10Tec iGrid.NET X
What's New in Release

Contents

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 Drawing Concept for Header Areas

This release of iGrid brings an updated drawing system of its column headers and row headers. The new drawing concept solves the problems of previous versions described below.

When row and column headers were rendered with the OS visual styles in the previous versions, iGrid got the column header background bitmaps from the OS. However, row headers are not part of the OS visual styles specification, and iGrid produced row header bitmaps on-the-fly by rotating the column header background image 90 degrees counterclockwise. These header images also came with the grid lines "built-in" directly in the bitmap. All this caused the following problems:

a. The developer had no way to change column header and row header grid lines when visual styles were used. This concerns their color, visibility and thickness. The only way to change these grid lines was turning the system drawing off by setting the **iGrid.Header.DrawSystem** and **iGrid.RowHeader.DrawSystem** properties to `False`, but this led to losing visual styles at all.

b. In Windows 7, the vertical grid line for the row header area was shifted with respect to the column header for the row header area:

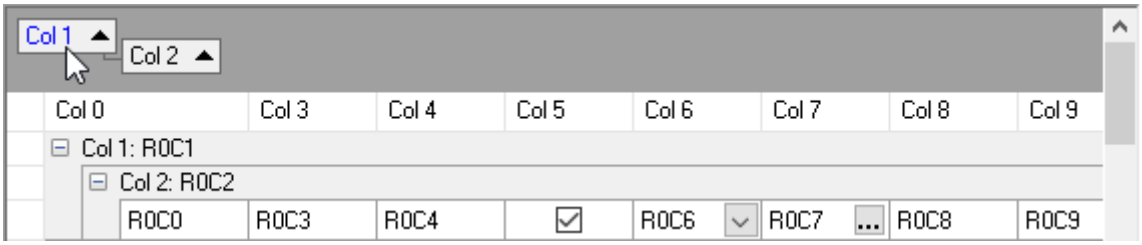
c. In Windows 10 the column header bitmap does not contain the horizontal grid line at the bottom, and actually the previous versions of iGrid added this horizontal line over the styled column header bitmap to visually separate column headers from cells. This drawing is hard-coded, which may lead to unpredictable visual effects if Microsoft decides to change their OS visual styles in one of the future updates of Windows 10.

d. The color of the column header grid lines could not be changed in Windows 10 because they are part of the OS-styled header bitmap returned by the OS. Thus, neither developers could set the required grid lines color nor we could choose a good default grid lines color to better separate the header area from the cells area (the grid lines that come from the OS are very light).

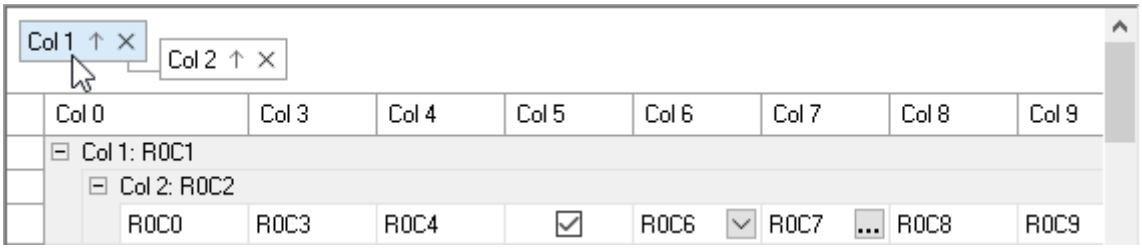
To solve all these problems, we decided to abandon the idea of getting column header bitmaps from the OS and now draw them from scratch using GDI+. We chose the style of the modern Windows Explorer, which is the same in Windows 7, 8 and 10. Now you will get this look in column and row headers in all editions of Windows if visual styles are used to render iGrid header areas (the default setting):

	Col 0	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Co
	R0C0	R0C1	R0C2	R0C3	R0C4	☐	R0C6	R0C7	R0
	R10C0	R10C1	R10C2	R10C3	R10C4	☑	R10C6	R10C7	R1
	R11C0	R11C1	R11C2	R11C3	R11C4	■	R11C6	R11C7	R1

In the previous versions of iGrid the column headers in the group box were rendered with a special hard-coded simplified style regardless of the style used in the column header row above cells. As a result, column header backgrounds and internal items of column headers (such as sort icons or combo buttons) were different in the group box and in the column header row above cells:



The new drawing system solves this problem too by making column headers the same both in the group box and the column header row:



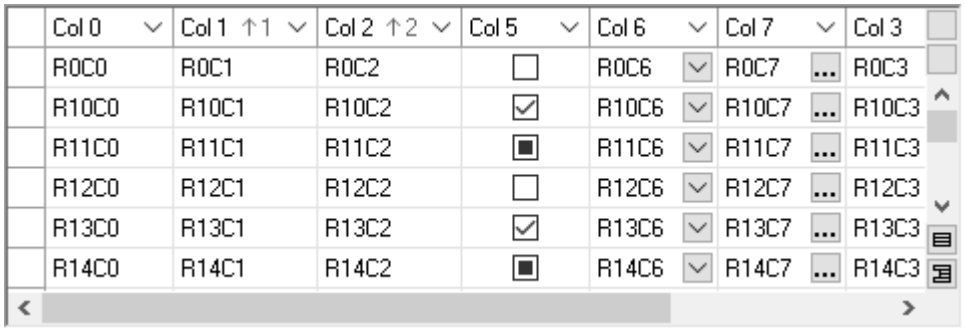
Notice the new default colors of grid lines in column headers, row headers, and cells on the pictures above. In iGrid X the items of the header areas are separated with darker grid lines, and cells are separated with lighter grid lines. Users find this choice of colors more natural as it helps them to distinguish header and cell areas easier.

The developers got the ability to change the styles of the vertical and horizontal grid lines in system-styled column headers and row headers with the existing **HGridLinesStyle** and **VGridLinesStyle** properties of the **iGrid.Header** and **iGrid.RowHeader** objects without losing of system look. These properties are initialized with the **iGPenStyle** objects corresponding to the new default grid line colors. The width of all these pen style objects is set to 1 pixel in contrast to the previous versions of iGrid in which it was set to 0, which also reflects the actual look.

The new header drawing system allows the PrintManager component to print column and row headers exactly as the user sees them on the screen if the **PrintingOptions** property of PrintManager contains the `DrawSystemStyledHeader` and `DrawSystemStyledRowHeader` flags. In the previous versions of iGrid, system-styled column headers and row headers were printed using the 3D look even if these flags were specified because of technical limitations related to screen-oriented drawing of visual styles.

Enhanced UI Elements

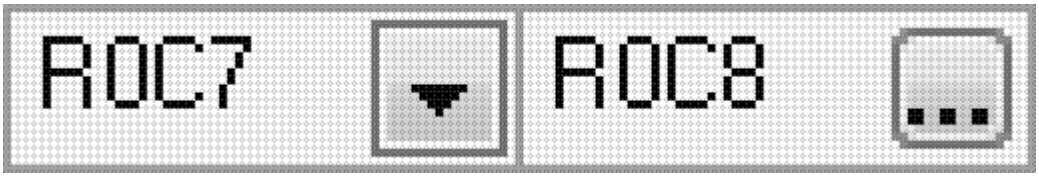
The drawing of many UI elements in this release was improved. All these new elements provide a cleaner and more consistent overall look compared to the previous versions of iGrid, especially if they are used together in the same grid:



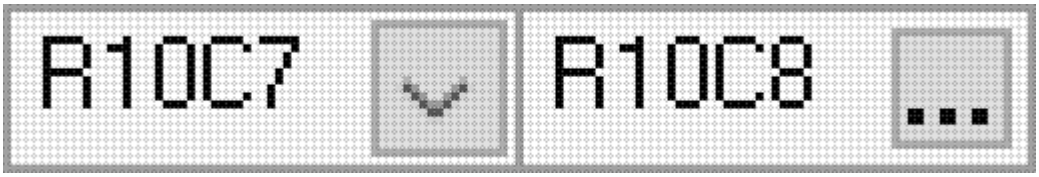
These enhancements concern mainly the default appearance when iGrid is rendered with the OS visual styles, but they are also applicable to the other available styles - 3D and Flat. Below you will find the detailed description of every enhancement in this area.

1. In the previous versions of iGrid, if it was rendered with the OS visual styles, cell combo buttons used the image of the combo button from the OS combo box control whilst cell ellipsis buttons and custom scroll bar buttons used the standard push button background. This worked enough well in Windows XP, but leads to noticeably different look of these buttons because of changes in the OS visual styles in Windows 7 and Windows 10:

   Windows 7:

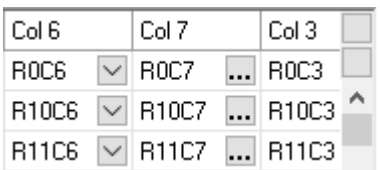
   

   Windows 10:

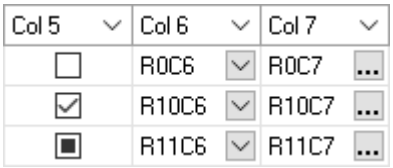
   Also pay attention to the sizes of combo and ellipsis buttons in Windows 10 in the previous versions of iGrid. They differ though they must have the same size as they are same cell button controls. You can see that the size of the combo button is 17x17 pixels whilst the ellipsis button has the size of 15x15 pixels. The drawing code of iGrid requests the drawing of the corresponding visual elements from the system in the rectangle of the same size in both cases (17x17 pixels by default), but the OS draws the push button image in the case of ellipsis button with an unneeded 1-pixel transparent border.

   To fix all these problems with cell buttons in visual styles, now iGrid uses the same push button background image for all cell buttons to provide a consistent look:



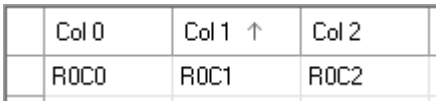
2. There was one more problem with the system image used for combo buttons in visual styles: the arrow glyph is drawn by the OS in the fixed vertical position. If a grid row height was small enough, this glyph was placed at the bottom of the combo button and the button looked not attractive in such cases:

   To fix this problem, now the arrow glyph is drawn by iGrid. The position of this glyph is centered vertically in the combo button, which creates an even better look compared to the previous implementation. The same drop-down glyph is drawn in column headers as well:



   Notice that the push button background is not drawn for combo buttons in column headers to provide a cleaner look. It appears only when the mouse pointer is placed over a column header combo button as the hot-track effect.

3. Column headers indicate sorting with the new sort arrows regardless of the look of iGrid (the OS visual style, 3D or flat). Their default color is set to `SystemColors.ControlDarkDark`, which is lighter than the default `SystemColors.ControlText` color of column header texts. This color combination leads to better distinguishing between column header texts and sort icons:



4. The plus/minus button used in group rows and trees to collapse/expand rows in the previous versions of iGrid was rendered by the OS and had this look:

   The look of this button has not been updated since the times of Windows 7 and remains the same in Windows 10. The main visual problem with it is that the button has a gradient background and overall convex look not corresponding to the general flat look of the Windows 10 controls. One more inconsistent element is the plus/minus glyph inside drawn with the hard-coded dark blue color. It does not correspond to the color scheme of controls in Windows 10 built on the black and white colors.

   iGrid X fixes these problems by rendering the plus/minus button itself regardless of the appearance settings, including the visual styles. The plus/minus button from the flat style is used everywhere now:

   This approach provides two more benefits. First, the colors of this button can be changed with the **iGrid.CellCtrlBackColor** and **iGrid.CellCtrlForeColor** properties – like the colors of other cell controls (combo button, ellipsis buttons, etc.). Second, the size of this button can be adjusted for easier use on touch screens – which is not available for the OS plus/minus button drawn with a fixed size.

5. The updated row header edit icon is monochrome now for consistent look with other UI elements.

Concept of Overridable Properties

iGrid calculates a lot of parameters related to the user interface automatically. For example, these are the size of the level indent used to show hierarchical row relations or the thickness of the scroll bars. However, you may need to override these parameters in specific cases. To do it, iGrid provides you with a set of properties based on a new concept of overridable values. The main rules of this concept are the followings:

- You can read the actual value of a UI parameter with the corresponding read-only property. Let's call it simply **Parameter**.
- To use a custom value for this UI parameter instead of the value calculated by iGrid automatically, you assign this value to the property with the 'Override' suffix – **ParameterOverride**.
- To retrieve the value calculated by iGrid automatically regardless of the fact whether you overwritten it with the **ParameterOverride** property, read the value of the corresponding read-only **ParameterAutoValue** property.

Let us explain this for the level indent area. In the previous versions of iGrid you could read/set the width of the level indent with the **iGrid.LevelIndent** property. This version of iGrid calculates this parameter automatically to provide the best look on the screen taking into account several other parameters, such as the size of the plus/minus button and the current dpi of the screen. The **iGrid.LevelIndent** property became a read-only property as now it returns the effective width of the level indent calculated automatically by default. If you want to redefine this default behavior and use a specific constant level indent size, you can do it by assigning this size value to the new **iGrid.LevelIndentOverride** property. The third level indent related property, **iGrid.LevelIndentAutoValue,** allows you to retrieve the level indent size iGrid would use automatically even if you overwrote it with **iGrid.LevelIndentOverride**.

The key point in this property system is that the **ParameterOverride** property has the **Nullable** data type for the underlying value type of the parameter. The default value of any **ParameterOverride** property is `null` (`Nothing` in VB), which means that the value determined automatically is used. When you assign a concrete value to **ParameterOverride** property**,** you ask iGrid to use namely this value – be it a positive value or even zero. To return to automatic calculation of the value, assign `null` to **ParameterOverride** again.

For example, to use the level indent of 15 pixels, use the following statement:

```
iGrid1.LevelIndentOverride = 15;
```

To return to automatic calculation of the size of the level indent, do the following:

```
iGrid1.LevelIndentOverride = null;
```

Pay attention to the following screenshots demonstrating how the value of any **ParameterOverride** property looks in the VS Property Browser when it contains the default value of `null` or a given value:

LevelIndentOverride = null

HotTracking	True
ImageList	(none)
LevelIndentOverride	
MarginAfterLastCol	0
MarginAfterLastRow	0

LevelIndentOverride = 0

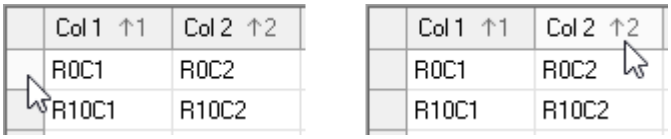
HotTracking	True
ImageList	(none)
LevelIndentOverride	**0**
MarginAfterLastCol	0
MarginAfterLastRow	0

To set the value of such a property to `null` in the Windows Forms designer, you simply erase the property value in the Property Browser.

You will find several examples of *Override and *AutoValue properties in this document below in the corresponding sections.

Hot-track Effects Enhancements

1. [Enhancement][Change] In the previous versions of iGrid, if you set the flat appearance for row headers and column headers, they used different hot-track effects by default: the hot row header under the cursor changed its background to a lighter color, but the hot column header changed only the color of its text. To get the same visual effects in all header areas when the flat style is used, now the active column header also changes its background color to a lighter one – exactly like row headers:



2. [Change][Removed][Renaming][Code-upgrade] The **Header.HotTrackFlags** property was used in the previous versions to control which parts of the hot column header are highlighted when the header is rendered with the 3D or flat style. This version of iGrid provides two different hot-track flag sets for the 3D and flat styles instead of one **Header.HotTrackFlags** property. These are **Header.HotTrackFlagsStyle3D** and **Header.HotTrackFlagsStyleFlat** properties respectively. The default setting for the 3D style is `iGHdrHotTrackFlags.Text` to provide the same default hot-track effect like in the previous versions of iGrid. The default setting for the flat style is `iGHdrHotTrackFlags.Background` to provide the similar hot-track effect for row headers and column headers described above.

   To upgrade your code for iGrid X, rename the 'HotTrackFlags' property references in the **iGrid.Header** object to 'HotTrackFlagsStyle3D' or 'HotTrackFlagsStyleFlat' depending on the used header style.

3. [New] The background color of the hot column header can be customized now with the new **iGrid.Header.HotTrackBackColor** property. The default value of this property is `Color.Empty`, which means that the native effect of the current appearance is used (the predefined system color if iGrid is rendered with the OS visual styles or a lighter header background color for 3D/flat styles).

   The background color of the pressed column header can be customized the same way with the new **iGrid.Header.PressedItemBackColor** property.

   These two new color properties of the header area together with other color properties give you full control over the color theme of the header area.

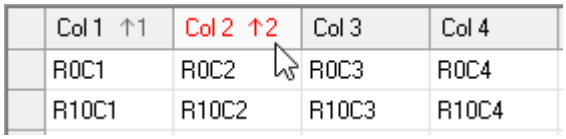
4. [New] The colors of the hot and pressed items in the row header area can be customized with the new **HotTrackBackColor** and **PressedItemBackColor** properties of the **iGrid.RowHeader** object property. These two new color properties together with other color properties of the **iGrid.RowHeader** object give you full control over the color theme of the row header area like the similar color properties for the column header area.

5. [Enhancement] The dragged column header is rendered semi-transparent using the effective column header background color – in contrast to the previous versions in which this color was lost.

6. [Enhancement][Change] If the flat style is used to render iGrid cells, cell combo buttons and cell ellipsis buttons use the same hot-track effect like row headers and column headers. In the previous versions of iGrid, they were highlighted with a dark extra border inside buttons, but now only a lighter background is used. This brings visual consistency in the flat style as all its UI elements supporting hot tracking (row and column header, cell ellipsis and combo buttons, standard scroll bar parts and custom scroll bar buttons) use the same hot-track effect.

7. [Enhancement][Change] Check boxes in the flat style correspond to the flat style of WinForms check boxes to provide consistent look in the whole app. The same concerns the hot-track effect used now for check boxes in the flat style. Earlier a dark extra border inside check boxes was used for hot tracking, but now their background is highlighted, and this make consistent look with other cell buttons and headers in the flat style.

8. [Enhancement][Change] If the **iGHdrHotTrackFlags.SortInfo** flag was specified in one of the header hot-track flag sets, the previous versions of iGrid highlighted only the sort text in the active column header. In the current version of iGrid, the new sort arrow is used in the sort info and it is highlighted together with the sort text in the active column header to provide a more consistent hot-track effect in the column header:



   Both the column header text and the sort info are highlighted with the color specified in the **Header.HotTrackForeColor** property.

9. [Fixed] If visual styles were not used in iGrid, the sort info text in the active column header changed its color to the color specified in the **Header.HotTrackForeColor** property even when the `iGHdrHotTrackFlags.SortInfo` flag was not specified in the **Header.HotTrackFlags** property.

10. [New] The event data object of the **RowHdrDynamicBackColor** event has two new properties, **State** of the **iGControlState** type and **Selected** of the Boolean type. They allow you to choose the color of a row's header depending on the row's header state (normal/hot/pressed) and the row's selection status.

11. [Fixed][Code-Upgrade] If you specified a custom background color for a row's header with the **RowHdrDynamicBackColor** event, this row's header under the mouse pointer was always highlighted as hot using the derived lighter color – even if this was not needed in the current drawing style (for example, the 3D style).

    If you used this effect in the previous version of iGrid, now you should code it explicitly yourself in the event handler of the **RowHdrDynamicBackColor** event based on the value of the new **State** property of the event data object.

High-resolution Screens and Tablets Support

iGrid X brings much better support for high-resolution screens and tablets by providing your users with the ability to make UI elements bigger so that they become usable on screens with high pixel density. Out-of-the-box iGrid automatically scales its UI elements according to the DPI value available for the app. In addition to this, iGrid provides you with new coding tools to adjust the sizes of the UI elements manually if you want to achieve effects differ from the automatic scaling. This section of the document describes the visual improvements and new members related to these tasks.

Pay attention to the fact that a C# or VB.NET WinForms app project created in a Visual Studio is not DPI aware by default. To make it possible, we recommend enabling this feature with the corresponding key in the app.config file:

```
<System.Windows.Forms.ApplicationConfigurationSection>
  <add key="DpiAwareness" value="PerMonitorV2" />
</System.Windows.Forms.ApplicationConfigurationSection>
```

For more info, read the High DPI support in Windows Forms article on the MSDN.

1. [Enhancement] In the previous versions of iGrid sort icons were not scaled at all and remained exceedingly small in apps supporting high DPI values. The size of the new sort arrows is linked to the size of the header font. Generally the size of a font is specified in points, which yields to automatic scaling of characters to provide the same linear sizes on screens with different DPI values. As a result, the new sort arrows are changed in size together with the header font and they have the same size comfort for eyes.

2. [Enhancement] In the previous versions of iGrid the cell check boxes and plus/minus buttons were not scaled according to the current DPI if iGrid was rendered with the 3D or flat style. Now these UI elements are increased in size according to the available DPI value.

3. [New][Enhancement] As earlier, the size of combo and ellipsis buttons is linked to the size of the scroll bars in the OS. The user may explicitly or implicitly change this setting in the OS to get adequate usable scroll bars in all apps - for example, by setting the display scale factor for a tablet screen to 200% or 250% in the Windows 10 settings. As a result, iGrid also changes the size of its scroll bars, combo and ellipsis buttons accordingly to provide consistent look.

   What is new in this release is that the sizes of cell check boxes and plus/minus buttons are also linked to that base scroll bar size value with some coefficients (0.8125 and 0.6 respectively). If the base scroll bar size is changed, all cell controls are changed in size harmoniously to provide the best look. The only exclusion is the cell check box when iGrid is rendered with the OS visual styles: this styled UI element never changes its size because the system provides the styled check box bitmap of a fixed size corresponding to the current screen dpi.

   The drawing of glyphs in cell controls (the tick mar in check box, the plus and minus sign in plus/minus buttons, etc.) was rewritten in this release of iGrid to support changeable control sizes. If the size of a control becomes bigger, the glyph pen become thicker to provide a better look. The new drawing algorithms take into account not only traditional 96dpi screens but high-resolution screens as well to provide perceptible cell control glyphs.

4. [New] iGrid X provides you with the ability to specify your own base size for all elementary controls in cells and column headers instead of the system scroll bar size used by default as described above.. These are combo buttons, ellipsis buttons, X-buttons, plus/minus buttons, check boxes. It is a very important new feature because actually it allows you to adjust the size of cell and column header controls for your particular needs – for example, to make them easily usable with fingers on touch screens.

This functionality is provided by the new **iGrid.ElemControlBaseSizeOverride** property of iGrid. This is a nullable integer property that specifies the desired base size of elementary controls in pixels. The default value of the property is `null`, which means that the system scroll bar size is used as the base value to calculate sizes of elementary controls as described above. To specify your own base size, assign the corresponding non-negative value to this property.

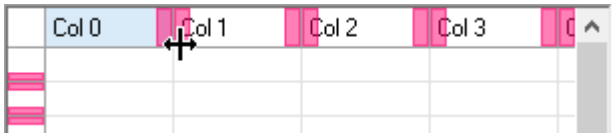
The base size value used by iGrid is returned by the new read-only **iGrid.ElemControlBaseSizeAutoValue** property. In this release it returns the system scroll bar size, but if the base size value provider will be changed in the future versions, you can always retrieve it with this property without changing your code.

The effective base size value for elementary controls can be retrieved with the read-only **iGrid.ElemControlBaseSize** property also introduced in this release.

Pay attention to the fact that you can assign 0 to the **iGrid.ElemControlBaseSizeOverride** property. If you do so, you will remove all elementary controls from cells. This feature can be used to hide all cell controls with one statement.

5. [New] iGrid X provides you with two new settings related to interactive column and row resizing. They allow you to adjust grid for more comfortable use on high-resolution screens and touch panels.

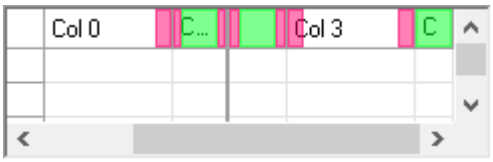
   The first of these configurable parameters is the size of the area around column and row dividers in which the mouse pointer turns into the resize cursor to indicate that the user can resize the corresponding object. These resize areas are marked with pink rectangles on the picture below:

   The sizes of these areas are configured independently for columns and rows with the new integer properties of iGrid named **ColResizeAreaExtentToDivider** and **RowResizeAreaExtentToDivider**. They specify the size of the resize area in pixels on both sides of the column and row dividers and default to 8 and 4 respectively.

   The second configurable parameter related to object resizing is the so called 'protected space' - the area between two consequent dividers that should not be covered by the resize areas as much as possible. This is an important concept of the object resizing functionality because the user always has some space in a resizable object they can click or grab to drag the object to another place.

   If the column width becomes small enough, iGrid automatically decreases the resize areas to provide maximum room for the protected space. The following picture demonstrate this concept:

   The same grid with 5 columns from the previous screenshot was modified to demonstrate the concept of protected space. We froze two first columns, decreased the size of the second and fifth columns, and scrolled the grid in the horizontal direction. The pink rectangles mark resize areas, the green rectangles mark protected spaces.

   The second column is small enough to provide full resize areas together with the protected space, so the resize areas in this column were decreased. The width of the third column was not decreased, but this column is partially hidden by the frozen area, so its resize areas were also decreased. The last column is so small so that it does not provide resize areas at all.

The size of protected spaces for columns and rows is set with the new integer **ColResizeAreaProtectedSpace** and **RowResizeAreaProtectedSpace** properties of iGrid. They specify the size of protected spaces in pixels and default to 12 and 8 pixels respectively.

6. [New] The thickness of iGrid scroll bars can be now set independently from the OS settings. This allows you to make iGrid scroll bars thicker for touch-oriented interfaces or thinner if you want to provide more space for cells.

   You can specify your custom height for the horizontal scroll bar and custom width for the vertical scroll bar with the new **HeightOverride** and **WidthOverride** properties of the **iGrid.HScrollBar** and **iGrid.VScrollBar** objects respectively. The default value for these properties is `null`, which means that iGrid sets the thickness of each scroll bar automatically depending on the system settings. To specify a custom scroll bar thickness in pixels, assign a positive integer value to the corresponding property. For example, the following statement will set the width of the vertical scroll bar to 25 pixels:

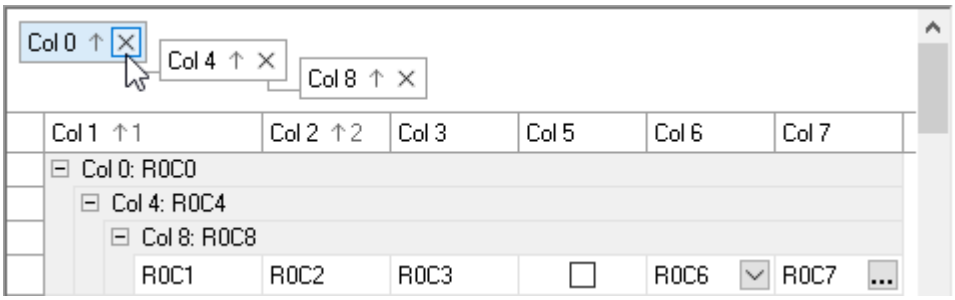
   ```
   iGrid1.VScrollBar.WidthOverride = 25;
   ```

   Pay attention to the fact that these new properties allow you to even set the thickness of the vertical scroll bar and horizontal scroll bar independently from each other if required.

7. [Enhancement] In the previous versions of iGrid row header glyphs (the triangle of the current row, the pencil icon reflecting editing state, etc.) kept their size in pixels on screens with high dpi values and remained very small. Now they are scaled according to the current dpi so that the user can recognize them easily.

X-buttons in Column Headers inside Group Box

1. [New] This release of iGrid provides you with a new possibility to remove column headers from the group box. This is done with the help of the new removal buttons added by default to every column header in the group box. These buttons look like the X letter, and the term X-buttons is used in all iGrid related members to refer this new UI element because of this:



   The main benefit of the X-button is the following. When you click this button, not only is the corresponding column header removed from the group box, but it is also placed on its original place in the grid. This eliminates the need to scroll grids with many columns in the horizontal direction to find the original place of a column. This saves much time and efforts for your users, especially when they work with touch screens.

   The visibility of the X-buttons is controlled with the new Boolean **ColHdrXButtons** property of the **iGrid.GroupBox** object. The default value of this property is `True`.

2. [New] The X-button is an elementary control. To refer to it in iGrid members, a new `XButton` item was added to the **iGElemControl** enumeration.

   This allows you to use existing iGrid members to adjust the functionality of X-buttons. For example, the **RequestColHdrElemControlToolTipText** event provides you with the **ElemControl** property in its event data object. Having it, we can show tooltips for column header X-buttons explaining their purpose:

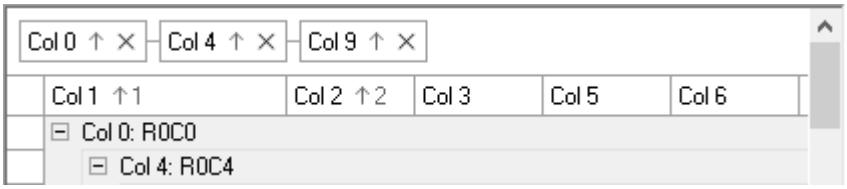
```
private void iGrid1_RequestColHdrElemControlToolTipText(
  object sender, iGRequestElemControlToolTipTextEventArgs e)
{
  if (e.ElemControl == iGElemControl.XButton)
  {
    e.Text = String.Format("Remove {0} from Grouping",
                           iGrid1.Cols[e.ColIndex].Text);
  }
}
```

3. [New] The new **ColHdrXButtonClick** event was implemented. It is raised when a column header X-button is clicked. The event data object properties tell you the column header in which the X-button is clicked (the **ColIndex** and **RowIndex** properties) and allow you to cancel the X-button action with the Boolean **DoDefault** property.

4. [New] For commonality with the **CellClick** event, the event data parameter of the **ColHdrClick** event was supplemented with the new **ElemControl** property of the **iGElemControl** type. This allows you to distinguish clicks on X-buttons and combo buttons in column headers and disable their actions if required dynamically using the **DoDefault** property of the event data object.

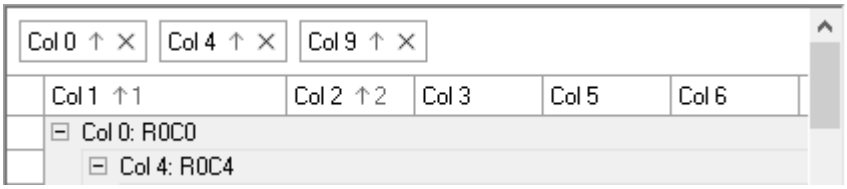
Advanced Group Box Settings

1. [New] iGrid X allows you to place the column headers in the group box in one row. This layout provides a modern look and can be used to save vertical space, which is important for tablets and laptops equipped with relatively small screens:



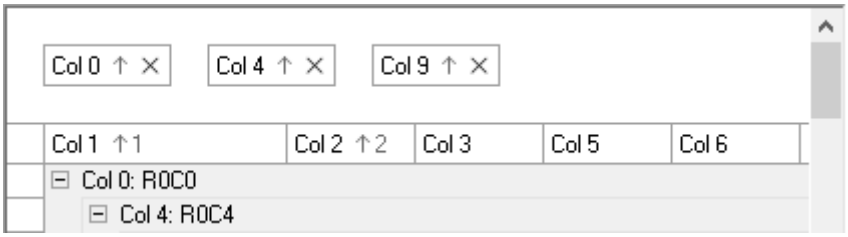
   The group box layout can be specified with the new **iGrid.GroupBox.Layout** property. It accepts one of the values from the new **iGGroupBoxLayout** enumeration – `iGGroupBoxLayout.Row` or `iGGroupBoxLayout.Steps`. The former value provides the new one-row look and is the default setting, the latter value provides the classical stairway or stepped look used in the previous versions.

2. [New] To make the group box look is even cleaner, you can turn off the lines between column headers:

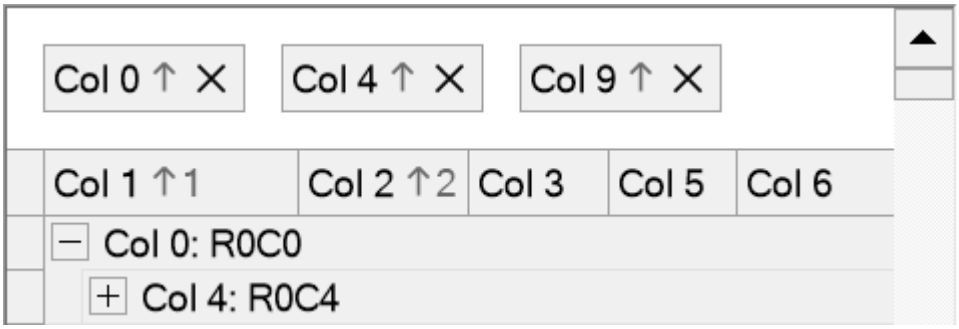


   The visibility of these lines is controlled with the new Boolean **iGrid.GroupBox.ConnectorLines** property set to `True` by default.

3. [New] The space around every column header in the group box is also configurable now. You can add more empty space around column headers with the new **iGrid.GroupBox.ColHdrSpace** property:



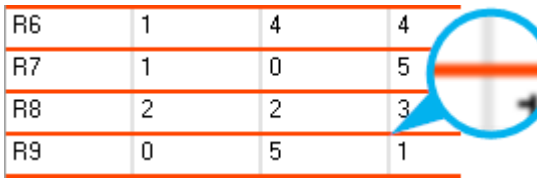
   This new integer property specifies the size of empty space at each side of the column header in pixels. It can be used together with the ability to increase font, sort icon, cell controls and scroll bars to provide a true touch-friendly interface in various drawing styles of iGrid:



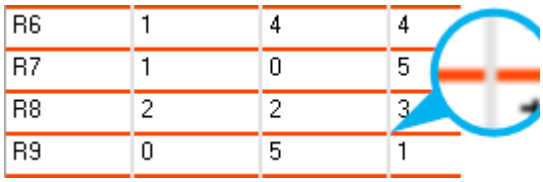
Improvements in Grid Lines Drawing

1. [New] If you use vertical and horizontal grid lines of the same color, there is no difference whether vertical grid lines are drawn over horizontal grid lines or vice versa. But if their colors differ, the order in which the vertical and horizontal grid lines are drawn yields different effects:

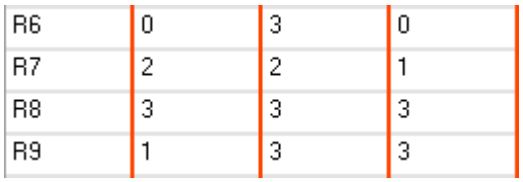
Horizontal over vertical:



Vertical over horizontal:



The previous versions of iGrid always drawn horizontal grid lines over vertical ones, which made impossible effects like on the following screenshot when columns are divided by vertical solid lines:



This release of iGrid allows you to specify the z-order of the vertical and horizontal cell grid lines to achieve effects like on the screenshot above. This is done with the new **iGrid.GridLines.ZOrder** property. This property accepts one of the two values from the **iGGridLinesZOrder** enumeration – `HorizontalOverVertical` or `VerticalOverHorizontal`. The default is `HorizontalOverVertical`, which corresponds the behavior of iGrid in the previous versions.

2. [New] The cells in the cell area are divided with the grid lines of 3 kinds:

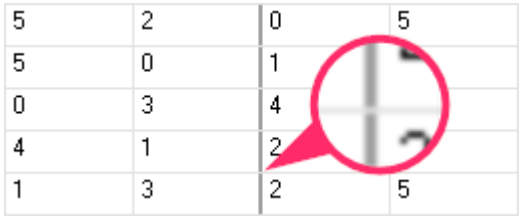
   (i) Normal grid lines (the **Vertical** and **Horizontal** properties of the **iGrid.GridLines** object)

   (ii) Grid lines bounding frozen areas (the **ColsEdge** and **RowsEdge** properties of the **iGrid.FrozenArea** object)

   (iii) Grid lines bounding the whole cell area (the **VerticalLastCol** and **HorizontalLastRow** properties of the **iGrid.GridLines** object)

   iGrid X introduces the notion of priority for grid lines of different kinds. Now every kind of grid lines is drawn in its virtual plane. The lower the priority of a grid line, the lower plane it is located in. The normal cell grid lines have the lowest priority, then the frozen edges go, and the lines bounding the whole cell area have the highest priority.

   As a result, grid lines of one kind cannot intersect grid lines of another kind. This new system guarantees that normal grid lines never break frozen edges as it was in the previous versions of iGrid:

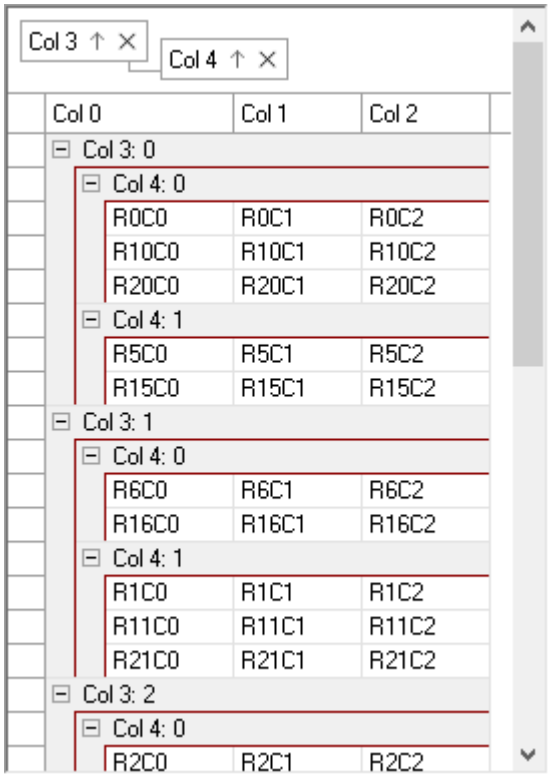


   The grid lines bounding the whole cell area also always look solid now and cannot be broken by normal cell grid lines or frozen area edges like it was in the previous versions of iGrid.

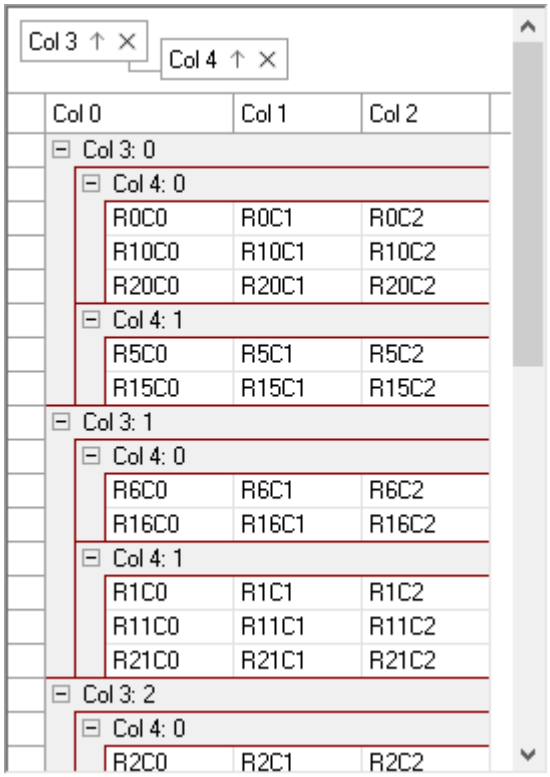
   The **GridLines.ZOrder** property introduced in this release actually specifies the z-order of vertical and horizontal grid lines in the plane of every kind of grid lines.

3. [Enhancement] If you specified a custom style for the grid lines in group rows with the **iGrid.GridLines.GroupRows** property, these lines are drawn above group rows too to provide a better look of groups. Compare the look of the same grid in which the group row grid line style is set to 'dark red' in the previous version of iGrid to what we have in iGrid.NET X now:

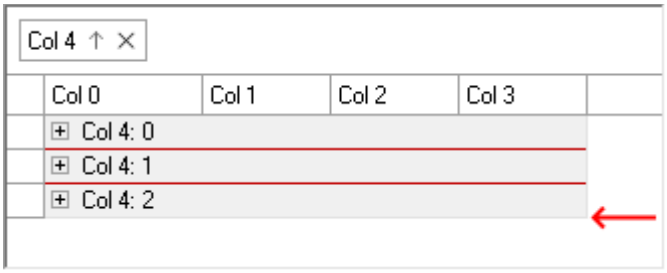
iGrid.NET 6:



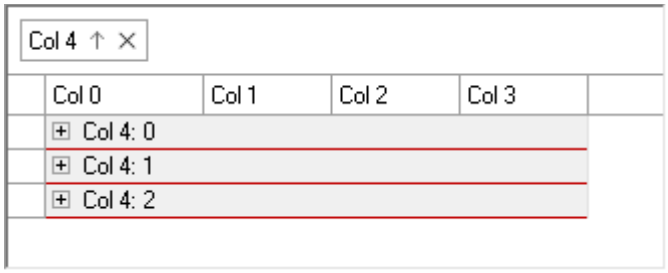
iGrid.NET X:



4. [Enhancement] If you specified a custom style for the grid lines in group rows with the **iGrid.GridLines.GroupRows** property and the user collapsed all group rows, the horizontal grid line in the last visible collapsed group row was drawn with the style defined for the horizontal grid line in the last row (the **iGrid.GridLines.HorizontalLastRow** property):

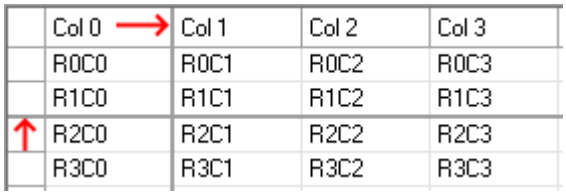


This looked inconsistent and even could result in decreasing the height available for the group row text if the last horizontal grid line had width greater than the default 1 pixel. This version of iGrid draws the horizontal grid line in the last visible collapsed group row with the style defined in the **iGrid.GridLines.GroupRows** property:



5. [Enhancement] All major spreadsheet apps like Microsoft Excel or Google Sheets extend the lines bounding frozen areas into their column and row header areas. The behavior of iGrid in the previous versions was different: only the special line bounding frozen columns was extended into the corresponding column header and only when the **Header.DrawSystem** property was set to `False`.

   This release of iGrid mimics the behavior of the major spread sheet apps for edges of the frozen areas. The vertical edge of the frozen area is extended in the header area for all flat-look drawing styles – the default visual style and flat style. The special line bounding frozen rows is also extended into the corresponding row header now for all flat-look styles:



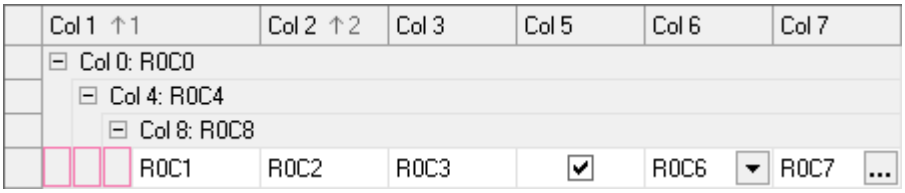
6. [Enhancement] The separating line for the footer section is not broken by the vertical grid lines similarly to the horizontal grid line in the last row of the cells area.

7. [Change][Code-Upgrade] If you specified 0 as the value of the width of the grid lines separating frozen columns and frozen rows (the **ColsEdge** and **RowsEdge** properties of the **iGrid.FrozenArea** object), iGrid drew normal grid lines instead of these separating lines. This behavior was inconsistent with other properties defining grid lines in which zero width means the absence of the corresponding grid line and it was fixed in this release of iGrid.

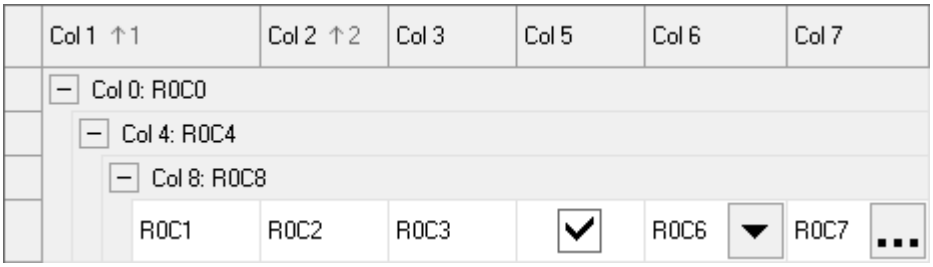
   If you used this feature in the previous versions of iGrid, now you must specify the normal grid line style explicitly in the **ColsEdge** and **RowsEdge** properties of the **iGrid.FrozenArea** object.

Controlling Level Indent Size

The **iGrid.LevelIndent** property in the previous versions of iGrid allowed you to control the size of the level indent area indicating row hierarchy levels (level indents are marked with pink rectangles on the following picture):



However, the plus/minus button could change its size in many situations. For example, the iGrid appearance defined by its **Appearance** and **UseXPStyles** properties, the indent around plus/minus button changed, custom draw plus/minus buttons were used. This release of iGrid with its support for high-resolution screens and the ability to change the size of controls with the new **ElemControlBaseSize** property adds new scenarios in which the plus/minus buttons may change dynamically. As a result, the grid will look not so good with the default size of level indent like on the following screenshot of iGrid from a screen with high dpi value:



To make level indent areas correspond to the size of the current plus/minus buttons with indents around them, you needed to adjust the value of the **iGrid.LevelIndent** property accordingly. This is a tedious task because you need to take into account many UI parameters and there are many situations when you need to do this.

To avoid the need in coding these operations in every form, iGrid X automatically sets the appropriate size of level indent based on the plus/minus button size and indent around it when anything related to these visual parameters has changed. Because of this change, the **iGrid.LevelIndent** property became a read-only property and can be used only to retrieve the effective level indent width.

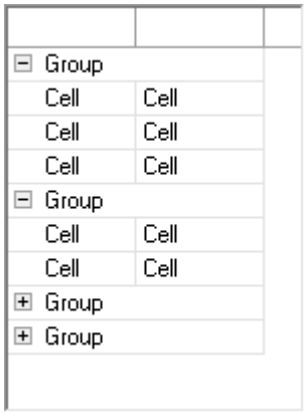
This default behavior can be turned off and you can still specify the required fixed size of level indent. This is done with the help of the new **iGrid.LevelIndentOverride** property. If this property equals `null` (the default value), iGrid changes the level indent size automatically as described above. To specify your custom level indent, assign the corresponding non-negative value in pixels to this property. If you specified your custom level indent in the previous versions of iGrid with its **LevelIndent** property, now you must do it with the **LevelIndentOverride** property.

There is one more new property related to the level indent area introduced in iGrid X. It is a read-only **iGrid.LevelIndentAutoValue** property returning the optimal size of level indent calculated by iGrid automatically for the best look. You can use this value at any time regardless of the value in the **LevelIndentOverride** property if you need to know the optimal size of the level indent area.

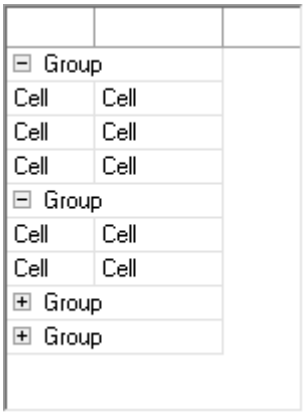
Pay attention to the fact that the **iGrid.LevelIndentOverride** property accepts non-negative integer values, which includes the zero value. This means that you can completely remove level indents in rows on the screen but still have hierarchical relations between rows if you set the level indent size to 0.

This can be helpful in situations if you use one level of parent-children relations and you want to save some horizontal space by removing level indents in child rows. Compare how a grid with manual group rows can look when the default and zero level indents are used:

Default level indent:



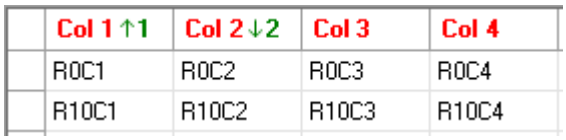
Level indent of zero:



The previous versions of iGrid did not allow you to use zero level indent due to restrictions of the drawing algorithm, but the new version of iGrid removes this limitation.

New and Updated Color Properties

1. [New][Removed][Renaming][Code-Upgrade] The new **SortInfoColor** property of the **iGrid.Header** object allows you to specify the color of the sort info area elements - sort icon and sort text. This can be useful if you want to make them differ from column titles, which was not possible in the previous versions:



   The **Header.SolidSortIconColor** property used to specify the color of solid triangle sort icons in the previous versions was removed. Actually the new **Header.SortInfoColor** property does the same for the new arrow sort icons. If you want to apply the previous color settings for solid triangle sort icons to the new sort arrow icons, replace all occurrences of the **SolidSortIconColor** property with **SortInfoColor** in your code.

2. [New] The new **ControlsForeColor** property of the **iGrid.Header** object can be used to specify the foreground color of controls inside column headers (the glyphs in combo buttons and X-buttons).

   iGrid does not provide a similar property to specify the background color for header controls because they are drawn without the background part until the user places the mouse pointer over them. The background bitmap of the hot column header control is determined automatically depending on the current drawing style and color properties of the header.

3. [Change][Code-Upgrade] The color inheritance scheme used in the previous builds of iGrid was ambiguous and too complex to use. For example, if the **Header.ForeControl** property was not set, it inherited the value from the **iGrid.CellCtrlForeColor** property, which in its turn if not set inherited the value from the **iGrid.ForeColor** property, which as an ambient property may inherit its value from the parent container (generally a form). As a result, you may have had an unexpected result if you wanted to set specific colors for the cell controls only without affecting the header or column header and cell controls changed their colors if the form's **ForeColor** or **BackColor** properties changed.

   To avoid any ambiguity and simplify the usage of the iGrid control, in this version all iGrid color properties related to the column header area (**Header.ForeColor**, **Header.BackColor**, etc.), group box (**GroupBox.BackColor**, **GroupBox.ColHdrBorderColor**, etc.), row header area (**RowHeader.BackColor**), cell controls (**iGrid.CellCtrlForeColor** and **iGrid.CellCtrlBackColor**) and scroll bars (**ScrollBarSettings.ForeColor** and **ScrollBarSettings.BackColor**) no longer inherit values from any parents when they are set to `Color.Empty`. By default all these color properties representing controls are set to non-empty values - mainly `SystemColors.Control` and `SystemColors.ControlText` depending on the part of the control they represent (background or foreground respectively). If you set one of these properties to `Color.Empty`, the corresponding iGrid part is simply not drawn. If you need to inherit colors in one header area from another area, now you should specify it in your code explicitly to avoid any undesirable effects.

4. [Change] To provide a more elegant and modern look of iGrid with the default settings, the default values for the following properties were changed:

   - The new default color for cell grid lines is `SystemColors.ControlLight`.
   - The new default color of lines separating the frozen area from normal cells (the **RowsEdge** and **ColsEdge** properties of **iGrid.FrozenArea**) is `System.ControlDark`.
   - the group box color properties **BackColor**, **HintBackColor** and **HintForeColor** are set to `SystemColors.Window`, `SystemColors.Window` and `SystemColors.GrayText` respectively by default.

Speed and Resource Usage Optimizations

1. [Optimization][New] This release of iGrid brings new and improved tools to read/write cell values.

   First of all this concerns the **iGCell.Value** property. It works 15% - 20% faster when you set cell values and up to 30% faster when you read cell values.

   Another new iGrid member you can use to read/write cell values instead of **iGCell.Value** property is the indexed **iGrid.CellValues[·,·]** property. This property provides access to cell values using the array syntax. It is indexed by row and column indices or string keys like the **iGrid.Cells[·,·]** property. For example, if you populated cells in a column using a loop like this:

   ```
   for (int i = 0; i < iGrid1.Rows.Count; i++)
     iGrid1.Cells[i, 0].Value = i;
   ```

   , now you can use the new **CellValues** property instead:

   ```
   for (int i = 0; i < iGrid1.Rows.Count; i++)
     iGrid1.CellValues[i, 0] = i;
   ```

   The main benefit of this property is that it provides 60% performance gain compared to the **iGCell.Value** property in the previous versions of iGrid because every access to a cell value is done without creating an intermediate **iGCell** object returned by the **iGrid.Cells[·,·]** call. One more result of this approach is that memory is not filled with many useless **iGCell** objects, which led to more frequent garbage collection operations in the previous version of iGrid.

   The performance improvements described above also lead to some acceleration of the **FillWithData** method used to populate iGrid from data sources as the method is based on the same internal routines.

2. [Optimization][Removed][Code-Upgrade] The **RequestDropDownControl** event and all related iGrid members (like the **iGRequestDropDownControlEventArgs** class) were removed in this release of iGrid. The ability to request the drop-down control for a cell dynamically provided by this event hasn't been used much in real-world apps since the first versions of iGrid, and we decided to remove it from iGrid after analyzing the iGrid behavior under memory profilers. The fact is that the **RequestDropDownControl** event was fired very frequently in many parts of iGrid because it affected the look and behavior of cells – for example, when a cell value was set or when the user was moving the mouse pointer over iGrid cells. All this led to useless creation of numerous objects like **iGRequestDropDownControlEventArgs** class in memory and was one of the bottleneck of performance because of garbage collection operations.

   If your code was based on the functionality of this event and you want to upgrade it for this release of iGrid, the same task can be implemented with the equivalent **iGCell.DropDownControl** property. This change will also be helpful for your app for the performance reasons described above.

3. [Optimization] The internal drawing routines of iGrid were reviewed and changed to use .NET objects related to drawing and text output (such as the **StringFormat** object) more effectively. The updated finalization code blocks of internal classes dispose used resources more intensively. This results in less memory consumption and fewer garbage collection operations during the lifetime of an app with iGrid instances. This is especially important for huge grids with frequent redrawing operations or computers with low-level technical characteristics, such as budget tablets.

4. [Optimization] The iGrid methods used to render its cells and header areas (**DrawCellContents**, **DrawColHdrContents**, **DrawRowHdrContents**, **DrawRowHeaderColHdrContents**) create less **Graphics** objects in every call. This leads to much smaller number of **Graphics** object creations while printing or print-previewing iGrid with the PrintManager add-on and less load on the garbage collector.

5. [Optimization] The code that renders iGrid using the OS visual styles (the default settings) was optimized for faster performance. Creation of object returning system information about visual styles was also optimized to save memory and minimize the number of garbage collection operations.

6. [Enhancement] iGrid's built-in implementation of drop-down lists, the **iGDropDownList** class, implements the **IDisposable** interface and provides you with the implementation of the interface's **Dispose()** method to free the resources used by the drop-down list when the drop-down list is no longer needed. However, not all developers call this method, which leads to resource leaks. To help the developers to avoid this issue when instances of the **iGDropDownList** class are created in the Windows Forms designer, the class implements a new overloaded constructor with the **IContainer** parameter:

```
public iGDropDownList(IContainer container)
```

When a new instance of the **iGDropDownList** class is created at design time by dragging its icon from the Visual Studio Toolbox onto the form, the Windows Forms designer finds this constructor and uses it in the generated code:

```
private void InitializeComponent()
{
  this.components = new System.ComponentModel.Container();
  …
  this.iGDropDownList1 = new
    TenTec.Windows.iGridLib.iGDropDownList(this.components);
  …
}
```

This guarantees that the **iGDropDownList.Dispose()** method will be called automatically in the standard **Dispose()** implementation of the form generated by the Windows Forms designer:

```
protected override void Dispose(bool disposing)
{
  if (disposing && (components != null))
  {
    components.Dispose();
  }
  base.Dispose(disposing);
}
```

We also strongly recommend that you check your code to be sure you dispose the resources of all used **iGDropDownList** objects. One of the easiest ways to fix this problem for drop-down lists created in the Windows Forms designer is to insert the **this.components** value into the **iGDropDownList** constructors generated by the Windows Forms designer. If you create **iGDropDownList** objects from scratch in your code, call the **Dispose()** method for them when these lists are no longer needed.

iGBool: New Rules

VB.NET makes implicit type conversion possible due to the **Option Strict Off** statement effective by default. Having this, many developers using Visual Basic expect that they can use the intrinsic Boolean `True` and `False` values to work with iGrid properties of the **iGBool** type like in the following statement:

```
iGrid1.Cols(1).CellStyle.SingleClickEdit = False
```

However, this code did not work as expected in the previous versions of iGrid: the **SingleClickEdit** property was actually set to `True`. This happened because the numeric equivalent of `False` in VB.NET is 0 and the **iGBool** enumeration was defined like this in the previous versions of iGrid:

```
public enum iGBool
{
  True = 0,
  False = 1,
  NotSet = 2
}
```

To avoid such problems hard to find and save time spent on debugging, we changed the numeric values of the **iGBool** members and introduced a new rule for **iGBool** properties based on customer requests.

First, the **iGBool** enumeration is defined like this in iGrid X:

```
public enum iGBool
{
  False = 0,
  True = 1,
  NotSet = 2
}
```

This number set corresponds to intuitive understanding of numeric constants equivalent to the logical `True` and `False` values, and it is also compatible with existing .NET enumerations like **System.Windows.Forms.CheckState**.

Second, if **Option Strict Off** is in effect, you can assign any **Int32** value to an **iGBool** property – including the case of VB `True` which equals -1. The new processing rule introduced in iGrid X treats any integer value stored in an **iGBool** property like `True` if it does not equal to `iGBool.False` or `iGBool.NotSet`.

These improvements make possible usage of the native VB `True` and `False` values in **iGBool**-based properties of iGrid. Pay attention to the fact that this concerns not only the `True` and `False` literals, but also any expressions returning Boolean values in VB (variables, control properties, etc.). This results in shorter and cleaner code:

```
iGrid1.Cols(1).CellStyle.SingleClickEdit = checkBoxSingleClickEdit.Checked
```

In the previous versions of iGrid, you would need to write something like the following statement for the same purpose or use the equivalent If/Else construction:

```
iGrid1.Cols(1).CellStyle.SingleClickEdit =
  DirectCast(IIf(checkBoxSingleClickEdit.Checked,
  iGBool.True, iGBool.False), iGBool)
```

If you did not use the numeric constants 0, 1 and 2 as equivalents of **iGBool** items in your programs, your code does not require any changes to work with iGrid X.

PrintManager Enhancements

1. [New] The **iGPrintCellControls** enumeration, which is the base type of the **PrintCellControls** property of the **iGPrintManager** class, contains the new `XButton` flag. This flag is used to control the visibility of x-buttons in the printed column headers.
2. [New] The previous versions of PrintManager used the color stored in the **GridLineColor** property of the **iGPrintManager** class to draw the borders around column headers in the group box. This release of PrintManager introduces the new **GroupBoxColHdrBorderColor** property to provide the ability to specify custom border color. The default value of this property is `Color.Empty`, which means that the value of the grid's **GroupBox.ColHdrBorderColor** property is used for this purpose.
3. [Change] The new sort icon from the flat style (the universal sort arrow) is used instead of the hard-coded solid black triangle to print sort icons in the plain printing style.
4. [Fixed] The combo button glyphs were drawn in the column headers in the group box area even if the **PrintCellControls** property of **iGPrintManager** did not contain the `ComboButton` flag.
5. [Fixed] If plain style was used for printing, check boxes with the indeterminate state were drawn as unchecked.

Changes in the IiGControlPaint Interface

iGrid provides the **IiGControlPaint** interface used to create classes providing custom drawing for iGrid elements through the **iGrid.CustomControlPaint** property. The changes in this interface are mainly related to the new visual elements and features introduced in this release of iGrid. They are listed below.

1. [New] The **iGHeaderPart** enumeration used in the **DrawHeader()** method of the **IiGControlPaint** interface was supplemented with the new **XButton** item to implement drawing of the corresponding interface item in custom drawn controls.

2. [New] The **IiGControlPaint** interface defines the two new Boolean read-only properties **HeaderHasBorder** and **RowHdrHasBorder**. They indicate whether the drawn column headers and row headers respectively look like fully finished areas that do not require grid lines around them. If the value of the property is `True`, iGrid does draw grid lines around the corresponding element.

3. [Change][Code-Upgrade] The **DrawHeader** and **DrawRowHdr** methods of the **IiGControlPaint** interface have the new **partColor**, **hotColor**, **pressedColor** parameters of the **Color** type and the **drawBackground** and **drawSystemBackground** parameters of the Boolean type.

   iGrid passes the color of the drawn part in the **partColor** parameter. The suggested hot and pressed colors of the item are passed in the **hotColor** and **pressedColor** parameters. They are used for column and row header backgrounds and generally contain the values of the **HotTrackBackColor** and **PressedItemBackColor** properties for the corresponding header area. The **drawBackground** and **drawSystemBackground** parameters also make sense only for column and row header backgrounds and specify whether to fill the background with the specified color and draw the system header effect.

4. [Change][Code-Upgrade] The following **IiGControlPaint** members were supplemented with the new **suggestedSize** parameter to give you the ability to use the suggested element size to look harmoniously with other UI elements:

   ```
   Size GetHeaderSortIconSize(Size suggestedSize)
   Size GetCheckBoxSize(Size suggestedSize)
   Size GetTreeButtonSize(Size suggestedSize)
   ```

   The suggested size for element of every type is calculated based on the base value defined by the new **ElemControlBaseSize** property of iGrid.

5. [Change][Code-Upgrade] The **GetScrollBarCustomButtonIndent** property of the **IiGControlPaint** interface was redefined as a method returning an **iGIndent** value. This is done for commonality with all other Get* methods of the **IiGControlPaint** interface returning indents and sizes.

   To upgrade existing code, change the property implementation to method implementation. For example, if you implemented **GetScrollBarCustomButtonIndent** as a property in C# code like this:

   ```
   public iGIndent GetScrollBarCustomButtonIndent
   {
     get
     {
       return new iGIndent(2, 2, 1, 1);
     }
   }
   ```

   , then the new version of the member will look like the following:

   ```
   public iGIndent GetScrollBarCustomButtonIndent()
   {
     return new iGIndent(2, 2, 1, 1);
   }
   ```

Other Changes and Enhancements

1. [Enhancement][Change][Code-Upgrade] Our analysis of user applications built with iGrid showed that in many cases texts and optional images in cells are centered vertically to look best. This also concerns column headers and texts displayed in group rows. All these settings require writing code and manipulations with the cell and column header style properties in column default cells (**iGrid.DefaultCol**), changes in the **iGrid.GroupRowLevelStyles[]** array and the like. All this may become tedious if you need to apply these settings to every grid in your applications.

   To shorten development time and to provide the best look out-of-the-box when an iGrid instance is placed on the form, iGrid X provides the default vertical centering for texts and images in cells, column headers and group rows. This improvement concerns the default values for the corresponding cell properties and their styles, and also cells created by the **FillWithData()** method used to populate iGrid with data from data sources.

   If you relied on the top vertical alignment in your grids in the previous versions of iGrid and want to keep it after upgrade to iGrid X, you need to specify the required alignment explicitly. The easiest way to do this is to use the corresponding statement(s) from the following ones in your code:

   ```
   iGrid.DefaultCol.CellStyle.TextAlign = iGContentAlignment.TopLeft;
   iGrid.DefaultCol.ColHdrStyle.TextAlign = iGContentAlignment.TopLeft;
   iGrid.GroupRowLevelStyles[0].TextAlign = iGContentAlignment.TopLeft;
   ```

   The same must be done for images if you use them: use the **ImageAlign** property of the iGrid cell style object for this purpose.

   And contrary, if you specified the default alignment for cell texts and images as `iGContentAlignment.MiddleLeft`, now these settings can be removed from your code.

2. [Enhancement][Change][Fixed] The drawing of vertical text in cells was reimplemented. The new algorithm draws cell texts by rotating the normal horizontal cell text 90 degrees clockwise or counterclockwise instead of using the `StringFormatFlags.DirectionVertical` flag in the **Graphics.DrawString()** call as it was in the previous versions. This allowed us to fix one annoying problem related to the `StringFormatFlags.DirectionVertical` flag when the text rows were rendered in the opposite order. The new approach improves some internal algorithms related to cell text drawing and calculation of cell text size and fixes various problems related to measuring of the vertical text in cells in auto-width and auto-height operations.

3. [Enhancement][Change] The **Bounds** property of the event data object of the **CustomDrawCellForeground** and **CustomDrawColHdrForeground** events does not include the area taken by the cell or column header contents indent (the **iGCell.ContentIndent** or **iGColHdr.ContentIndent** properties). This helps to write shorter code to draw custom foreground contents because the developer does not need to deduct content indent from the **Bounds** area as it was in the previous versions.

4. [Enhancement][Change] The cell content indent specified in the **iGCell.ContentIndent** property is also applied to cell buttons now (combo button, ellipsis button). This allows the developer to control the space between cell buttons and cell edges and align the buttons vertically together with the cell main contents like image and text. The same is applicable to column headers and their content indents specified in the **iGColHdr.ContentIndent** property.

5. [New] The **Header** and **RowHeader** properties of iGrid implement the new Boolean **ItemHasBorder** property indicating whether the items of the corresponding area look like fully finished areas and iGrid does not draw grid lines around them because of that. These members are intended for usage mainly by the internal infrastructure of iGrid from the PrintManager add-on.

6. [New][Removed][Code-Upgrade] Now you can adjust the indent around the tree (plus/minus) button with the new integer **iGrid.TreeButtonIndent** property. The **iGrid.GetTreeButtonIndent()** function returning the hard-coded tree button indent in the previous versions was removed as its functionality is implemented with the new **TreeButtonIndent** property.

7. [New] The **DrawGroupBox()** method of iGrid has two new Boolean parameters, **drawComboButton** and **drawXButton**, which can be used to control the visibility of combo buttons and x-buttons in the drawn group box area. The **drawXButton** parameter was also added to **DrawColHdrContents()** method for the same purpose.

8. [New] The **iGColHdr** class has two new methods that allow you to retrieve the text and image bounds depending on the placement of the column header – in the normal header row or in the group box:

```
Rectangle GetTextBounds(bool groupBox)
Rectangle GetImageBounds(bool groupBox)
```

   You can specify whether the column header is placed in the group box using the Boolean **groupBox** parameter. These two new methods are needed because the layout of column headers may differ depending on the placement area – column headers in the group box display the x-buttons by default.

9. [New] In the previous versions of iGrid, if you set the **RightToLeft** property of iGrid to `RightToLeft.Yes` to enable the right-to-left mode for the whole grid interface and its cells, you could not display left-to-right texts in particular cells. Now you can do that with the new member of the **iGStringFormatFlags** enumeration named `DirectionLeftToRight`. It can be used to force iGrid to use the left-to-right direction in particular cells regardless of any other settings that set the right-to-left text layout.

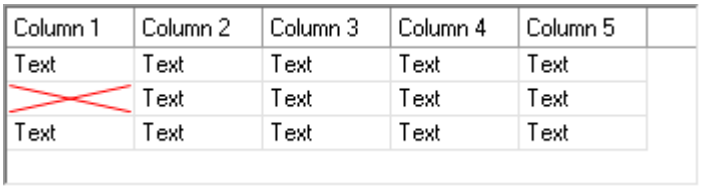
10. [New] Now you can determine whether a part of a column header or footer cell (text, image) is clipped with the new **IsCellPartClipped()** methods of the **iGColHdr** or **iGFooterCell** classes. These methods work like the similar **IsCellPartClipped()** method for cells in the **iGCell** class.

11. [New] The event data object of the **RowHdrDynamicBackColor** event has two new properties describing the row header item: the Boolean **Selected** property indicating whether the row is selected and the **State** property of the **iGControlState** type indicating the item state (normal, hot or pressed).

12. [Enhancement] iGrid uses GDI+ methods to render text in its cells. Sometimes these methods fail, and this does not depend on iGrid. For example, the **Graphics.DrawString()** method may fail if we pass an enough long string into it:

    https://stackoverflow.com/questions/14467663/gdi-drawstring-generic-exception-when-drawing-a-string-with-a-length-over-65536

    Previous versions of iGrid crashed in this case throwing the corresponding exception. To make the new version of product robust, we decided to use the try..catch construction to intercept such unpredictable errors while rendering cell texts. If an exception occurred while rendering cell text, iGrid informs about that with a red cross on a white rectangle on the place of cell text:



    We chose this way of informing about problems because .NET also does the same way if an exception occurs in a custom event handler of the **Paint** event for a control.

    Pay attention to the fact that the cell tooltip still works and allows the user to see the extra long cell text even if an exception occurred during cell text rendering.

13. [Removed][Code-Upgrade] The rarely used ability to turn off system drawing in the column header and row header areas provided by the **DrawSystem** property in the **iGrid.Header** and **iGrid.RowHeader** objects was removed. This removal simplified the header areas drawing conveyor as now only two properties of these areas affect their look – **Appearance** and **UseXPStyles**.

    Taking into account all changes in the drawing model of iGrid X, all visual features provided by iGrid when the **DrawSystem** property was `False` are duplicated in the flat drawing style. To upgrade your code from the previous versions of iGrid to be compatible with iGrid X, use the following settings instead of setting **DrawSystem** to `False` (an example for the column header area):

    ```
    iGrid.Header.UseXPStyles = false;
    iGrid.Header.Appearance = iGControlPaintAppearance.StyleFlat;
    ```

    The same works for the row header area if you replace **iGrid.Header** with **iGrid.RowHeader**.

14. [Change][Code-Upgrade] The **iGCol.AutoWidth()** and **iGColCollection.AutoWidth()** methods no longer analyze the **AllowSizing** property of the columns they are called for and adjust the column widths regardless of the value of this property. This enhancement was implemented to provide commonality with other members of iGrid because column properties whose names start with 'Allow' should enable/disable only the corresponding user actions but not actions performed from code.

    If you called these **AutoWidth()** methods in the previous versions of iGrid knowing that they do not affect columns with the **AllowSizing** property set to `False`, you should modify your code and call the **iGCol.AutoWidth()** method only for the columns allowing interactive width change.

15. [Enhancement][Change] The **iGRow.AutoHeight()** method was enhanced in this release.

    First, the **iGRow.AutoHeight()** method uses the contents of the row header in its calculations only if the row header is visible.

    Second, if you called this method in the previous versions of iGrid, it was considered that every row cell has at least one row of text even if a cell was empty to provide enough space for the cell if it would contain text in the future. In this release this is true only if the **Flags** property of the cell contains the `iGCellFlags.DisplayText` flag (the default setting). This is done to provide the ability to adjust row height based only on cell images in the case if row cells will never contain text.

    These enhancements are also applied to the **AutoHeight()** method of the row collection returned by the **iGrid.Rows** property.

16. [Enhancement][Change] The behavior of the **iGrid.GetPreferredRowHeight()** method was changed to provide the behavior symmetrical to the **iGRow.AutoHeight()** method. First, the **GetPreferredRowHeight()** method takes into account the row header only if it is visible. Second, the **GetPreferredRowHeight()** method skips cells in invisible columns.

17. [Enhancement] The `iGCellTypeFlags.NoTextEdit` flag can be used to disable text editing in cells with ellipsis buttons. Earlier you could do that only if you wrote the corresponding event handler for the **RequestEdit** event.

18. [Enhancement][Change] iGrid changed the selection of a row regardless of the mouse button used to click the row header. This made impossible displaying of a context menu by the right mouse button click on row headers without changing row selection. Now row selection is changed only if row headers are clicked with the left mouse button.

19. [Enhancement] If the sort info area is wider than the available space, its elements (sort icon and sort text) do not disappear but are drawn partially clipped. This helps to know whether a column is sorted even if its width is very small.

20. [Enhancement] Cell controls, such as check boxes and plus/minus buttons, are no longer squeezed if there is no enough space to draw them fully. As for cell buttons (combo and ellipsis buttons), they are squeezed only in the vertical direction if the row height is not enough to draw them fully.
21. [Enhancement] Sorting by the same column with repeated values keeps the original row creation order within the range of same values, which may be important if the user wants to see the rows in the original order in this case.
22. [Removed] The **systemBackgroundX** and **systemBackgroundWidth** parameters were removed from the **DrawColHdrContents()** and **DrawRowHeaderColHdrContents()** methods intended for internal usage mainly by the PrintManager add-on.

Fixed Bugs and Issues

1. [Fixed] If the user activated editing in a cell using the built-in text editor or a custom external editor based on the **iGCellEditorBase** class, the cell may have displayed remaining parts of text if the editor was not cover the cell text area completely.
2. [Fixed] Vertical header grid lines specified with the **Header.VGridLinesStyle** property were not drawn if the header used a system style for drawing (the **Header.DrawSystem** property was set to `True`).
3. [Fixed] A small mistake in detecting the hot column header under the mouse pointer in the group box was fixed.
4. [Fixed] The width of a column header in the group box was not adjusted automatically when the combo button appeared or disappeared in it (after adding/removing a drop-down control to the column header).
5. [Fixed] If the object in the **CustomControlPaint** property implemented custom drawing for the group box background, it was not used.
6. [Fixed] The semi-transparent image of the dragged column header did not cover the full rectangle of the column header in all cases.
7. [Fixed] If automatic adjustment of column header height was on, iGrid did not recalculate the row height after changing column header properties.
8. [Fixed] Various problems with drawing iGrid interface elements in the 3D style, especially noticeable in the right-to-left mode, were fixed.
9. [Fixed] The hot-tracking effect was not applied to cell check box correctly if the cell had other controls like the tree button.
10. [Fixed] iGrid did not take into account the tree button while calculating the optimal row height when the user double-clicks row divider or the **iGRow.AutoHeight()**, **iGRowCollection.AutoHeight()** methods were called.
11. [Fixed] If the drawing of extended grid lines beyond the cell area was turned on, the row header area below the real rows was not drawn if iGrid had frozen columns and the horizontal scroll bar.
12. [Fixed] The column headers inside the group box did not resize automatically to show column titles without clipping when the header font changed.
13. [Fixed] If the **iGrid.ShowControlsInAllCells** property was set to `False`, the bounds of various cell parts retrieved with properties like **iGCell.ImageBounds** were incorrect and the tooltips for invisible combo and ellipsis buttons appeared in a cell if it was not the current cell.
14. [Fixed] iGrid may have thrown an exception when the **FrozenArea.RowCount** was changed in stay-sorted mode.
15. [Fixed] If iGrid had columns with repeated cell values, editing of cells in stay-sorted mode led to reordering grid rows even if the new cell value did not change row order in the effective sort criteria.
16. [Fixed] iGrid may have frozen the app when displaying a tooltip for a custom scroll bar button if the tooltip text was long enough. This problem was caused by the bug in the system USP10.dll which handles Unicode layout of characters on screen when visual styles are on:

    http://social.msdn.microsoft.com/Forums/windowsdesktop/en-US/a5a95763-9f10-4640-9d32-afd050a60bb3/tooltip-rendering-extremely-slow-on-vista-due-to-wordwrap

    The problem is still not fixed in the Windows OS by the moment of releasing this major update of iGrid. To avoid this issue, iGrid truncates the tooltip text by 300 characters and adds an ellipsis at the end if the original tooltip text was truncated.

17. [Fixed] The Collapse All or Expand All commands performed with custom scroll bar buttons or the **PerformAction()** method caused incorrect calculation of the parameters of the vertical scroll bar.
18. [Fixed] If extended grid lines were drawn below real rows, the row headers in these fake rows were drawn using the `SystemColors.Control` color regardless of the value specified in the **iGrid.RowHeader.BackColor** property.
19. [Fixed] The **DrawRowHdrContents** method did not draw the row header glyph.
20. [Fixed] The grid lines of a merged cell did not use the style of the last column or row if required when the column and/or row containing the root cell were invisible.
21. [Fixed] iGrid used the hot-track effect for its column and row headers even if the **iGrid.HotTracking** and **iGrid.Header.HotTracking** properties were set to `False`.
22. [Fixed] Built-in drop-down lists and other drop-down controls opened from column headers in the group box were drawn at incorrect positions and/or with incorrect sizes. This problem also occurred for the AutoFilterManager filter boxes as they are based on the same drop-down control infrastructure.