozen columns and group rows were not scrollable were fixed.

2. [Fixed] Custom grouping may have worked improperly if iGrid had invisible rows.

3. [Fixed] If column auto-resize mode was turned on by setting the **iGrid.AutoResizeCols** property to True, iGrid did not consider the value of the **iGrid.MarginAfterLastCol** property while adjusting column widths.

4. [Fixed] In some cases iGrid did not draw a group row as selected after clicking its row header to make it the current row.

5. [Fixed] iGrid did not select the next available cell after pressing the RIGHT ARROW key in a group row.

6. [Fixed] iGrid did not select the row text cell after pressing the RIGHT ARROW key in the last normal cell of the row.
7. [Fixed] The **iGrid.CellValues[,]** property threw an **ArgumentOutOfRangeException** if you tried to access a cell in the row text column using the column index -1.
8. [Fixed] If the **iGrid.Header.AllowPress** property was set to False, pressing on a column header changed its appearance to normal instead of remaining hot.
9. [Fixed] Several issues with printing grid lines for row text cells, group rows, column headers and the frozen area edges were fixed in the PrintManager add-on.
10. [Fixed] The Reset command available for iGrid in the WinForms Designer at design time did not reset all sub-properties of iGrid object properties.
11. [Fixed] User could initiate editing of group titles by pressing F2 even if you didn't allow this explicitly in an event handler of the **RequestEdit** event by setting **e.DoDefault** to True.
12. [Fixed] Calling the **GetFilterCriteria()** method of the **iGColAutoFilter** object returned by the **iGAutoFilterManager.ColAutoFilter()** method for columns without attached autofilter threw an exception. Now **GetFilterCriteria()** returns null for columns without attached autofilter.
13. [Fixed] iGrid threw an exception if the width of vertical grid lines was set to 0 when the footer and row header areas were visible.

Addendum: Tags used to classify changes

[New]	-	Completely new feature.
[Change]	-	Change in member functionality or interactive behavior.
[Fixed]	-	Fixed bug or solved problem.
[Removed]	-	Member was completely removed.
[Enhancement]	-	Some functionality was enhanced.
[Optimization]	-	Performance was improved.
[Renaming]	-	Member was renamed.
[Code-Upgrade]	-	Source code for previous versions may require changes.