10Tec iGrid ActiveX 7.x
What's New in the Latest Builds

V7.50, build 0020 | 2021-Oct-12

Windows 11 visual styles support

Updated cell control rendering engine

By default, iGrid renders its interface elements using the OS visual styles. Windows 10 and its predecessors provided us with the rectangular combo button and check box controls without semi-transparent areas. The drawing code in the previous versions of iGrid used a graphical control cache to draw those cell elements to provide maximal performance, especially while scrolling iGrid with a lot of cell controls.

Windows 11 brings a new visual style containing semi-transparent interface elements with rounded corners. As a result, the optimized cell control drawing code from the previous builds of iGrid no longer renders combo buttons and check boxes correctly in Windows 11. You can see almost black combo buttons and rounded check boxes inscribed into black squares in the previous builds of iGrid in Windows 11:

The cell control drawing code was rewritten in this release of iGrid to support the new Windows 11 visual styles. Now iGrid cell controls are rendered as expected:

Note regarding selection colors

The default setting of iGrid is to use the system selection colors, which are still a dark blue for background and white for selected text in Windows 11. You may find that the styled combo buttons and check boxes are very difficult to see against the default selection background in Windows 11. This especially concerns the semi-transparent combo button with the hot hover effect:

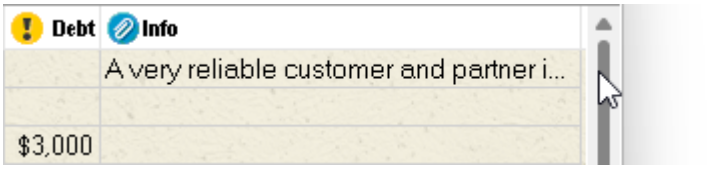
To improve the situation, you can use the selection colors hard coded in the Windows Explorer in the latest versions of Windows:

Pay attention to the fact that the color of the selected text is not changed, and the selected row is framed with a solid blueish line instead of the default dotted focus rectangle. You can gain this effect with the following simple settings:

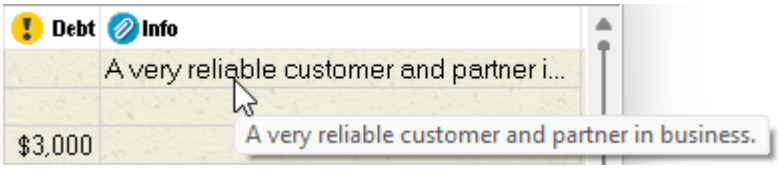
```
iGrid1.HighlightForeColor = RGB(0, 0 ,0)
iGrid1.HighlightBackColor = RGB(204, 232, 255)
iGrid1.FocusRectColor1 = RGB(153, 209, 255)
iGrid1.FocusRectColor2 = RGB(153, 209, 255)
```

Known issue with the vertical scroll bar update

We found one bug in Windows while working on our iGrid ActiveX control. This bug is present in Windows 10, but it is especially noticeable in Windows 11 with its new styled scroll bar. To reproduce it, first place the mouse pointer inside the over the vertical scroll bar to redraw it in the hot state when all items like the up arrow appears:



Now move the mouse pointer quickly inside iGrid into a cell or column header with clipped text so that the tooltip showing the full text should appear soon. You may notice that the upper part of the vertical scroll bar is not redrawn correctly:



This update problem occurs only in the area of intersection of the native Microsoft Header Control and the styled vertical scroll bar when a native Windows tooltip is attached to the control. These system components are used in iGrid to provide the look-and-feel of the OS, but they do not work correctly together in the latest versions of the OS. This is a problem not only of iGrid ActiveX: the standard ListView control based on the same elements also has this issue. We could reproduce this issue in the ListView control from the WinForms package in .NET Framework.

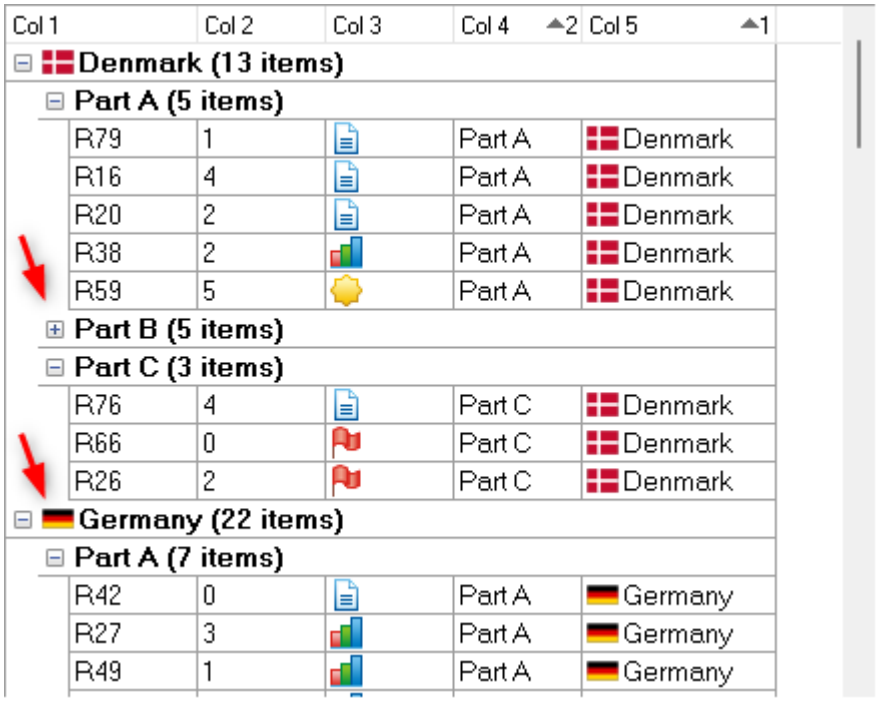
Unfortunately, there is no workaround for this problem we can implement in iGrid unless we disable core parts of the iGrid functionality. We reported about this problem to Microsoft and are looking forward to the fix in the future OS updates. If this problem annoys you or your customers, you can turn visual styles off in iGrid by setting its **UseXPStyles** property to False or suppress cell and column header tooltips in the event handlers of the **RequestCellToolTip** and **RequestColHeaderToolTip** events like in the following code snippet:

```
Private Sub grdCust_RequestColHeaderToolTip(ByVal lCol As Long, _
   sTipText As String, eTipIcon As EToolTipIcon, sTipTitle As String)

   sTipText = ""

End Sub
```

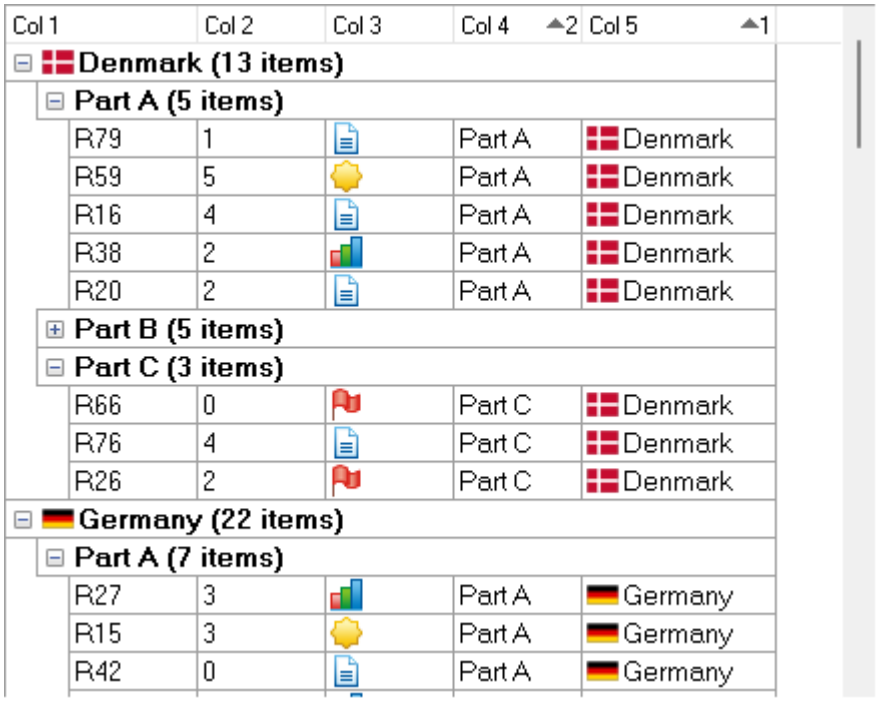
Enhancements in grouping functionality

1. [Enhancement] This build of iGrid draws more segments of horizontal and vertical grid lines around group rows to provide a complete and more appealing look of group rows. This is especially noticeable when

the background part of group rows is not filled with a color and/or the color of grid lines is set to a non-standard dark color. Look at the areas marked with the red arrows on the screenshot below:



This is the look of group rows in the previous builds of iGrid. The enhancement implemented in this release of iGrid provides a better and complete look of group rows as they are fully framed with grid lines:



2. [Enhancement] The plus/minus buttons in group rows are always vertically centered. In this build of iGrid the text of group rows is also vertically created by default. This gives a better look of group rows with big heights without additional coding because the default alignment of texts in group rows was set to 'top' in the previous builds of iGrid.

Other improvements

1. [Enhancement] You can retrieve the current sorting state of iGrid with the **ColSortKey**, **ColSortOrder** properties and the **SortObject** object property. In the previous builds of iGrid you could access these properties in the event handlers of the **BeforeContentsSorted** and **AfterContentsSorted** events, but iGrid did not guarantee you got the actual sorting state before cell values are sorted and after they are sorted respectively. Now this is possible due to the restructured internal functionality.
2. [Fixed] iGrid did not update correctly sort icons in column headers when its sorting state was restored with the **LayoutCol** property.
3. [Fixed] The ENTER and ESCAPE keys used to commit or cancel editing in a cell of the **igCellTextCombo** type may have not worked.
4. [Fixed] When the user activated a cell of the **igCellTextCombo** type, the combo button may have been drawn as pressed.
5. [Fixed] If the **ShowControlsInAllCells** property was set to False, iGrid did not apply the hot effect to the combo button after you had changed the current cell.

v7.50, build 0004 | 2021-Mar-01

1. [Fixed] iGrid rows may have disappeared from the viewport after grouping.
2. [Fixed] iGrid may have crashed while doing drag select iGrid had event handlers for the selection change events (such as **AfterSelectionChange**).
3. [Fixed] The **lRow**/**lCol** parameters of the **RowHeightChanged**/**ColWidthChanged** events were set to 0 after resizing the row/column to the height/width of zero.

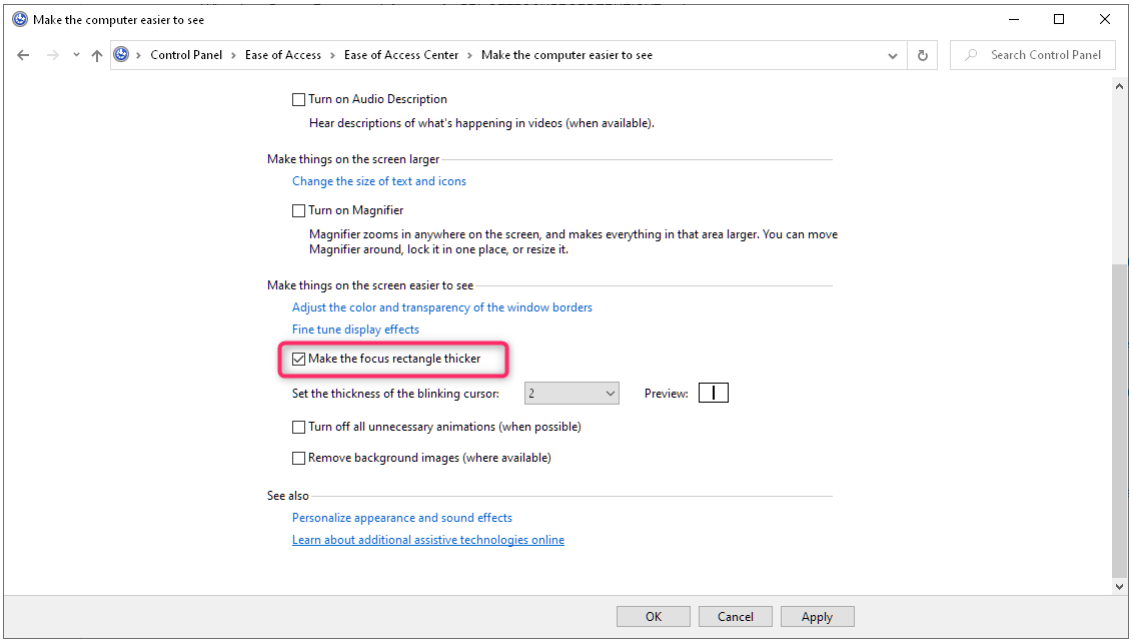
v7.50, release build | 2020-Oct-15

New focus rectangle drawing

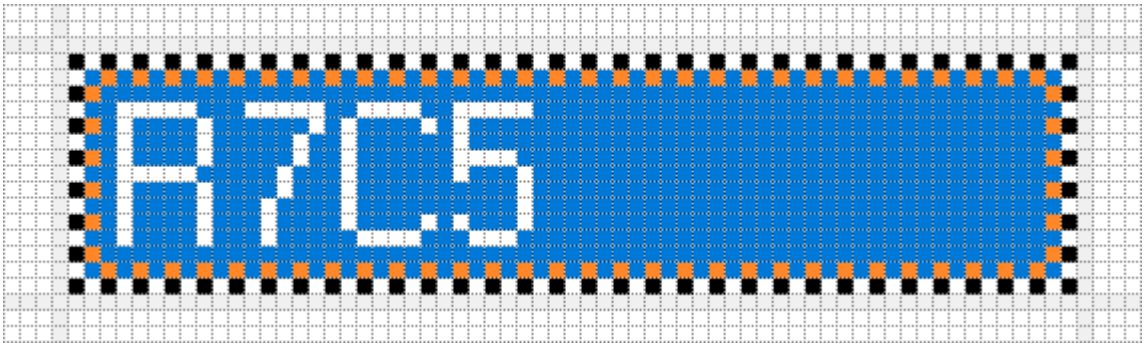
1. [Fixed] Even though the ActiveX platform is losing its popularity among developers, one of the key points of our product support policy is to provide flawless performance of your existing apps with iGrid ActiveX on the computers of your end users. This also concerns cases if something is changed in the Windows API so that iGrid starts functioning improperly.

   The drawing of focus rectangle in iGrids of all previous versions may be incorrect, especially in the latest releases of Windows 10. This issue is related to the **DrawFocusRect** function from the Windows API used for two decades to draw the focus rectangle in iGrid cells. This function may draw the focus rectangle thicker than 1 pixel so it is more visible for high-resolution displays and accessibility needs, and this behavior leads to problems with drawing the focus rectangle in iGrid cells.

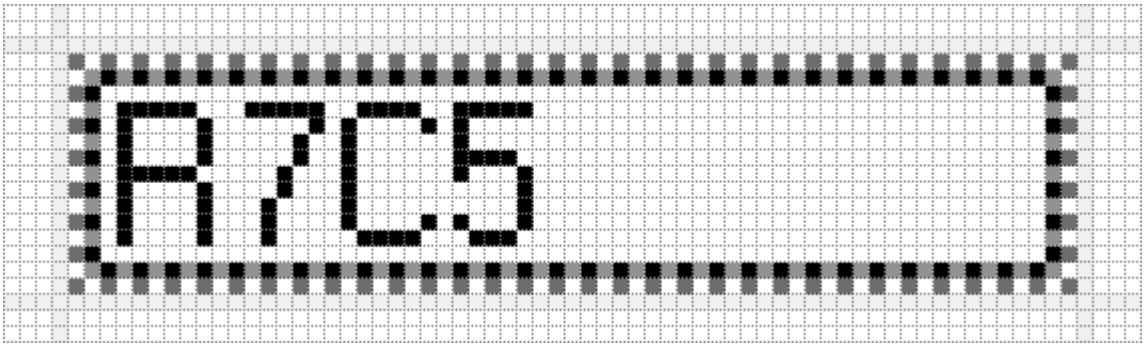
   One of the system options that affects the thickness of the system focus rectangle is 'Make the focus rectangle thicker'. At the time of writing this document, this option can be found in the classic Control Panel when you open 'Ease of Access' > 'Ease of Access Center' > 'Make the computer easier to see':



It turned out that this option can be set on some computers even if the user did not do this explicitly. For example, it can be a preinstalled copy of Windows configured by the computer manufacturer to make the use of the computer equipped with a high-resolution screen easier. Presumably, it can also happen after a Windows upgrade and/or depending on the presence of a high-resolution screen. If this option is set, the **DrawFocusRect** function draws a 2-pixel focus rectangle and blends its pixels with the cell selection color producing dark-orange pixels. The overall picture looks weird and does not correspond to the look expected by the developer and the user, when the current cell must have a 1-pixel focus rectangle of black and white dots:



Another example of incorrect focus rectangle drawing is when the selection is turned off with the **HighlightSelCells** property, the **FocusRectNoFocus** property is set to True and iGrid loses the input focus:



There can be other side effects related to how **DrawFocusRect** draws the focus rectangle in a particular situation. To provide the expected look of the focus rectangle consisting of alternating dots of the two colors specified in the **FocusRectColor1** and **FocusRectColor2** properties, the focus rectangle drawing code was rewritten in this release of iGrid. Now this part of code no longer relies on the **DrawFocusRect** function and implements pixel-by-pixel drawing of the focus rectangle to avoid any issues.

2. [New] All previous versions of iGrid draw the focus rectangle with the thickness of 1 pixel, and there is no way to change this. This release of iGrid implements the new **FocusRectThickness** property that can be used to adjust the thickness of the focus rectangle. This property of the Long data type specifies the thickness of the focus rectangle in pixels explicitly or can be used to set automatic retrieval mode of this value from the OS.

   If this property is set to a value greater than 0, this value is considered the value of the focus rectangle thickness set explicitly. If the value of the property is set to 0, iGrid retrieves the thickness from the OS.

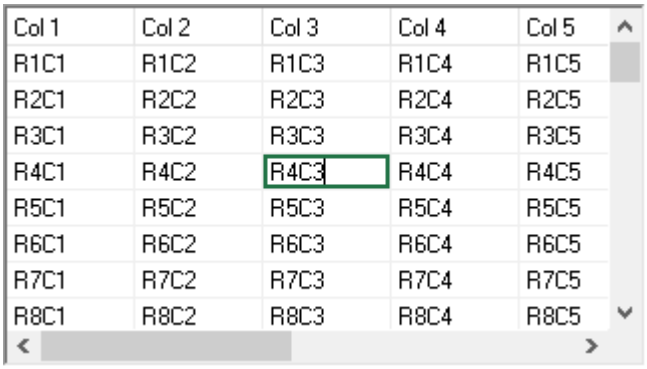
   The retrieval of the system thickness is done at the moment when you set **FocusRectThickness** to 0. The retrieved thickness is used by iGrid until you assign a new value to the property. The OS does not provide a notification sent in all cases when the system thickness of the focus rectangle is changed, and resetting this property to 0 is the only way to update the system focus rectangle thickness used by iGrid.

   Pay attention to the fact that the zero value of this property does not turn the drawing of focus rectangle off. It can be done with the **FocusRect** property.

   You can retrieve the effective thickness of the focus rectangle including the case when the system thickness is used with the help of the existing **Sys()** function. To do this, request the value of the **igSysFocusRectEffectiveThickness** parameter added to the **ESystemParameters** enumeration in this release.

   The default value of the **FocusRectThickness** property is 1, which corresponds to the hard-coded thickness of the focus rectangle in the previous versions of iGrid.

3. [New] The new Boolean **FocusRectInEditMode** property can be used to show the focus rectangle while the user is editing a cell. Using it, you can imitate Excel-style cell editing, when the edited cell is framed by a border helping the user to find the edited cell visually:



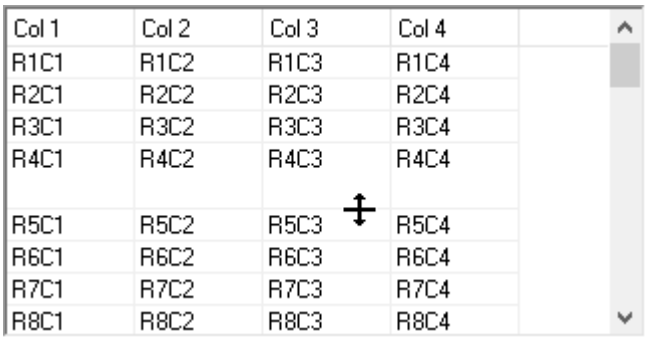
   The effect on the picture above was implemented with the following code:

```
With iGrid1
   .HighlightSelCells = False

   .FocusRectThickness = 2
   .FocusRectInEditMode = True

   .FocusRectColor1 = RGB(33, 115, 70)
   .FocusRectColor2 = RGB(33, 115, 70)
End With
```

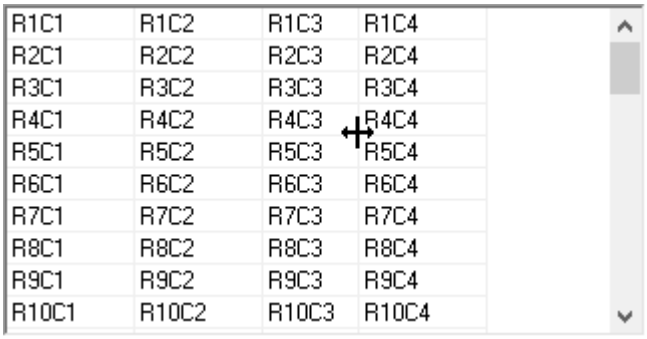
The default value of the **FocusRectInEditMode** property is False.

Resizing rows and columns by dragging grid lines

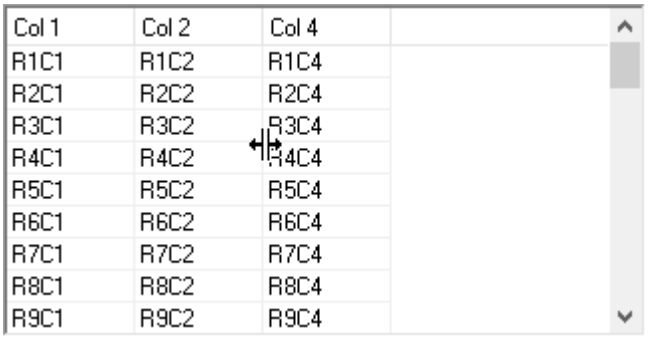
1. [New] This release of iGrid implements the ability to resize columns and rows by dragging grid lines. If this feature is enabled, the cursor changes to the corresponding V-Split or H-Split image when the mouse pointer is in the resize area near a grid line. The user can hold down the left mouse button to drag the grid line to resize the corresponding object. This means that now users can resize not only columns but rows too:



Columns can be resized the same way. This may help users to easier resize columns as they do not need to move the mouse pointer to the small header area at the top. And this ability is indispensable if you want to provide your users with the ability to resize columns when the grid header is invisible:



Pay attention to the fact that if a column has zero width, there is a difference whether the mouse pointer is at the left or right of this column. If the mouse pointer is at the left, the visible column prior to the column with zero width will be resized. If the mouse pointer at the right, the width of the column with zero width will be changed. The latter situation is indicated with the special form of the V-Split cursor with a gap inside (the same also works for rows):



The ability to resize columns and rows using their grid lines can be enabled separately for columns and rows with the new **GridLineDrag** property of the new **EGridLineDrag** enumeration type:

```
Public Enum EGridLineDrag
   igGridLineDragNone = 0
   igGridLineDragVertical = 1
   igGridLineDragHorizontal = 2
   igGridLineDragBoth = 3
End Enum
```

The default value for the **GridLineDrag** property is **igGridLineDragNone**.

2. [New] When the mouse pointer is in the resize area, a double-click leads to auto-sizing of the corresponding object. The **eAction** parameter of the **DblClick** event of iGrid informs you about this operation with the two new values of the **EDblClickAction** enumeration, **igDblClickActionAutoWidthCol** and **igDblClickActionAutoHeightRow**. Note that **eAction** is passed by reference, which gives you the ability to cancel the auto-sizing operation for particular columns/rows depending on some conditions evaluated in an event handler of iGrid's **DblClick** event.

3. [New] Column resizing with grid line can be disabled for individual columns using the existing Boolean **ColAllowSizing** property. To disable resizing of individual rows with grid line, the new Boolean **RowAllowSizing** property was implemented. The **AddRow** method was also supplemented with a new Boolean parameter named **bAllowSizing** to specify this status for new rows created with this method.

4. [New] Column width constraints defined in the **ColMinWidth** and **ColMaxWidth** properties are in effect when the user is resizing columns by dragging their grid lines. However, similar constraints for rows are defined with the new **RequestRowMinMaxHeight** event. The event-based approach was chosen to save memory: properties would consume memory in any case, even if they are not used (which is true for most grids), and memory loss could be large for grids with 100'000+ rows.

   The **RequestRowMinMaxHeight** event has the following signature:

   ```
   Event RequestRowMinMaxHeight(ByVal lRow As Long, _
     ByRef lMinHeight As Long, ByRef lMaxHeight As Long)
   ```

   This event is raised every time when the user is trying to resize a row interactively by dragging its grid line or double-clicking it. The **lMinHeight** and **lMaxHeight** parameters passed by reference are used to specify the minimum and maximum height of the row. The default value for both parameters is -1, which means that the corresponding limit is not specified.

5. [New] iGrid raises the **ColWidthStartChange**, **ColWidthChanging**, and **ColWidthChanged** events while the user is resizing a column by dragging a vertical grid line as if it were done with the column divider in the header area. A similar set of events was introduced to notify about interactive row height change:

```
Event RowHeightStartChange(ByVal lRow As Long, ByVal lHeight As Long)
Event RowHeightChanging(ByVal lRow As Long, ByVal lHeight As Long)
Event RowHeightChanged(ByVal lRow As Long, ByVal lHeight As Long)
```

6. [New] To specify the maximal distance from the vertical grid line within which the mouse pointer turns into the V-Split cursor indicating that column resizing is possible, use the new **GridLineDragDX** property. The same distance for horizontal grid lines is specified with the new **GridLineDragDY** property. The default values for these properties are 7 and 4 pixels respectively.

Other new features of iGrid

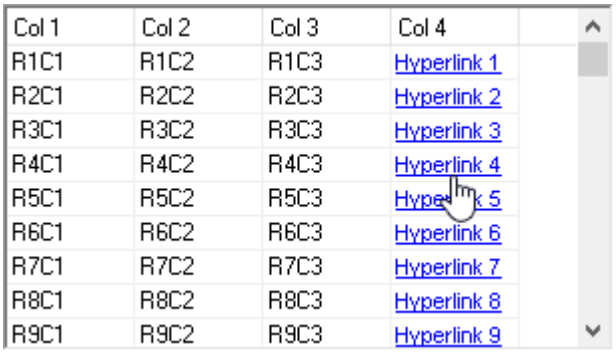
1. [New] This release of iGrid introduces two new properties to specify the look of the mouse pointer inside the control. These are **MousePointer** and **MouseIcon** that work the same way as their counterparts in Visual Basic 6 or Microsoft Office VBA. But iGrid brings some improvements to the existing functionality of the **MousePointer** property compared to VB6/VBA.

   The base type for iGrid's **MousePointer** property is the new **EMousePointer** enumeration:

```
Public Enum EMousePointer
   igMousePointerDefault = 0
   igMousePointerArrow = 1
   igMousePointerCrosshair = 2
   igMousePointerIbeam = 3
   igMousePointerSizeNESW = 6
   igMousePointerSizeNS = 7
   igMousePointerSizeNWSE = 8
   igMousePointerSizeWE = 9
   igMousePointerUpArrow = 10
   igMousePointerHourglass = 11
   igMousePointerNoDrop = 12
   igMousePointerArrowHourglass = 13
   igMousePointerArrowQuestion = 14
   igMousePointerSizeAll = 15
   igMousePointerHand = 98
   igMousePointerCustom = 99
End Enum
```

   These constants provide access to all standard cursors currently available in the Windows OS. All constants except **igMousePointerHand** (98) are compatible with the equivalent constants from the **MousePointerConstants** enumeration in Visual Basic 6. The **vbIconPointer** constant (4) from **MousePointerConstants** is no longer supported by Windows, and it was not included into **EMousePointer** for this reason. The **vbSizePointer** constant (5) is now obsolete and works like **vbSizeAll** (15), and **EMousePointer** provides only **igMousePointerSizeAll** because of this.

   The **igMousePointerHand** constant (98) allows you to use the native OS hand cursor in your apps. Generally is it used to indicate clickable hyperlinks. Now you can also implement hyperlink cells in iGrid easily due to this item in the **EMousePointer** enumeration. An example is on the picture below:



The hyperlink hover effect with the hand cursor was implemented using the following two event handlers:

```
Private Sub iGrid1_MouseEnter(ByVal lRow As Long, ByVal lCol As Long)
   If lCol = 4 Then
      iGrid1.MousePointer = igMousePointerHand
   End If
End Sub

Private Sub iGrid1_MouseLeave(ByVal lRow As Long, ByVal lCol As Long)
   If lCol = 4 Then
      iGrid1.MousePointer = igMousePointerDefault
   End If
End Sub
```

iGrid also improves the functionality of the **MousePointer** property in the scroll bar area compared to VB6/VBA. If you are using a cursor different from the default one, it is automatically turned into the standard arrow cursor in the scroll bar area in iGrid to indicate a clickable area. Users find this behavior more natural than what they see in VB6/VBA when the cursor remains unchanged (non-default) in the scroll bars. The same effect is used for the header area and the space occupied by the iGrid border.

2. [New] iGrid implements the new **PopupMenu** method to display one of its built-in context menus:

```
Sub PopupMenu( _
   ByVal eContextMenu As EContextMenuSource, _
   Optional ByVal x As Variant, Optional ByVal y As Variant, _
   Optional ByVal vRow As Variant, Optional ByVal vCol As Variant)
```

The method parameters are the followings:

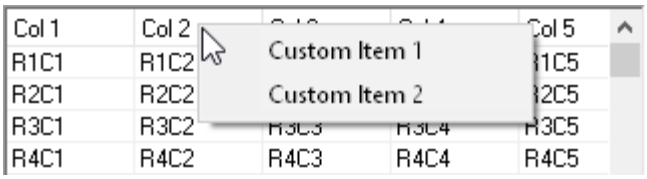
- The **eContextMenu** parameter specifies the context menu to display.
- The **x** and **y** parameters are the coordinates of the top-left corner of the context menu in the grid coordinates. They are optional. If the value for **x**/**y** is omitted, the corresponding coordinate of the mouse pointer is used.
- The **vRow** and **vCol** parameters are used to specify an existing cell or column header if this makes sense for the context menu specified in the **eContextMenu** parameter. If **eContextMenu** equals **igContextMenuCell** or **igContextMenuTextEdit**, you must specify an existing row and column in these parameters. If **eContextMenu** equals **igContextMenuColHeader,** only an existing column must be specified in the **vCol** parameter. For other kinds of menus, **vRow**/**vCol** can be omitted or any value can be specified.

Below is an example how you can use this method to display the built-in column header context menu when the user clicks a column header with the left mouse button:

```
Private Sub iGrid1_ColHeaderClick(ByVal lCol As Long, bDoDefault As Boolean, _
      ByVal Shift As Integer, ByVal x As Long, ByVal y As Long)

   bDoDefault = False
   iGrid1.PopupMenu igContextMenuColHeader, x, y, 0, lCol
End Sub
```

Note: if the value of the **eContextMenu** parameter is **igContextMenuTextEdit**, no context menu is displayed because the corresponding context menu is generated by the system for text edit controls (unless you specified custom items for this kind of menu).

3. [New] The **EContextMenuSource** enumeration contains one new item named **igContextMenuCustom**. It can be used to define your own custom popup menu for iGrid. This menu can be displayed with the new **PopupMenu** method introduced in this release of iGrid. Below is an example of a custom context menu displayed when the user clicks a column header:



This custom context menu was created with the following code:

```
iGrid1.ContextMenuCustomItems(igContextMenuCustom).Add "Custom Item 1"
iGrid1.ContextMenuCustomItems(igContextMenuCustom).Add "Custom Item 2"
```

It is displayed when the user clicks a column header using the following event handler:

```
Private Sub iGrid1_ColHeaderClick(ByVal lCol As Long, bDoDefault As Boolean, _
      ByVal Shift As Integer, ByVal x As Long, ByVal y As Long)
   bDoDefault = False
   iGrid1.PopupMenu igContextMenuCustom, x, y, 0, lCol
End Sub
```

4. [New] The **Header** object property provides you with two new properties to adjust the indent at the left and right of the sort info in column headers. These are **SortInfoLeftIndent** and **SortInfoRightIndent** respectively. Using them, you can make column header contents more compact by decreasing the distance between sort info and column header text or add some space at the right of sort info if your users find the sort icon located very close to the right edge of a column header. The default values of these properties are 3 and 0 pixels respectively, which corresponds to the hard-coded parameters in the previous versions of iGrid.

5. [New] iGrid implements the read-only **Version** property you can access to know the version of the OCX. The version is returned in the following format: <major version>.<minor version>.<build number>. The minor version part occupies two digits and is aligned to the left, the build number occupies 4 digits and is aligned to the right; non-essential digits are filled with "0". For example, for the iGrid of the version 7.5 and build 24, the property will return "7.50.0024".

6. [New] The new Boolean **SelectionAlphaBlendCellText** property controls whether cell text is blended with semi-transparent selection when it is on. The default value is True, which corresponds the behavior of iGrid in the previous versions.

7. [New] The **CellObject** returned by the **ColDefaultCell** object property now has the **lExtraData** property to specify default value for the **CellExtraData** property for new cells in the column.

8. [Enhancement][Code-Upgrade] If the **SortObject** was empty (for example, after calling its **Clear** method), the **Sort** method of iGrid did nothing in the previous versions. Now it sets all column sort keys to zero, i.e. clears sort info in every column header on the screen.
9. [Enhancement] [Code-Upgrade] When you assign a column index to the **SortCol** property of **SortObject** or **GroupObject**, iGrid checks whether the specified column index is valid and does not allow you to specify a non-existing column, zero or negative values. The corresponding iGrid error is generated in this case, and this helps to detect potential problems in the developer's code.

Fixed bugs

1. [Fixed] The **lRow** parameter of the **ContextMenuPopup** event contained an incorrect value if the cell context menu was activated from the keyboard.

v7.00, build 0017 | 2019-Sep-06

1. [Fixed] The 'Type mismatch' error occurred while copying/pasting strings into combo box cells.
2. [Fixed] The **EndUpdate** method failed after removing rows with the **RemoveRow** method.

v7.00, build 0015 | 2019-Mar-15

1. [Enhancement] Now iGrid does not remove selection from a cell if the row or column it belongs to becomes invisible. This feature allows you to implement filters and save the selection status for cells or whole rows in row mode when a row becomes visible/invisible during filtering. To provide the behavior expected by the user, the DELETE key and the CTRL+X/CTRL+C keyboard combinations do not process selected invisible cells.
2. [Enhancement] This version of iGrid guarantees that the **ClickNoDblClick** event is raised after the **MouseUp** and **Click** events.
3. [Fixed] Several problems related to the copy-paste functionality and the DELETE key were fixed (when iGrid had invisible rows/columns, when the row text column was visible, etc.).
4. [Fixed] If row mode was on and the visibility of a column changed, iGrid may have updated the selection status of the cells from this column in selected rows improperly.
5. [Fixed] If row mode and multi-select mode were on, iGrid did not remove selection from a group row when you cleared the **CellSelected** property for the row text cell of this row:

```
iGrid1.CellSelected(1, iGrid1.RowTextCol) = False
```

6. [Fixed] When iGrid had the input focus, the Escape key was processed entirely by iGrid and was not passed to the form's key handling events. As a result, the logic for the Escape key implemented in the form's **KeyDown** event or the form's button set as the cancel button did not work. In this build of iGrid the Escape key reaches the form's key handling events in all situations unless it does something in iGrid like canceling user input in a text cell or clearing the search string if incremental search is on.
7. [Fixed] The **Group** method failed if you calculated the SUM() or AVG() aggregate functions for non-numeric cell values.
8. [Fixed] Pausing on a break point inside an event handler of the **RequestCellToolTip** event crashed the development environment.
9. [Fixed] The **LayoutCol** property did not recalculate column widths properly for non-standard dpi settings.

10. [Fixed] iGrid may have drawn its contents incorrectly when the **DisplayMode** property of its scroll bars was set to **igScrollBarDisplayNever** or **igScrollBarDisplayAlways**.
11. [Fixed] The **Click** event was raised even if the mouse pointer was outside of the cell in which the mouse button was pressed, and the coordinates of the cell under the mouse pointer in which the button was released were passed in the event parameters. Now the **Click** event is raised only if the mouse pointer remains within the bounds of the cell in which the mouse button was pressed.
12. [Fixed] iGrid may have worked improperly if the input focus was moved to another window after pressing the mouse button in iGrid but before releasing it. For example, if multi-select mode was on and the user pressed the mouse button holding down the SHIFT key to select a range of cells, and then a message box was displayed from an event handler of the **CurRowChange** event, iGrid continued to select a range of cells after closing the message box as if the mouse button was still pressed. Another example of such a situation is when Windows brings another application to foreground.