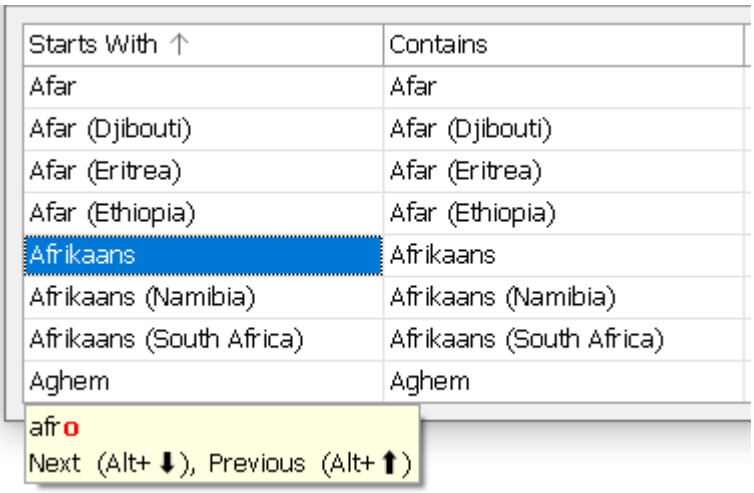
10Tec iGrid for .NET X
What's New in Latest Builds

v10.0.30 | 2021-Jul-05

1. [Enhancement] The window displayed during search-as-type operations was modified. First, the bitmap indicating that the last typed symbol leads to no matches (an exclamation in a red circle) is no longer displayed after the last typed symbol – only the last typed symbol is displayed in red in this case. The bitmaps for the Arrow Down and Arrow Up icons were replaced with the equivalent Unicode characters depicting the corresponding arrows:



   The updated window better corresponds to the visual enhancements of iGrid.NET X. The changes also make this window looking better on high-resolution screens as the bitmaps in the previous implementations were not sized according to the effective dpi value.

2. [Fixed][Change] The **iGRow.Parent** property ignores the visibility status of rows while searching for the parent row, which allows you to retrieve row parents properly even if group rows or tree nodes are not currently visible when their parent rows are collapsed.

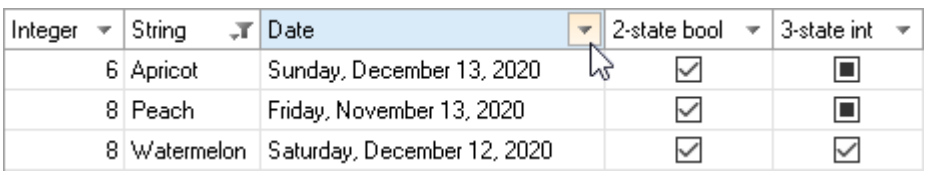
3. [Fixed] PrintManager threw an exception when printing a grid with merged cells from the print-preview dialog.

4. [Fixed] PrintManager threw an exception if you tried to print different grids with the same PrintManager instance.

v10.0.24 | 2021-Feb-12

AutoFilterManager add-on improvements

1. [Enhancement] iGrid.NET X brought a lightweight look compared to its previous versions. One of the related UI changes is a new look of column header combo buttons, which are now drawn with borders only in the hot state. To correspond to this concept, the filter buttons added by AutoFilterManager are also drawn with borders only in the hot state:



2. [Enhancement] The AutoFilterManager add-on is represented with instances of the **iGAutoFilterManager** class implementing the **IDisposable** interface and its **Dispose()** method to free the used resources. However, not all developers call this method when a form with AutoFilterManager is closed, and this leads to resource leaks. To help the developers to avoid this issue when instances of the **iGAutoFilterManager** class are created in the Windows Forms designer, the class implements a new overloaded constructor with the **IContainer** parameter:

```
public iGAutoFilterManager(IContainer container)
```

When a new instance of the **iGAutoFilterManager** 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.iGAutoFilterManager1 = new
    TenTec.Windows.iGridLib.Filtering.iGAutoFilterManager(this.components);
  …
}
```

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

We strongly recommend that you check your code to be sure you dispose the resources of all used **iGAutoFilterManager** objects. If you do not do this, the easiest ways to fix this problem for instances created in the Windows Forms designer is to delete them from the Designer's tray area and add them again from the Visual Studio toolbox. Note that you need to remember the non-default values of AutoFilterManager properties and set them again after doing this (if any).

An alternative way keeping non-default property values is to add **this.components** manually as an argument of all **iGAutoFilterManager** constructors in the code generated by the Windows Forms designer. To make it work, check also that the form's **components** collection is initialized in the **InitializeComponent()** method of the form like in the code snippet above.

v10.0.20 | 2020-Nov-27

Enhancements in the core grid control

1. [Enhancement] This build of iGrid brings enhancements to the algorithm that converts the entered string into the cell value. One of the improvements makes it easier for the user to edit formatted numeric values.

   For example, the developer can specify a format string like "{0:0,0.00}" or "{0:C}" for currency amounts to display them with additional formatting characters like thousand separators. In the case of the format string "{0:C}" the decimal value of 5000 will be displayed as "$5,000.00" on a computer with the regional format set to "English (United States)". The user can edit one character in this string – for example, change it to "$5,001.00" – and save this new value. In the previous builds of iGrid the user would get the message "Input string was not in a correct format" at that, but the current build properly processes the new string and saves 5001 as the cell value. Moreover, the user can enter values with incorrect positions of thousand separators, and they will be accepted by iGrid too. For example, strings like "$5,00.00" or "$5,0000.00" will be saved as 500 and 50000 respectively without any error messages.

   A brief version of the enhanced algorithm that converts entered strings looks like this:

1) If the entered string is empty, convert it to the corresponding empty value depending on the value of the **EmptyStringAs** property if it is defined or save the string "as is" otherwise. Exit.
2) Determine the value type to convert to. This is the type defined in the **ValueType** property or if it is not defined, the type of the current cell value if it is not null (Nothing in VB) or **DBNull**.
3) If it was not possible to determine the value type on Step 2, try to parse the entered string as a number. The **Double**, **Int32**, **Int64**, **Decimal** types are tested for this (in this order). If the conversion is not possible, save the entered string "as is" in the cell value. Exit.
4) If the value type to convert to is
    a. numeric (all integer types, **Decimal**, **Single** and **Double**), convert the entered string to it by calling the **Parse()** method of the corresponding type;
    b. an enumeration, call the **Enum.Parse()** method to get the enumeration item from the entered string;
    c. otherwise try to convert the entered string to the cell value using the **Convert.ChangeType()** method for the value type to convert to.

The parts of this algorithm are wrapped with the try..catch blocks that intercept possible conversion errors. If an exception occurs, it is displayed in the message box like in the previous versions.

2. [New] iGrid implements the new public method **DrawFooterCellContents** used by the internal infrastructure to print the contents of footer section.
3. [Enhancement] The vertical grid line separating the fake row header from footer cells in the footer section is always drawn even if drawing of vertical grid lines were turned off in the **iGrid.GridLines.Mode** property. This helps to see the bounds of this footer row header area regardless the visibility of the vertical grid lines the same way as we see row headers in the row header area when it is visible.
4. [Enhancement] The special last row horizontal grid line has a higher drawing priority than the special last column vertical grid line regardless of the value of the **iGrid.GridLines.ZOrder** property. This enhancement provides a better look of iGrid because the normal cell vertical grid lines never break this special last row horizontal grid line. This enhancement also provides a look consistent with the special header and footer separating lines that always have a higher drawing priority than the normal vertical grid lines in the corresponding grid areas.
5. [Fixed] The normal horizontal grid lines in the footer cells broke the special vertical grid line drawn after the last footer column.
6. [Fixed] The **CustomDrawFooterRowHdr** event was raised even when the footer was not visible.
7. [Fixed] The header may contain extra unneeded empty space after calling the **Header.AutoHeight()** method.
8. [Fixed] The special vertical grid line after the last frozen column was broken by the horizontal grid lines in the footer section.
9. [Fixed] The **iGColHdr.SpanRoot** and **iGFooterCell.SpanRoot** properties returned incorrect values after reordering columns.
10. [Fixed] The Reset command in the context menu of the Visual Studio Property Grid set the **GroupBox.HintBackColor** and **GroupBox.HintForeColor** properties to incorrect default values.

Enhancements in the PrintManager add-on

1. [New] Printing of the grid footer section was implemented in this release. If the footer section is visible in iGrid on the screen, it is automatically printed by the PrintManager add-on.

PrintManager uses the settings for normal cells when printing footer cells, such as visibility and style of grid lines, their Z-order, the printing style (screen or plain look), and so on.

The footer section is framed by the separating line at the top and bottom and the special vertical grid line for the last column at the left and right to provide beautiful symmetric look on paper.

2. [New] The print-preview dialog supports scrolling with the mouse wheel.

   The mouse wheel rotation works exactly like in iGrid. If you do not hold the CTRL key while rotating the mouse wheel, the preview picture is scrolled vertically. If you hold the CTRL pressed, the preview picture is scrolled in the horizontal direction.

3. [New] Now PrintManager supports the new grid line priority system introduced in iGrid.NET X. The special grid lines framing the last column and row are not broken by normal cell grid lines on the paper. The same concerns the special grid lines drawn on edges of the frozen area. The value of the **iGrid.GridLines.ZOrder** property is also taken into account when drawing the grid on the paper.

4. [Enhancement] If printing of the row header area is allowed, it is printed only on the pages containing the first visible column of iGrid. This allows the user to understand whether a merged cell is broken by a vertical page break because the row header is absent at the left of a broken merged cell. One more benefit of this enhancement is that now reports printed on several pages can be combined into one whole report easily with sellotape.

5. [Enhancement] In the previous versions of PrintManager, the special grid lines of the last column and row set with the **iGrid.GridLines.VerticalLastCol** and **iGrid.GridLines.HorizontalLastRow** properties were printed after the last column and last row on every page. Now they are printed only after the last grid column and row exactly like the user sees these special grid lines in iGrid on the screen.

   In addition to that, the last column's grid line is printed before the very first column if the row header is not printed. Similarly, the last row's grid line is printed before the very first row if the column headers are not printed. This provides a beautiful symmetric look of iGrid on paper.

6. [Enhancement] The vertical level area for group rows is not broken by the horizontal grid line of the last row on a page. This helps the user to understand that there are more rows belonging to group rows.

7. [Fixed] If iGrid had group rows, the horizontal grid line in the last row was not drawn under the level areas of group rows.

8. [Fixed] Custom page margins set with the **Document.DefaultPageSettings.Margins** property were not processed properly in right-to-left mode.

9. [Fixed] The special grid lines of the last row set with the **iGrid.GridLines.HorizontalLastRow** property was not printed – the normal horizontal cell grid line was printed instead.

10. [Fixed] PrintManager raised the **CustomDrawPageHeaderGetBounds** and **CustomDrawPageFooterGetBounds** events instead of **CustomDrawDocumentHeaderGetBounds** and **CustomDrawDocumentFooterGetBounds** events while drawing the document header and footer respectively.

11. [Fixed] The **ColorizeRowLevelIndent** setting of iGrid was not processed by PrintManager properly in some cases.

12. [Fixed] The contents of merged cells were not printed in right-to-left mode.

13. [Fixed] Minor problems with drawing styled row and column headers were fixed.