# 10Tec iGrid.NET 10.1
# What's New in Control

## v10.1.0 | 2022-Mar-14

### Changes in the iGrid DLL file names and deployment system

The file names of the DLLs of all previous versions of iGrid include the major and minor version numbers of the product, for example: TenTec.Windows.iGridLib.iGrid.v10.0.dll. Many developers want to keep different builds of iGrid in separate files. To provide this ability, all iGrid DLL file names will include the build number in addition to the major/minor version numbers starting from this release.

Starting from iGrid.NET 10.0 two separate DLL sets are provided for the traditional .NET Framework 4.x and the newest .NET Core Framework (.NET 5+). To distinguish between them, the iGrid DLL file names now will also include the suffix corresponding to the framework they were built for (so-called target framework moniker). For example, these are 'net4' for the .NET Framework 4.x and 'net6' for .NET 6.

The DLL file names are simplified to shorten their names containing the full version number and target framework moniker now. Every DLL file name now starts with the root namespace of the classes it implements:

- TenTec.Windows.iGridLib – the core iGrid control functionality
- TenTec.Windows.iGridLib.Design - the design-time functionality of iGrid and its add-ons
- TenTec.Windows.iGridLib.Filtering - the AutoFilterManager add-on
- TenTec.Windows.iGridLib.Printing - the PrintManager add-on

The small letter 'v' before the full version number was removed for brevity, clarity, and according to the standards Microsoft uses in its .NET assembly names.

Putting it all together, you can look at an example of the new DLL file names for this release of iGrid for the .NET Framework 4.x:

- TenTec.Windows.iGridLib.10.1.0.net4.dll
- TenTec.Windows.iGridLib.Design.10.1.0.net4.dll
- TenTec.Windows.iGridLib.Filtering.10.1.0.net4.dll
- TenTec.Windows.iGridLib.Printing.10.1.0.net4.dll

This new DLL file name system will work together with the updated demo installers. Now you can install the demo and all accompanying files for different builds of the same major/minor version of iGrid simultaneously. The folder structure for the installations is the following:

C:\10Tec\iGrid.NET\<major>.<minor>.<build>

The iGrid DLLs will be stored in the Bin subfolder and will be grouped by target frameworks in the corresponding subfolders, for example:

C:\10Tec\iGrid.NET\10.1.0\Bin\net4 - for .NET Framework 4.x

C:\10Tec\iGrid.NET\10.1.0\Bin\net6 - for .NET 6

To provide support for design time features in WinForms projects targeting .NET 6, the NuGet packages with iGrid DLLs will be also installed locally by the demo installer into the following default location:

C:\10Tec\NuGetPackages\

## .NET 6 and Visual Studio 2022 compatibility

This release of iGrid was successfully tested with the latest version of the .NET Core Framework on the moment of release, which is .NET 6. iGrid is now provided in a separate set of DLLs compiled specially for .NET 6 in addition to similar DLL sets compiled for .NET Framework 4 and .NET 5. The control was also successfully tested in the latest version of Visual Studio on the moment of release, which is 2022.

iGrid has a new form of redistribution, a NuGet package, which can be used as an alternative to the traditional exe installers deploying the control DLLs on a hard drive in a specific folder. The NuGet approach is the only way recommended by Microsoft if you want to enable all design time features of a control in WinForms projects targeting .NET Core - such as control icons on the toolbox, the ability to drop a control onto a form surface in the designer and change its properties in the property grid, etc.

To provide this design-time functionality, iGrid's design-time DLL for .NET 6 was revisited and recompiled using the newest WinForms Designer Extensibility SDK developed by Microsoft for this purpose. To find out more about the challenges Microsoft faced during the implementation of this SDK, read this blog post:

State of the Windows Forms Designer for .NET Applications - .NET Blog (microsoft.com)

The work on this SDK had not been finished on the moment of this iGrid release. As a result, some design time features are not available in this release of iGrid for .NET 6. These are:

- Creation and change of columns, rows, and cells. The 'Edit Columns' and 'Edit Rows and Cells' designer verbs were hidden because of that. The **Cols** and **Rows** properties that provide the same collection editors via the ellipsis buttons in the property grid were also hidden for design time.
- The 'Auto Fit', 'Clear', 'Reset', and 'About' designer verbs are not available because they display additional dialogs not fully supported in the WinForms Designer SDK.
- The verb 'Edit Items' for the **iGDropDownList** component. Currently you cannot open the item collection editor to create/edit items using the component's verb, but you can open this editor using the ellipsis button in the field containing the value of the Items property in the property grid.
- The collection editor for the **iGDropDownList.Items** property does not automatically generate item values ("Value1", "Value2", "Value3", …) for the created items.
- The drop-down list showing possible values for the **iGDropDownListItem.ImageIndex** property displays only numeric image indices without corresponding images.
- All properties based on the **iGContentAlignment** enumeration (such, as **iGCell.TextAlign** and **iGCell.ImageAlign**) display possible value list as a plain text list instead of a visual editor.
- The **iGrid.GroupRowLevelStyles** and **iGrid.ScrollBarSettings.CustomButtons** properties. They are hidden in the property grid because Microsoft did not fully implement design-time collection editors.
- The **iGrid.ScrollBarSettings.CustomButtons** property does not have a visual editor based on the **TrackBar** control. You can specify the level of opacity only by typing a numeric percentage value.
- The **iGrid.RowTectCol.CellStyle** property is not initialized with a new **iGCellStyle** object.

We are continuing to monitor the changes in the WinForms Designer Extensibility SDK and will make the corresponding iGrid design time features available as soon as it is possible.

## Windows 11 support and hover effect for scroll bars

1. [Enhancement] This version of iGrid supports the hover effect for its scroll bars. This means that the iGrid scroll bars are redrawn using the special hover state when the mouse pointer is placed over them. The support for this effect is especially important in Windows 11 in which the scroll bar arrow buttons are not visible at all when the mouse pointer is outside of the scroll bar area. Compare the look of the horizontal scroll bar in iGrid 10.0 and iGrid 10.1 when the mouse pointer hovers over it in Windows 11:

   iGrid.NET 10.0: iGrid.NET 10.1:

   Windows 7 is still in the range of the OS versions supported in iGrid. The implemented hover effect is also noticeable in this OS and makes the scroll bar look similar with the native Windows 7 scroll bars:

   iGrid.NET 10.0: iGrid.NET 10.1:

2. [Enhancement] If iGrid scroll bars are rendered using the OS visual styles and the scroll box contains a gripper, the gripper is drawn using the special hot style when the mouse pointer is over the scroll box. Below is an example for Windows 7:

   iGrid.NET 10.0: iGrid.NET 10.1:

3. [Enhancement] If the OS styles are turned off for the iGrid scroll bars and their appearance is set to 'flat', the scroll box uses the same press effect as the scroll bar arrow buttons. In the previous builds it simply remained highlighted in the pressed state:

   iGrid.NET 10.0: iGrid.NET 10.1:

4. [Enhancement][Code-Upgrade] To support the hover effect in the main scroll bar elements and custom scroll bar buttons, the Boolean **hoverEffect** parameter was added to the **DrawScrollBar()** and **DrawScrollBarCustomButton()** methods of the **IiGControlPaint** interface:

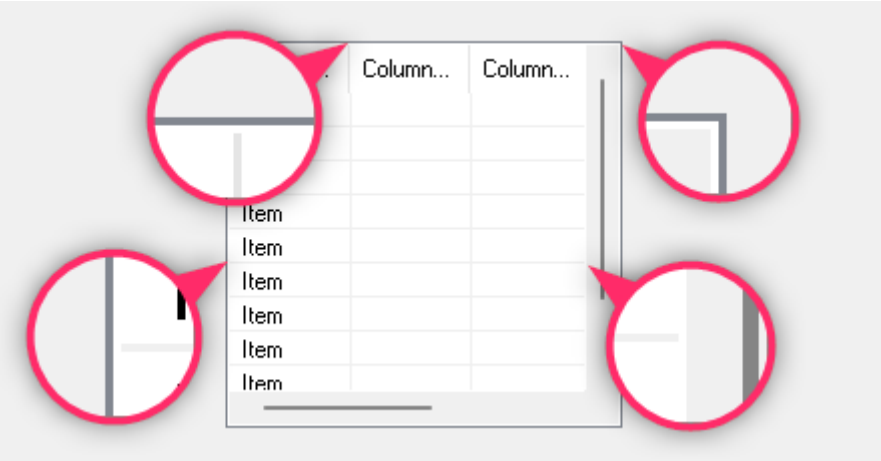
   ```
   void DrawScrollBar(Graphics g, int x, int y, int width, int height,
   iGScrollBarPart scrollPart, iGControlState controlState, bool hoverEffect);

   void DrawScrollBarCustomButton(Graphics g, int x, int y, int width, int
   height, iGControlState controlState, bool hoverEffect);
   ```

   This parameter is set to True when the mouse pointer is in the area occupied by the scroll bar.

   To upgrade your code from the previous versions, it is enough to add the Boolean **hoverEffect** parameter to the signatures of the implemented methods. If required, you can now also implement the hover effect for your custom drawn scroll bars based on the value of the **hoverEffect** parameter.

## New system border with OS style support

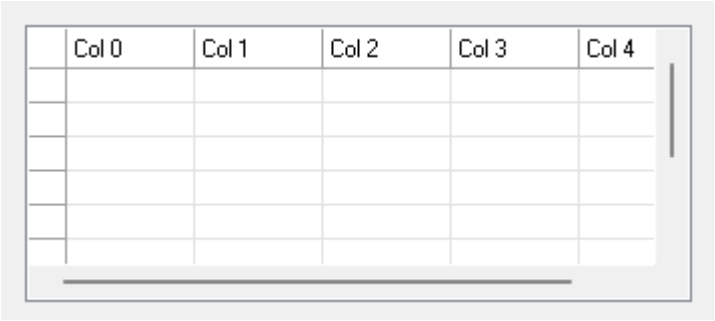
1. [Enhancement][Change] If you look carefully at the default border of standard list controls like ListBox or ListView in a Windows app rendered with the OS visual styles, you will notice that the border consists of 2 lines. The outer line is a 1-pixel dark line, and the inner line is the white 1-pixel line:




Note that the same 1-pixel white line is drawn at the left of the vertical scroll bar and at the top of the horizontal scroll bar, which creates a beautiful symmetric effect of empty space around the scroll bars.

This is the way of how the border of the default **Fixed3D** style is rendered using the OS visual styles (the default value for the **BorderStyle** property in the WinForms controls like ListBox or ListView). Pay attention to the fact that the border of this style is rendered using the 3D effect from earlier versions of Windows like Windows 95 if visual styles are not available.

iGrid provides a set of border styles similar to what we have in WinForms controls. These are **Standard**, **System**, **Flat**, and **None** you can specify with iGrid's **BorderStyle** property. The default value is **System**, which corresponds to the **Fixed3D** setting in the WinForms list controls. In the earlier versions of iGrid the default **System** border was rendered using the 3D effect regardless of the availability of visual styles in the app. Starting from this release of iGrid, the default **System** border is rendered using the visual styles if they are available:

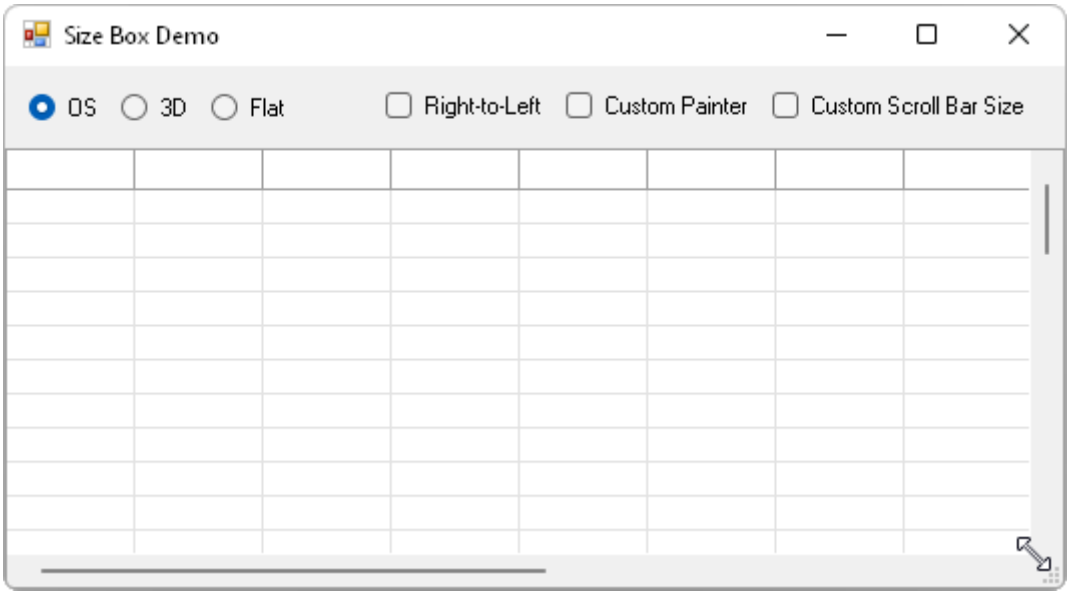



This provides the same visual effect for all iGrid elements out-of-the-box as in the WinForms ListBox and ListView controls with the default settings. If you used the default **System** border in your apps with iGrid, no need to change your code to get this system border look. If you need the old 3D effect for the border regardless of the availability of visual styles, specify the **Standard** value in the **BorderStyle** property of iGrid.

2. [New] iGrid implements the **BorderWidth** property that allows you to set/retrieve the width of the flat border. But this property can't help you to know the width of border of other types (for example, for the border of the **System** style). The new **GetEffectiveBorderWith()** function implemented in this release of iGrid fills this gap. Now you can retrieve the current border size in pixels using it.

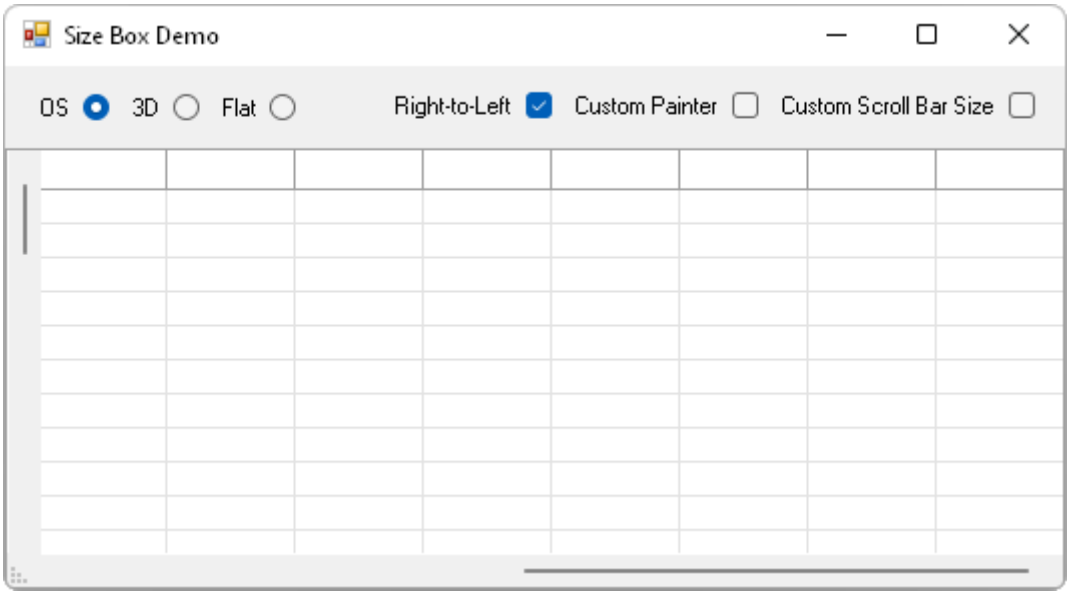
## Size box and size grip enhancements

When both scroll bars are visible, iGrid draws a special rectangular area called 'size box' below the vertical scroll bar and at the right of the horizontal scroll bar (at the left in right-to-left mode). If the bottom-right corner of iGrid is placed in the bottom right corner of the containing form and the form is sizeable, the size grip automatically appears inside the size box indicating that the form can be resized by dragging any point inside the size box area:




This release of iGrid implements the following enhancements related to the size box area.

1. [Enhancement] The appearance of the size grip no longer depends on the appearance of the iGrid scroll bars (OS style, 3D, or flat look). Even if the scroll bars are rendered using the flat or 3D style, the size grip is rendered using the OS visual style if the form supports it. This corresponds to user expectations when they must see the same size grip in all forms regardless the fact whether iGrid is placed on it or not.
2. [Enhancement] The size box with the size grip now works in right-to-left mode:




3. [Enhancement] The size grip now supports high-resolution displays.

4. [Renaming][Code-Upgrade] iGrid provides you with the **IsPointOverSizeBox()** function that allows you to test whether a point belongs to the size box area. This function has the following declaration in the previous versions of iGrid:

```
bool IsPointOverSizeBox(int x, int y, bool checkDrawSizeBox)
```

The name of the **checkDrawSizeBox** parameter may have misled the developer because actually the visibility of the size grip was checked when the whole size box is visible. This parameter was renamed to **checkSizeGripVisible** in this release:

```
bool IsPointOverSizeBox(int x, int y, bool checkSizeGripVisible)
```

No changes in your code are required unless you use named parameters to invoke this function.

5. [Change][Code-Upgrade] You can redefine the size box drawing if you implement the **DrawSizeBox()** method of the **IiGControlPaint** interface in a custom painter class. The previous versions of iGrid called this method only when the size grip was drawn. The size box area was also filled automatically by iGrid with the corresponding color before calling this method. All this did not allow you to draw the size box area by yourself when the size grip was not visible, and you could not make the size box area completely transparent (unfilled) if required.

The new implementation of the size box drawing in this release is free from these drawbacks. Now you have full control over the size box drawing. The **IiGControlPaint.DrawSizeBox()** method is called every time when iGrid draws the size box area. To know whether the size grip must be drawn inside the custom size box, the **DrawSizeBox()** method was supplemented by the new Boolean **sizeGripVisible** parameter set to True when the size grip must be drawn:

```
void DrawSizeBox(
    Graphics g, int x, int y, int width, int height,
    iGSizeBoxAlign align, bool sizeGripVisible);
```

If you already have code drawing a custom size grip in the **IiGControlPaint.DrawSizeBox()** method, it will be easy to upgrade it to work in the current release. First, add the Boolean **sizeGripVisible** parameter to the method signature. Then add code to fill the size box area and draw the size grip using the existing code like in the following example:

```
public static void DrawSizeBox(
    Graphics g, int x, int y, int width, int height,
    Color backColor, iGSizeBoxAlign align, bool sizeGripVisible)
{
    using (Brush myBrush = new SolidBrush(backColor))
        g.FillRectangle(myBrush, x, y, width, height);

    if (sizeGripVisible)
    {
        // Existing size grip drawing code
        …
    }
}
```

## Enhancements in the header areas

1. [Enhancement][Change] The **BackColor** property of the **Header** and **RowHeader** object properties is set to **SystemColors.Control** by default in the previous versions of iGrid. However, this does not correspond to the actual background color used by default when the column header and row header areas are rendered using the OS visual styles. Moreover, if visual styles are used in these header areas, changes of the **BackColor** property of the **Header**/**RowHeader** object have no effect.

   This version of iGrid provides the developer with the ability to change the background color of the column header and row header areas with the **BackColor** property. Now its default value is **Color.Empty**, which means that the corresponding color is used depending on the look of the header area (**SystemColors.Window** for the OS visual styles, **SystemColors.Control** for the 3D and flat styles). If you assign a color value to this property, the specified color will be used in the corresponding header area – even if its look is set to 'visual styles'.

   This new functionality of the **BackColor** property of the **Header**/**RowHeader** objects is also fully corresponds to the functionality of the existing **HotTrackBackColor** and **PressedItemBackColor** properties of the **Header**/**RowHeader** objects. These properties are set to **Color.Empty** by default, which means that the corresponding color effects are used for the hot and pressed header items. If they are set to a non-default value, the specified color is used for the corresponding effect.

   This change does not require any changes in existing code in the vast majority of cases. If you used the default value of the **BackColor** property of the **Header**/**RowHeader** objects, your code will work as earlier. If you set the **BackColor** property to a non-default value and used the 3D or flat look for the header area by setting the **UseXPStyles** property to False to render the header with the specified color, no changes are required as well. You only need to change your code if you specify a non-default value for **BackColor** and the header is rendered using the OS styles because now this color will be used instead of the standard system color.

2. [Enhancement] The default empty value of the **BackColor**, **HotTrackBackColor**, and **PressedItemBackColor** properties of the **Header** and **RowHeader** objects is now displayed as 'NotSet' in the Visual Studio property grid. An empty string was used in this case in the previous versions of iGrid, which did not correspond to the behavior of other color properties at design time.

3. [Enhancement] Row headers now supports the animated pressed effect like column headers.

## Memory usage and performance optimizations

1. [Optimization][New][Change][Code-Upgrade] iGrid.NET 6 introduced the **PostPaint** event to draw over its contents. The scrolling optimization for typical scenarios of iGrid usage was disabled after that because it could not work together with this ability.

   The **PostPaint** event is not raised in this release unless you enable it with the new Boolean **PostPaintEnabled** property. This made possible the re-enabling of the scrolling optimization when the **PostPaint** event is not used, which is the case in the vast majority of iGrid usage scenarios. As a result, iGrid uses much fewer resources to redraw its contents when the user is scrolling it. The scrolling of an iGrid with a huge number of cells in the viewport or cells with heavy contents is performed faster as well, which is especially noticeable on devices with low performance.

   If you upgrade your code to work with this release of iGrid and your code contains a **PostPaint** event handler, assign True to the **PostPaintEnabled** property to make your event handler work.

2. [Optimization] iGrid uses fewer resources and is initialized faster due to the rewritten OS theme manager used to render iGrid parts with the OS visual styles.

3. [Optimization] iGrid creates a lot less GDI+ objects while rendering column and row headers with the OS visual styles, especially if custom background colors are specified. This leads to rarer garbage collection operations.

## Fixed bugs

1. [Fixed][New] If the **TexFormatFlags** property of a cell or a cell style object contained the **iGStringFormatFlags.DirectionRightToLeft** or **iGStringFormatFlags.DirectionLeftToRight** flag, they were not applied to the cell text editor which simply inherited the grid's right-to-left setting in the previous versions.

   This fix became possible due to the related improvement in the in the **iGCellEditorBase** class. It was supplemented with the new **RightToLeft** virtual property in this release:

   ```
   public virtual System.Windows.Forms.RightToLeft RightToLeft { get; set; }
   ```

   If you have a custom editor based on this class and its right-to-left mode can be adjusted, override this property implementation in your editor to accept the correct right-to-left setting from iGrid.

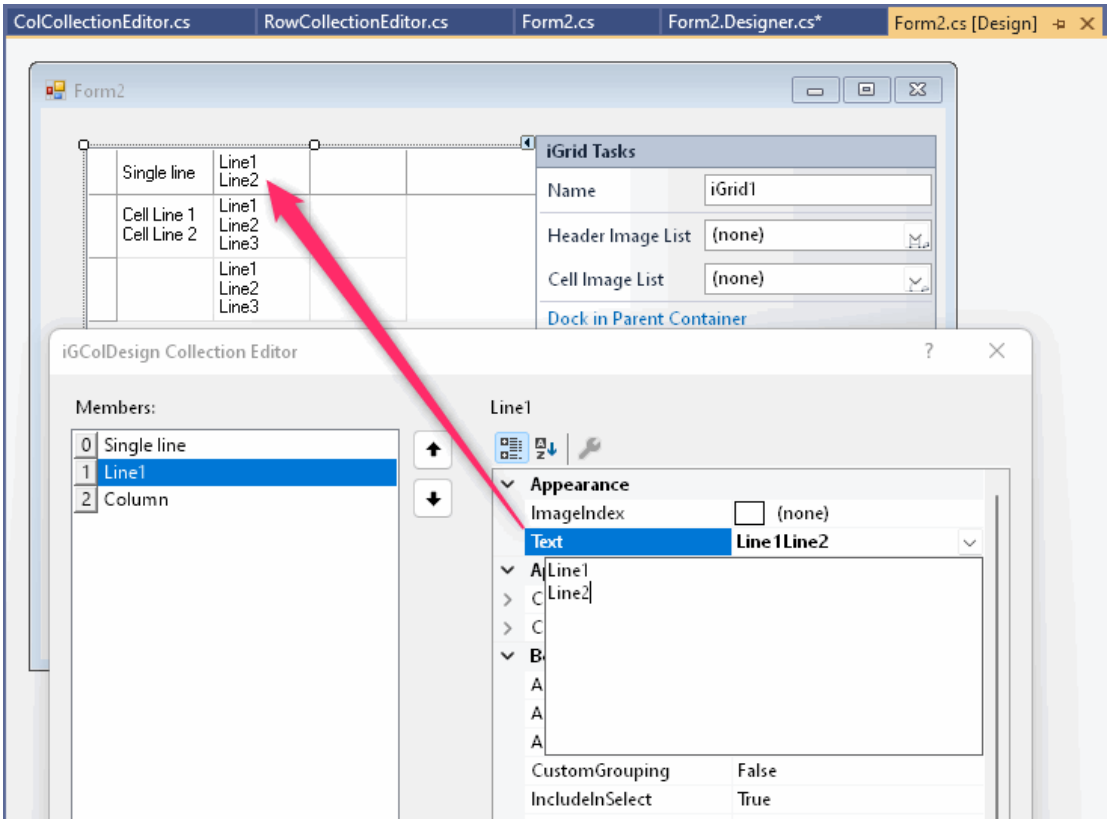
2. [Fixed] The **CopyTo()** method of the **iGrid.SelectedCells** and **iGrid.SelectedRows** collections threw an **ArgumentOutOfRangeException** even if the destination array had enough items.
3. [Fixed] iGrid threw a **NullReferenceException** if it was sorted or grouped by a column with the **SortType** property set to **iGSortType.ByAuxValue** and the **AuxValue** property of a cell in the column was set to null.
4. [Fixed] A custom value assigned to the **Header.ControlsForeColor** property at design time did not work at run time.
5. [Fixed] The **iGDropDownList** class threw a **NullReference** exception if the value one of its items was not initialized (the **iGDropDownListItem.Value** property was null).
6. [Fixed] The **iGrid.Header.ForeColor** property was not reset with the correct default value when the developer chose the Reset verb for iGrid at design time.
7. [Fixed] If the **iGrid.HotTracking** or **iGrid.Header.HotTracking** property was set to False to disable the hot track effect in the header, this effect was not turned off for x-buttons.
8. [Fixed] Dynamic background color settings for row headers made in an event handler of the **RowHdrDynamicBackColor** event did not work.
9. [Fixed] The **CustomAreaBounds** property of the event data object of the **CustomDrawRowHdrForeground** event contained incorrect coordinates in right-to-left mode.
10. [Fixed] The **iGDropDownList.AutoWidth** setting did not work after changing from True to False.
11. [Fixed] The **IsPointOverSizeBox()** method did not work correctly for coordinates outside of iGrid.
12. [Fixed] Merged cells used incorrect row cell style while scrolling the grid.

# v10.1.8 | 2022-May-06

## Changes in iGrid

1. [Enhancement] The ability to enter multiline text at design time were added to the following properties:
   - The **Text** and **DefaultCellValue** properties of the iGrid column object (**iGCol**).
   - The **Value** property of the iGrid cell object (**iGCell**).
   - The **Text** and **Value** properties of the iGrid drop-down list item (**iGDropDownListItem**).
   - The **Text** property of the page band section class in the PrintManager add-on (**iGPageBandSection**). This property is provided by the **LeftSection**, **MiddleSection**, and **RightSection** object properties of the **DocumentHeader**, **DocumentFooter**, **PageHeader**, and **PageFooter** object properties of PrintManager.

This is done with the help of the standard multiline text editor you can find in many properties of WinForms controls, such as the **Text** property of the **TextBox** control:




2. [Enhancement][Change][Code-Upgrade] iGrid cells provide the **EmptyStringAs** property that specifies how an empty string entered by the user should be saved in the cell value (as **null**, **DBNull**, etc.) In the previous versions of iGrid this setting was applied not only to empty strings but also to strings containing of a series of white space characters. This prevented typing a string of multiple spaces into a cell.

   This behavior was changed in this release of iGrid. Now the **EmptyStringAs** setting is applied only to an empty string (i.e. a string containing no characters), and strings consisting of one or more white space characters are saved "as is" in iGrid cells.

   If the logic of your app was based on the old behavior of iGrid, you can use the following event handler to provide the same functionality in this release of iGrid:

```
private void IGrid1_BeforeCommitEdit(
    object sender, iGBeforeCommitEditEventArgs e)
{
    if (string.IsNullOrWhiteSpace(e.NewText))
    {
        e.NewValue = null;
    }
}
```

## Changes in AutoFilterManager

1. The source code of the AutoFilterManager add-on was converted from Visual Basic to C#. AutoFilterManager no longer depends on the Visual Basic runtime services.

2. [Optimization] The performance of the AutoFilterManager add-on was greatly enhanced. The process of attachment to the target grid can be up to 30 times faster compared to the previous versions. This

can help to avoid significant delays when a form with iGrid and AutoFilterManager starts, especially on slow computers. The more columns iGrid has, the more noticeable the effect.

The resource usage was also optimized. Now the add-on uses significantly less graphical resources, which is especially noticeable when iGrid has many columns.

3. [Enhancement] The previous versions of AutoFilterManager showed empty cell texts containing one or more spaces as separate filter items in the filter box. Starting from this release such cell texts are considered blank items and can be filtered using the same special "(Blank)" filter item.

   AutoFilterManager also removes leading and trailing spaces from cell texts to show them as one filter item in the filter box. For example, the cell texts "Apricot" and " Apricot" (one leading space) will be represented as one "Apricot" filter item.

   These two changes provide user-expected behavior that is also implemented in Microsoft Office applications, which are considered the industry standard.

4. [Enhancement] The previous versions of AutoFilterManager didn't allow you to enter a series of white spaces as parameters of custom filter conditions ("Equals", "Contains" and so on). This became possible in this release. Now you can find cells containing one or more space characters or starting with them using AutoFilterManager.

5. [Fixed] The filter box item list could show all items unchecked after adding custom conditions without parameter value and applying such a filter. Actually all items must be checked because no filter is applied, and it works as expected in this release.

6. [Fixed] The filter box item list could show "(absent)" instead of "(Blank) (absent)".

## Addendum: Tags used to classify changes

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