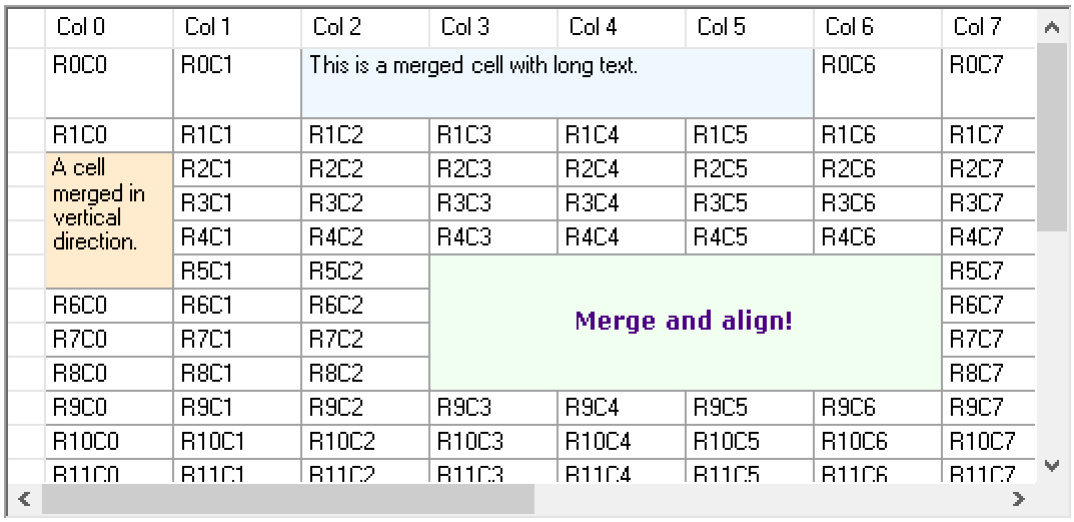
10Tec iGrid for .NET 6.0
What's New in the Release

Tags used to classify changes:

- [New] – a totally new feature;
- [Change] – a change in a member functionality or interactive behavior;
- [Fixed] – a fixed bug or solved problem;
- [Removed] – a member was completely removed;
- [Enhancement] – some functionality was enhanced;
- [Optimization] – a feature has speed improvements;
- [Renaming] – a member was renamed;
- [Code-Upgrade] – the source code for the previous versions requires changes.

New cell merging feature

1. [New] The cell merging functionality for normal cells was implemented. You can use the new **iGCell.SpanRows** and **iGCell.SpanCols** properties to merge normal grid cells:



   By default **iGCell.SpanRows** and **iGCell.SpanCols** properties are set to 1 for every cell of iGrid, which means that a cell is not merged with adjacent cells. To merge a cell with the adjacent cells in the rows below, set the cell's **iGCell.SpanRows** to the corresponding positive value that specifies the number of rows to span across, including the cell's row. For example, to merge a cell with the cells in the next two rows, set the cell's **SpanRows** property to 3.

   The same principle is used when you specify the number of columns to span across. Note that the order of columns on the screen may differ from their definition order because the user can reorder columns. When you merge cells with the **SpanCols** property, the current visible order of columns is used to determine which cells are included into the merging.

2. [New] If you specified a non-default value for the **SpanRows** and/or **SpanCols** property of a cell, this cell is called the root of the merged cell. If you access the **SpanRows**/**SpanCols** properties of all other cells covered with this merged cell, they will return the default value of 1. But you can know the root cell of the merged cell covering a cell if you access its **SpanRoot** property. This new property returns the **iGCell** object representing the root of the merged cell that covers the cell you retrieve this property for. If a cell is the root of a merged cell or it is not included into a merged cell, the **iGCell.SpanRoot** property returns the cell itself.

3. [New] The **iGCellPattern** class used to define a cell pattern implements the two new properties **SpanRows** and **SpanCols**. You can use them to specify the number of rows and columns to span across for the future cells created using a cell pattern. The class also implements a new constructor to specify these values together with other cell properties in one call.

4. [New] You can retrieve a collection of all currently merged cells using the new **iGrid.MergedCells** property. This property returns an enumerable list of merged cells ordered by rows and columns. Having this, you can enumerate all merged cells by rows and by columns within each row using a loop like this:

```
foreach (iGCell item in iGrid1.MergedCells)
{
  Debug.WriteLine($"({item.RowIndex}, {item.ColIndex})");
}
```

   Actually the **iGrid.MergedCells** property provides you with the same features like the collection of selected cells returned by the **iGrid.SelectedCells** property. Both collections have the same base type, which is **iGSelectedCellsCollection**.

5. [New] iGrid implements the new **MergeCellsInCols** method to merge adjacent cells with same values in columns. The picture below demonstrates the result of calling this method for the Customer and Order columns of the grid:

Customer ▲1	Order ▲2	Item	Price
Customer 1	Order 1	Item 3-688	66.70
Customer 1	Order 1	Item 3-823	27.34
Customer 1	Order 1	Item 6-389	52.24
Customer 1	Order 2	Item 4-225	44.18
Customer 1	Order 2	Item 6-718	29.33
Customer 1	Order 2	Item 1-475	60.53
Customer 1	Order 3	Item 3-307	40.39
Customer 1	Order 3	Item 3-440	46.53
Customer 2	Order 1	Item 1-142	95.43
Customer 2	Order 1	Item 1-913	75.69
Customer 2	Order 1	Item 1-620	79.93
Customer 2	Order 1	Item 6-837	99.32
Customer 2	Order 2	Item 6-213	25.11
Customer 2	Order 2	Item 5-709	59.62
Customer 2	Order 2	Item 2-573	85.11
Customer 2	Order 2	Item 3-676	77.77
Customer 2	Order 3	Item 6-846	10.11
Customer 3	Order 1	Item 4-361	46.19
Customer 3	Order 1	Item 1-353	64.83
Customer 3	Order 2	Item 6-589	80.43
Customer 3	Order 2	Item 2-756	23.95

Customer ▲1	Order ▲2	Item	Price
Customer 1	Order 1	Item 3-688	66.70
		Item 3-823	27.34
		Item 6-389	52.24
	Order 2	Item 4-225	44.18
		Item 6-718	29.33
		Item 1-475	60.53
	Order 3	Item 3-307	40.39
		Item 3-440	46.53
Customer 2	Order 1	Item 1-142	95.43
		Item 1-913	75.69
		Item 1-620	79.93
		Item 6-837	99.32
	Order 2	Item 6-213	25.11
		Item 5-709	59.62
		Item 2-573	85.11
		Item 3-676	77.77
	Order 3	Item 6-846	10.11
Customer 3	Order 1	Item 4-361	46.19
		Item 1-353	64.83
	Order 2	Item 6-589	80.43
		Item 2-756	23.95

```
iGrid1.MergeCellsInCols(new string[2] { "Customer", "Order" });
```

   The **MergeCellsInCols** has several overloaded versions that simplify calling this method for one column and specifying column(s) with their numeric indices or string keys.

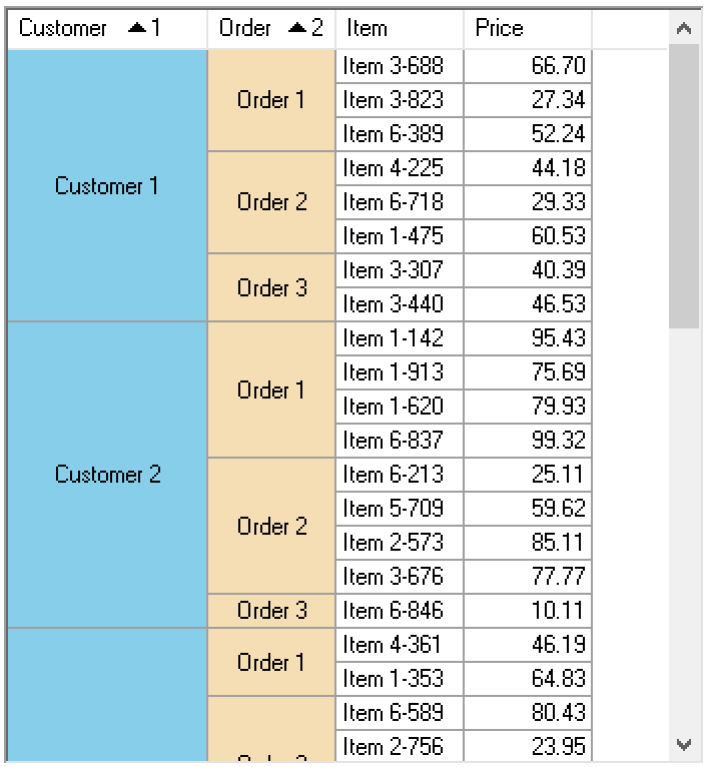
   Other overloaded versions of this method are used to specify the cell style(s) applied to the merged cells in the corresponding columns. For example, you can colorize merged cells and center their texts using the following code snippet:

```
iGCellStyle myMergedCellStyle1 =
iGrid1.Cols["Customer"].CellStyle.Clone();
myMergedCellStyle1.TextAlign = iGContentAlignment.MiddleCenter;
myMergedCellStyle1.BackColor = Color.SkyBlue;

iGCellStyle myMergedCellStyle2 = myMergedCellStyle1.Clone();
myMergedCellStyle2.BackColor = Color.Wheat;

iGrid1.MergeCellsInCols(
  new string[2] { "Customer", "Order" },
  new iGCellStyle[2] { myMergedCellStyle1, myMergedCellStyle2 });
```

   The result of execution of this code is on the picture below:



Customer ▲1	Order ▲2	Item	Price
Customer 1	Order 1	Item 3-688	66.70
		Item 3-823	27.34
		Item 6-389	52.24
	Order 2	Item 4-225	44.18
		Item 6-718	29.33
		Item 1-475	60.53
	Order 3	Item 3-307	40.39
		Item 3-440	46.53
Customer 2	Order 1	Item 1-142	95.43
		Item 1-913	75.69
		Item 1-620	79.93
		Item 6-837	99.32
	Order 2	Item 6-213	25.11
		Item 5-709	59.62
		Item 2-573	85.11
		Item 3-676	77.77
	Order 3	Item 6-846	10.11
	Order 1	Item 4-361	46.19
		Item 1-353	64.83
		Item 6-589	80.43
		Item 2-756	23.95

6. [New] iGrid implements the new **UnmergeCellsInCols** method to remove merging from the cells in the specified columns. This method is a handy tool to remove merging created with the **MergeCellsInCols** method.

   The **UnmergeCellsInCols** has several overloaded versions that simplify calling this method for one column and specifying column(s) with their numeric indices or string keys. Other overloaded versions of this method are used to optionally remove links to cell styles from the cells in the specified columns.

7. [New] iGrid provides you with the new **UnmergeAllCells** method to remove merging from all merged cells. This method works faster than enumerating all cells and resetting their **SpanRows**/**SpanCols** properties due to special internal optimization.

   One of the overloaded versions of this method allows you to remove links to cell styles in the processed merged cells to clear any formatting in them.

8. [New] If iGrid contains merged cells, some operations you could do from code or interactively become unavailable. These restrictions are caused by the following two basic principles of merged cells:

   1) A merged cell cannot be separated into pieces.
   2) Merged cells cannot intersect.

   The main cases that can lead to separation of merged cells are:

   - Column or row insertion is not allowed if it can break a merged cell.
   - Reordering of rows or columns (interactively or from code) is not allowed if the moved row/column belongs to a merged cell.
   - If a column or row intersects a merged cell, the removal of this column or row is not allowed.
   - The horizontal or vertical frozen area edges cannot cross merged cells, though merged cells can abut these lines by one of their sides.
   - Merged cells cannot span across group rows or row text cells.
   - If a grid contains merged cells spanning rows, sorting or grouping is not allowed by default as reordering rows leads to breaking merged cells in the general case.

Regarding intersection of merged cells, note that row text cells and group rows are also considered merged cells in this context and their intersections in various combinations are not allowed.

If the user is trying to do something that can lead to intersection of merged cells or their separation, nothing happens without a message for the user. For example, iGrid simply does not allow dragging of the column headers of the columns belonging to cells merged in the horizontal direction. If the developer is trying to do something in code that would separate a merged cell into pieces or lead to intersection of merged cells, iGrid raises an exception with the corresponding message.

Note that interactive sorting and grouping are disabled by default if iGrid contains cells merged in the vertical direction. However, if the logic of your app requires this functionality, you can enable these operations by unmerging the merged cells on the fly in the corresponding situations (for example, when the user clicks a column header to sort the grid by the corresponding column). This is done with the help of the new **UnmergeCellsRequired** event introduced in this release of iGrid (read below).

9. [New] Sorting or grouping leads to breaking cells merged in the vertical direction because of changing row order, and these operations are disabled by default in iGrid if it contains cells spanning rows. If you try to call the **Sort** or **Group** method from code when iGrid has such cells, you get an exception with the corresponding message. In the case of interactive operations iGrid acts intelligently in this situation.

   To make sorting or grouping possible when the user clicks a column header or drags it to/from the group box area, iGrid raises the new **UnmergeCellsRequired** event that allows you to unmerge cells in its event handler and tell iGrid about that to proceed with sorting/grouping. The event's data are represented with an object of the new **iGUnmergeCellsRequiredEventArgs** class:

```
public class iGUnmergeCellsRequiredEventArgs : EventArgs
{
  public bool CellsUnmerged;
  public readonly iGUnmergeCellsRequiredReason Reason;
  public readonly int ColIndex;
}
```

   The read-write **CellsUnmerged** property is used to tell iGrid that the developer has unmerged cells in the event handler and iGrid may proceed. The default value of this property is False, so you must set it to True explicitly after unmerging cells to complete the sorting or grouping. If the property remains unchanged, iGrid simply aborts the current sorting or grouping operation.

   The **Reason** property of the new **iGUnmergeCellsRequiredReason** enumeration type indicates the current operation: sorting (**iGUnmergeCellsRequiredReason.Sorting**) or grouping (**iGUnmergeCellsRequiredReason.Grouping**).

   The last property, **ColIndex**, contains the index of the column this event is related to.

   Below is an example of the **UnmergeCellsRequired** event handler that demonstrates how to remove merging from all merged cells to enable sorting when the user clicks column headers:

```
private void iGrid1_UnmergeCellsRequired(
  object sender, iGUnmergeCellsRequiredEventArgs e)
{
  if (e.Reason == iGUnmergeCellsRequiredReason.Sorting)
  {
    iGrid1.UnmergeAllCells();
    e.CellsUnmerged = true;
  }
}
```

   The **UnmergeCellsRequired** event allows you to write any code that removes merging from cells according to the logic of your app. For instance, you may also clear formatting in the processed merged cells and/or remove merging only from the cells merged vertically but leave the cells merged

horizontally untouched as they moved within sorted rows. Pay attention to the fact that you need to remove cell merging properly so that iGrid will be able to finish the sorting/grouping operation – otherwise an exception will be thrown.

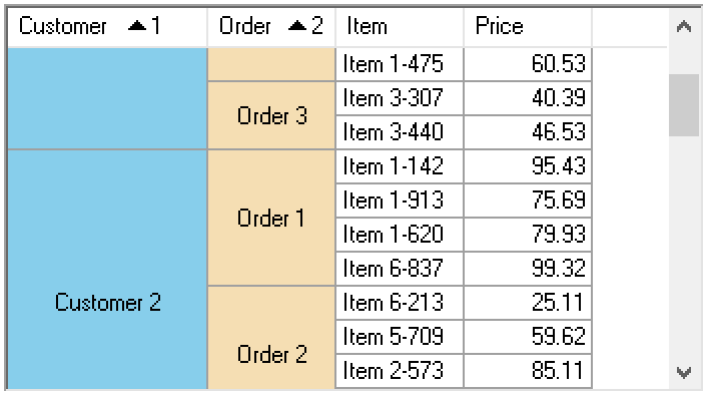
10. [New] The design-time functionality of iGrid was enhanced to provide the ability to define merged cells in the Windows Forms designer at design time.

11. [New] The PrintManager add-on also supports printing of merged cells. It takes into account information about merged cells in the source grid and draws them on the paper accordingly. For example, if a merged cell is broken between pages, it is drawn without corresponding grid lines at the top/bottom/left/right to indicate this.

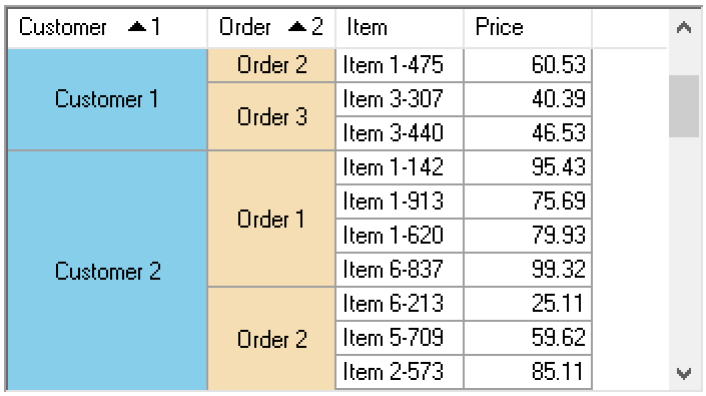
Changes in the core grid control

1. [New] This release introduces a new feature that allows you to force iGrid to draw cell contents in the cell area viewport. This feature is especially useful for merged cells with heights or widths exceeding the size of the visible cell area.

   Look at the following screenshot:



   It's not clear for what customer and order we see the item and price in the first visible row below the column header area. If we enable the new feature to force iGrid to draw cell contents in the visible cell area, we will improve the look significantly:



   This functionality can be enabled individually for each cell using the new **FitContentsInViewport** property of the iGrid cell object (**iGCell**) or the iGrid cell style object (**iGCellStyle**). This property accepts values from the new **iGCellFitContentsInViewport** enumeration:

```
[Flags]
public enum iGCellFitContentsInViewport
{
  None = 0,
  NotSet = 1,
  Vertically = 2,
  Horizontally = 4
}
```

As you can see, you can force iGrid to draw cell contents in the viewport by moving the cell contents dynamically in the vertical or horizontal direction. The enumeration is marked with the **Flags** attribute, so you can combine these two values:

```
myMergedCellStyle1.FitContentsInViewport =
  iGCellFitContentsInViewport.Horizontally |
  iGCellFitContentsInViewport.Vertically;
```

2. [New] The **iGSortObject** class used as the base type for the **SortObject** and **GroupObject** properties of iGrid implements the new read-only property named **ColArray**. This property returns an array of integer column indices representing the columns iGrid is currently sorted by. The order of columns in the array correspond order of columns in the sort criteria in the case if iGrid is sorted by multiple columns.

   The **ColArray** property is especially useful in a scenario when you want to allow your users to merge cells with similar values in the sorted columns. The following event handler of the **AfterContentsSorted** event in a pair with the **UnmergeCellsRequired** event solve this task:

```
private void iGrid1_AfterContentsSorted(object sender, EventArgs e)
{
  iGrid1.MergeCellsInCols(iGrid1.SortObject.ColArray);
}

private void iGrid1_UnmergeCellsRequired(
  object sender, iGUnmergeCellsRequiredEventArgs e)
{
  iGrid1.UnmergeAllCells();
  e.CellsUnmerged = true;
}
```

   When the user clicks a column header, iGrid is sorted according to the user click and the cells in the sorted column are merged based on their values after that. Multiple column sorting also works in this case.

   Pay attention to the fact that iGrid does not automatically removes the existing cell merging before applying the new sort criteria and we need to do this manually in the event handler of the **UnmergeCellsRequired** event introduced in this release of iGrid.

3. [New][Change][Code-Upgrade] The new Boolean **AdjustScrollBarValuesRedrawOff** property can be used to specify whether the values of its scroll bars are adjusted after every operation that may affect the positions of the scroll boxes even if iGrid updates are turned off with the **BeginUpdate** method. Among these operations are rows removal, new columns creation and the like.

   The default value of this property is True. This means that if you, for example, remove all grid rows between the **BeginUpdate** and **EndUpdate** calls, the position of the scroll box on the vertical scroll bar will be set to the top after the **EndUpdate** call. In the previous builds of iGrid, the scroll box position was not changed in this case if the new scroll range allowed to use the same scroll box position. If you used this feature of iGrid and would like to maintain backward compatibility after upgrade to this release of iGrid, set the **AdjustScrollBarValuesRedrawOff** property of iGrid to False in your apps.

4. [New][Code-Upgrade] iGrid implements the new **TextRenderingHint** property of the **System.Drawing.Text.TextRenderingHint** type that allows you to get or set the rendering mode for all texts displayed in iGrid. This property is useful if you want to set the desired antialiasing effect for texts in iGrid. The default value of this property is **TextRenderingHint.SystemDefault**, which corresponds the antialiasing effect used in all previous builds of iGrid.

   The **iGCellEditorBase** class and the **IiGDropDownControl** interface used to implement custom cell editors were also supplemented with the following **SetTextRenderingHint** method:

   ```
   SetTextRenderingHint(
     System.Drawing.Text.TextRenderingHint textRenderingHint)
   ```

   It is called by iGrid before displaying a custom editor and allows you to inherit the **TextRenderingHint** setting of iGrid in the editor, which is passed in the **textRenderingHint** parameter of the method.

   To compile your code successfully after upgrade from earlier versions of iGrid, add an implementation of the **IiGDropDownControl.SetTextRenderingHint** method to all classes based on the **IiGDropDownControl** interface – though the method body can be empty. There is no need to do this for cell editors based on the **iGCellEditorBase** class as the **SetTextRenderingHint** method is defined as virtual (Overridable in VB.NET) and has basic empty implementation in the **iGCellEditorBase** class.

   The current **TextRenderingHint** setting of iGrid is also applied automatically to the filter value list and custom condition list in the filter boxes in the attached AutoFilterManager add-on. All other controls in the filter box interface (the top toolbar items, the buttons at the bottom) inherit the ClearType OS setting like all native Windows controls (command buttons, check box controls, menu items, etc.) to provide the consistent visual effect.

5. [New][Code-Upgrade] The **DrawCellContents** methods has a new parameter named **customTreeLinesPen**. This is a parameter of the .NET **Pen** type and it can be used to specify a custom **Pen** object to draw tree lines. If it is not specified (null in C# or Nothing in VB.NET), the default tree lines defined with the iGrid **TreeLines** object property are used.

6. [New] Now you can retrieve the displayed text in a footer cell using the new string **iGFooterCell.Text** read-only property.

7. [New] The **iGFooterRow** class implements the **Cells** property returning the collection of the cells in a footer row. Every item of this collection representing one footer cell can be retrieved by a column numeric index or string key.

   This collection of footer cells is also enumerable and can be used in for-each loops. This allows you to easily enumerate all footer cells and retrieve their properties like in the following example printing the texts of all footer cells:

   ```
   foreach (iGFooterRow footerRow in iGrid1.Footer.Rows)
   {
    foreach (iGFooterCell footerCell in footerRow.Cells)
    {
     System.Diagnostics.Debug.WriteLine(
      $"Cell ({footerRow.Index}, {footerCell.ColIndex}): {footerCell.Text}");
    }
   }
   ```

12. [New] The **iGColHdrPattern** class used to define a column header pattern implements the two new properties, **SpanRows** and **SpanCols**, to specify the number of rows and columns to span across. The class also implements a new constructor to specify these values together with other column header properties in one call.

8. [New] The **iGColHdrPattern** class implements the standard .NET **ICloneable** interface and the **Clone** method returning a copy of the class in its native type.

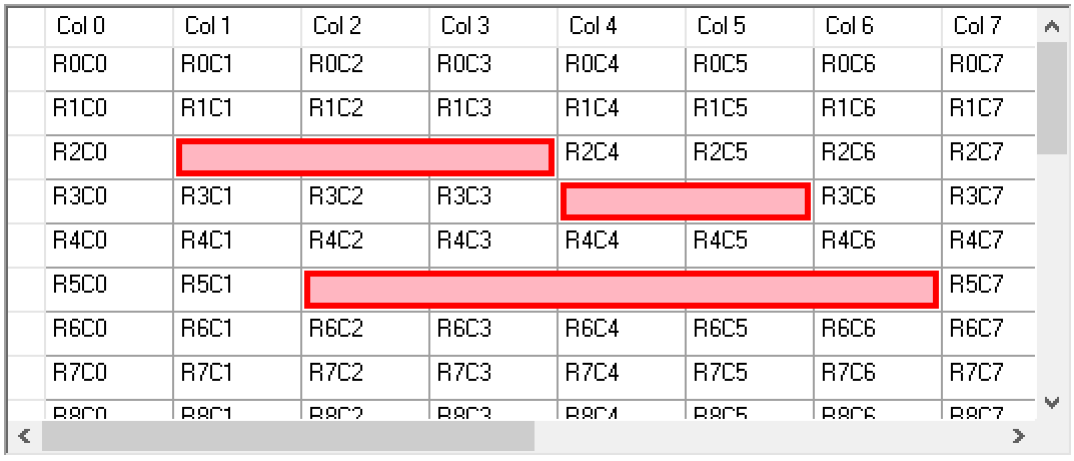
9. [New] The **iGRow** class implements the new **Parent** property of the same **iGRow** type returning the parent tree node or group row for a row.

10. [New] The **iGCol** class implements the new **ContainsRowText** function that returns a Boolean value indicating whether a column is one of the columns row texts are displayed in. The function takes into account the current position of the column in the grid, the visibility of row text cells and their start and end columns if they were specified with the **RowTextStartColNear** and **RowTextEndColFar** properties of iGrid.

11. [New][Change][Renaming][Code-Upgrade] The **iGRowPattern** class implements the new Boolean **VisibleParentExpanded** property that specifies whether a new row created with this pattern will be visible if its parent row is collapsed or expanded. This property corresponds the **iGRow.VisibleUnderGrouping** property from the previous builds of iGrid, which was renamed to **iGRow.VisibleParentExpanded** in this release.

    There are several reasons why this property was renamed. First, this status is applied to both tree grids and grids with group rows, and 'VisibleParentExpanded' is a generic name for these types of grids. Second, the previous builds of iGrid could set the **iGRow.VisibleUnderGrouping** property automatically for rows created in some situations (for instance, if the developer created a row nested to a tree node). This behavior could be useful in some scenarios but it hindered writing effective code in other scenarios when the developer could specify the expected **VisibleUnderGrouping** status for new rows. As a result, the property was renamed to avoid unexpected changes in the behavior of existing code after upgrade to this release of iGrid.

12. [New][Renaming] This release of iGrid provides you with the new **PostPaint** event raised after iGrid has finished all drawing operations in the current drawing cycle. With the help of this event you can draw your own graphics over the standard grid contents. This allows you to mark cells with colored rectangles around them, draw arrows between cells and overlay normal cells with bigger rectangles to emulate Gantt charts and so on. Below is a simple demonstration of red rectangles placed over normal cells:



The **PostPaint** event work similar to the **CustomDrawBackground** event, though in contrast to **CustomDrawBackground** the new event does not require to set a property to enable triggering of this event. Both **PostPaint** and **CustomDrawBackground** events provide you with the same event arguments encapsulated into an object of the **iGCustomDrawControlEventArgs** class. Actually the **iGCustomDrawControlEventArgs** class is the renamed **iGCustomDrawBackgroundEventArgs** class used in the previous versions of iGrid with the **CustomDrawBackground** event. The **iGCustomDrawBackgroundEventHandler** delegate was also renamed to **iGCustomDrawControlEventHandler** accordingly.

13. [Optimization][Removed][Fixed][Code-Upgrade] The internal core functionality of row and column keys was completely rewritten to provide better performance and to avoid possible problems with finding rows or columns by specified keys improperly.

    As a part of the updated functionality, now all keys you use in iGrid must be unique. The previous versions of iGrid implement the Boolean **UniqueKeys** property to turn key uniqueness check off for better performance. This property was removed in this release because the new key functionality provides the same performance with automatic check of key uniqueness.

    Some operations with rows when row keys are specified, namely sorting and row movement, work 20% faster now. Row insertion and row removal may work longer because of the fixed problems with key management in the previous versions, but the performance degradation is measured in tens of milliseconds when iGrid has 100'000+ rows and theoretically may be noticeable for the user only on slow computers.

14. [Change][Code-Upgrade] In the previous builds of iGrid the **SpanRows** and **SpanCols** properties of a column header object (**iGColHdr**) or a footer cell object (**iGFooterCell**) returned 0 for cells covered with a merged cell. Now they return 1 because actually they were not changed. You can still retrieve the root of the merged cell covering them with the **SpanRoot** property of the corresponding object.

15. [Enhancement][Change] The **CommitEditCurCell** and **CommitEditColHdr** methods now return a Boolean value indicating whether the current editing has been finished after their call. If the editing has been committed or cancelled, the return value is True; if it proceeds, the return value is False. If iGrid was not in edit mode when one of these methods was called, True is returned.

16. [Enhancement][Change][Code-Upgrade] In the previous builds of iGrid the default values for the width and color of the frozen area edges were 0 and **SystemColors.ControlDark** respectively (these are the **ColsEdge.Width**/**RowsEdge.Width** and **ColsEdge.Color**/**RowsEdge.Color** sub-properties in the **iGrid.FrozenArea** object property). To display a frozen area edge and make it noticeable for the user, you needed to specify a non-zero width and specify a color that is different from **SystemColors.ControlDark** as this color is used as the default color for grid lines too:

    ```
    iGrid1.FrozenArea.ColsEdge.Color = Color.Red;
    iGrid1.FrozenArea.ColsEdge.Width = 2;
    iGrid1.FrozenArea.ColCount = 3;
    ```

    Starting from this release of iGrid, the width and color of the horizontal and vertical frozen area edges are set to 2 and **SystemColors.ControlDarkDark** by default. This allows you to display a thick noticeable line that separates the frozen area from non-frozen one simply by specifying the number of columns or rows to freeze, for example:

    ```
    iGrid1.FrozenArea.ColCount = 3;
    ```

17. [Enhancement] In the previous builds of iGrid the current editing was cancelled when the user started to rotate the mouse wheel to scroll iGrid. Now iGrid applies its standard editing infrastructure, including the ability to check the user's input and proceed editing in the **BeforeCommitEdit** event, before scrolling iGrid when the user rotates the mouse wheel. If the developer tells iGrid that the editing must proceed, the grid isn't scrolled.

18. [Enhancement] If iGrid does not have the current cell, pressing a key to move the current cell (Right Arrow, Down Arrow, etc.) automatically selects the very first cell available for selection.

19. [Enhancement] If iGrid works in row mode and the **ShowControlsInAllCells** property is set to False, cell controls (combo button, ellipsis button) are displayed in all cells of the current row. In the previous builds the cell controls were displayed only in the current cell inside the current row, but in most cases this cell was not highlighted unless you set the **RowModeHasCurCell** and **CurCellBackColor** properties of iGrid for that.

20. [Enhancement] The system of iGrid exceptions was revised and enhanced. iGrid throws many new exceptions explaining in detail why an operation cannot be done (for instance, when columns or rows cannot be moved, or if adding or removing a column can break a merged cell, etc.).
21. [Optimization] The internal algorithms related to storing/retrieving cell data were optimized.
22. [Fixed] Some problems with messages in the existing exceptions were fixed.
23. [New][Fixed] iGrid did not draw the horizontal segment of tree lines for root nodes in some situations. The fix for this problem required the new **iGTreeBranchState.Right** enumeration member that was added to the **iGTreeBranchState** enumeration in this release of iGrid.
24. [Fixed] Several critical bugs related to column management (adding, removing and reordering columns) that may lead to data loss and crashes were fixed.
25. [Fixed] If you opened the iGrid column collection editor at design time, iGrid may have lost all your column settings made at design time after you closed the column collection editor form.
26. [Fixed] iGrid did not draw special vertical grid lines in the footer section properly if it had cells merged horizontally. This concerns the grid line separating the frozen area and the grid line for the last visible column when their width was greater than 1 pixel.
27. [Fixed] The auto-resize columns algorithm enabled with the **iGrid.AutoResizeCols** property did not work properly in some specific cases.
28. [Fixed] iGrid may have failed if you added or removed grid rows from the mouse events related to the footer section.
29. [Fixed] iGrid did not skip row with zero height and column with zero width during the keyboard navigation using such keys as Right Arrow, Down Arrow and the like.
30. [Fixed] iGrid did not select cells in invisible columns and rows when the **SelectInvisibleCells** property was set to True.
31. [Fixed] iGrid did not highlight combo buttons in column headers and did not process clicks on them correctly. The same problems were also fixed for the column header buttons used to open column filter boxes when AutoFilterManager is attached to the grid.
32. [Fixed] The height of drop-down list items in **iGDropDownList** objects was not adjusted automatically after changing the drop-down list's font.
33. [Fixed] Column headers were not drawn correctly in the right-to-left mode: the column header dividers were shifted and did not correspond grid lines. This problem was especially noticeable in Windows 8 and Windows 10 because of the visual styles used in these versions of the OS.
34. [Fixed] The iGrid scroll bars may have been drawn with visual styles when usage of visual styles was turned off.
35. [Fixed] iGrid may have failed if a message box or another dialog was displayed from an event handler of the **ColHdrClick** or **ColHdrEndDrag** event.
36. [Fixed] iGrid may have crashed after doing some operations with it in an event handler of the **ColHdrComboBeforeCommit** event.
37. [Fixed] iGrid may have crashed when the user clicked a column header's combo button if a cell's drop-down list was visible.
38. [Fixed] If a cell was set as a password cell using the **iGCellTypeFlags.Password** flag, iGrid displayed the contents of the cell "as is" in the built-in tooltip if the cell text was clipped on the screen. In this release of iGrid the built-in tooltip is not displayed for such a cell – though you can always specify it dynamically if required with the **RequestCellToolTipText** event.

Changes in the PrintManager add-on

1. [New][Removed][Fixed][Code-upgrade] The new **FitColsOnPage** property was implemented. If the grid printout is more than one page wide, this property specifies how the component shrinks the grid control on paper so that it becomes one page wide. The **FitColsOnPage** property accepts one of the following 3 values from the new **iGFitColsOnPage** enumeration:

iGFitColsOnPage.None	Print the grid "as is" without any modifications.
iGFitColsOnPage.ResizeCols	Shrink every column so that the result grid width becomes equal the page width.
iGFitColsOnPage.ScaleGrid	Scale down the whole grid printout on paper to fit the page width.

   The **FitColWidths** property from the previous versions of the add-on were removed as now its work is performed with the **iGFitColsOnPage.ResizeCols** setting in the new **FitColsOnPage** property.

   The column resizing algorithm applied to the grid when the **FitColsOnPage** property was set in the previous version contained some bugs causing improper printing of the grid. These problems were fixed in this release of the add-on.

2. [New][Enhancement] The new **PrintGridHeader** property was implemented. It allows you to specify how PrintManager prints the grid header area containing the column headers and the header for the row header band. This property accepts on of the values of the new **iGPrintGridHeader** enumeration:

iGPrintGridHeader.Never	The grid header is never printed.
iGPrintGridHeader.BeforeFirstRow	The grid header is printed on every page containing the first grid row.
iGPrintGridHeader.OnEveryPage	The grid header is printed on every page.

   The default value for the **PrintGridHeader** property is **iGPrintGridHeader.OnEveryPage**, which corresponds the behavior of PrintManager in the previous versions.

   Note that even if **PrintGridHeader** is set to **iGPrintGridHeader.BeforeFirstRow** or **iGPrintGridHeader.OnEveryPage,** the grid header is printed only if it is visible in the grid (the **iGrid.Header.Visible** property equals True).

   The new **PrintGridHeader** property can be used to enhance the behavior of existing apps. In the previous versions of PrintManager, if you needed to print the grid without header even if its header was visible on the screen, you could temporarily hide the grid header by setting the **iGrid.Header.Visible** property to False before printing and then restore the original True value after printing. When you did this, the iGrid header disappeared on the screen while printing or print-previewing the grid. But now you can avoid this effect and print the grid without header on the paper simply by setting the PrintManager's **PrintGridHeader** property to **iGPrintGridHeader.Never**.

3. [Fixed] Several bugs related to using the PrintManager add-on in high-DPI environments were fixed.

4. [Fixed] Segments of tree lines were not drawn properly.

5. [Fixed] The system Print… dialog invoked from the corresponding tool bar button of the print-preview window did not have the input focus and the user needed to click it to give it the input focus.

6. [Fixed] This version of PrintManager contains fixes for various minor problems with drawing grid lines on the paper if non-default grid line settings were used.