10Tec iGrid for .NET 4.x
What's New in the Control

Keywords used to classify changes:

- [New] – a totally new feature;
- [Change] – a change in a member functionality or interactive behavior;
- [Improved] – something is implemented better than in previous versions;
- [Renaming] – a name of the member was changed so it is enough to rename it in your code;
- [Removed] – a feature was removed;
- [Fixed] – a fixed bug or solved problem.

v4.60, build 0020 | 2014-Dec-24

1. [Enhancements] The sorting algorithm was optimized. The new version allows the user to sort columns with primitive data types like Integer or String up to 2.5 times faster.
2. [Change] The **ShowControlsInAllCells** property no longer affects the column headers. In the previous builds, the column header combo buttons were hidden if this property was set to False, and there was no way to access them.
3. [Fixed] The two-headed arrow cursor displayed in the row header area and indicating the ability to resize the row was flickering while moving the mouse.
4. [Fixed] The current cell may have disappeared after some keyboard commands that change the current cell (TAB, RIGHT ARROW, CTRL+END, etc).
5. [Fixed] This build of iGrid fixes some minor problems with releasing resources used to draw the iGrid contents.

v4.60, build 0012 | 2014-Sep-26

1. [Enhancement] The iGrid header supports autosrolling now. If a column header is moved outside of the iGrid scrollable area, the grid is automatically scrolled in the corresponding horizontal direction to show the column headers hidden at the left or right. This allows users to reorder columns in wide grids much easier than earlier.

   For column headers, iGrid uses the same adaptive autoscrolling behavior like for its cells: the far the mouse pointer from the scrollable area, the greater the speed of scrolling.
2. [Fixed] Column headers were drawn incorrectly when AutoFilterManager was attached to iGrid while it had frozen columns.
3. [Fixed] The **CellMouseDown**/**CellMouseUp** events were not raised in some cases (for instance, when a context menu was activated for iGrid).
4. [Fixed] The built-in incremental search did not work for row text cells.
5. [Fixed] If the ESCAPE key was pressed while the search-as-type window was visible, the form was closed automatically if it had the cancel button used to close this form (the form's **CancelButton** property was set).

v4.60, build 0005 | 2014-May-16

1. [New] Added support for the standard touch gestures available on touch screen in Windows 7, 8 and 8.1. The current version of iGrid.NET allows you to use the pan gesture with inertia to scroll the grid contents, and the press-and-hold gesture as an equivalent of right-click.

2. [Fixed] If the row text cells were not visible, the current cell selection disappeared when the user pressed the TAB key in the cell of the last visible column (the same for SHIFT+TAB in the first visible column).
3. [Fixed] The current cell selection disappeared while using the TAB/SHIFT+TAB key combinations in group rows.
4. [Fixed] The ENTER key used to insert line breaks in multiline text edit mode did not work.
5. [Fixed] The **GetStartEndCells** method failed if iGrid had no rows and/or columns.

v4.60, build 0000 | 2014-Jan-17

1. [New] The new custom cell editor model based on the new **iGCellEditorBase** class was implemented. To implement a custom cell editor, inherit this class and implement its two mandatory members – the **EditControl** property and the **SetBounds** method. The former member should return an instance of the control you use as the cell custom editor; the latter method is called by iGrid when a cell is put in edit mode so you can position the custom edit control in the cell properly.

   To specify a custom cell editor for a cell or set of cells, assign a reference to it to the new **CustomEditor** property of the cell object (**iGCell**) or cell style object (**iGCellStyle**), for example:

```
iGrid1.Cols[0].CellStyle.CustomEditor = new NumericUpDownCellEditor();
```

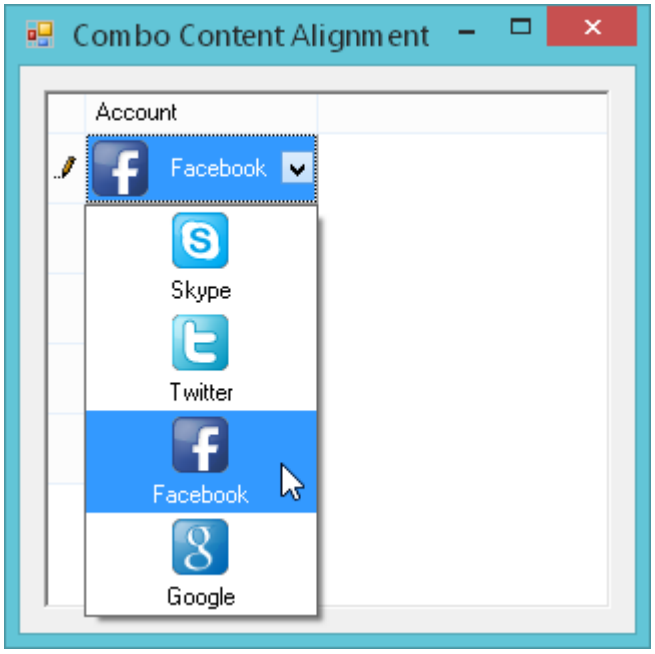
   The **CustomEditor** property has the **iGCellEditorBase** type and accepts any objects based on the **iGCellEditorBase** class. In our example, the NumericUpDownCellEditor class that represents a custom editor based on the standard WinForms NumericUpDown control may look like this:

```
internal class NumericUpDownCellEditor : iGCellEditorBase
{
   private NumericUpDown fNumericUpDown = new NumericUpDown() {
      BorderStyle = BorderStyle.None, Maximum = 10000, Increment = 5};

   public override Control EditControl
   {
      get
      {
         return fNumericUpDown;
      }
   }

   public override void SetBounds(
      Rectangle textBounds, Rectangle suggestedBounds)
   {
      fNumericUpDown.Bounds = textBounds;
   }
}
```

   The main benefit of the new custom editor system is that it simplifies your code a lot if we compare it to the approach used in the previous versions (based on the **RequestEdit**/**QuitCustomEdit** events). All you need to do is to tell iGrid what control is used as the edit control and position it properly taking into account the control resize capabilities – all other things (setting font and foreground/background colors, processing special control keys like TAB and ENTER, etc) will be done automatically.

   Note that you can define just one object that implements custom editing and use it in several columns of the same grid, or in several grids on the form or in the whole project. All custom editing functionality is concentrated in one class now, and there is no longer need to copy it from one form to another if you wish to use the same custom editor in different forms of your app!

2. [New] The built-in iGrid drop-down list class has 3 new properties you can use to set the image and text alignment for its items. These are the same properties you can use for iGrid cells: **TextAlign**, **ImageAlign** and **TextPosToImage**. With these new properties, you can use the same layout for the drop-down list items which is used in the cell the drop-down list is attached to, or use another one which differs from it:



The default values for these properties are **iGContentAlignment.MiddleLeft**, **iGContentAlignment.MiddleLeft** and **iGTextPosToImage.Horizontally**. These values correspond to the layout used in the most real-world cases. These are also the settings used in the previous versions of iGrid internally, so the new default settings provide backward behavior compatibility without the need to change existing code.

The **NotSet** values can be also used as the values of these properties. In this case, the corresponding layout settings are inherited from the cell and are applied to the drop-down list on the fly when it is opened.

3. [New] The iGrid drop-down list item collection class (**iGDropDownListItemCollection**) was enhanced by two new methods, **FindByValue** and **FindByText**, you can use to search an item by item value or item text. Both methods are accessible through the **iGDropDownList.Items** property and return the found drop-down list item (**iGDropDownListItem**) or null if there is no matching item. Here is code example of how to find the list item with the value "car" and display its text:

```
iGDropDownListItem myFoundItem = iGDropDownList1.Items.FindByValue("car");
if (myFoundItem == null)
   MessageBox.Show("Not found!");
else
   MessageBox.Show("Item text: " + myFoundItem.Text);
```

The **FindByValue** method has the following syntax:

```
public iGDropDownListItem FindByValue(object value)
```

To search the list, it uses a fast search algorithm based on the internal sorted list of values. In the case when the list has several items with the same value, the method may return not the first item in the list because of that.

The **FindByText** method has the following syntax:

```
public iGDropDownListItem FindByText(string text, bool ignoreCase)
```

It searches for the item by enumerating the list items from top to bottom, so it guarantees that the first matching item is returned. The **ignoreCase** parameter is used to perform case-sensitive or case-insensitive search.

4. [New] This release introduces the new cell and cell style property **MaxInputLength**. It gets/sets the maximum number of characters that can be entered into a cell. It is an Int32 property with the default value of zero, which indicates that the limit is not set.

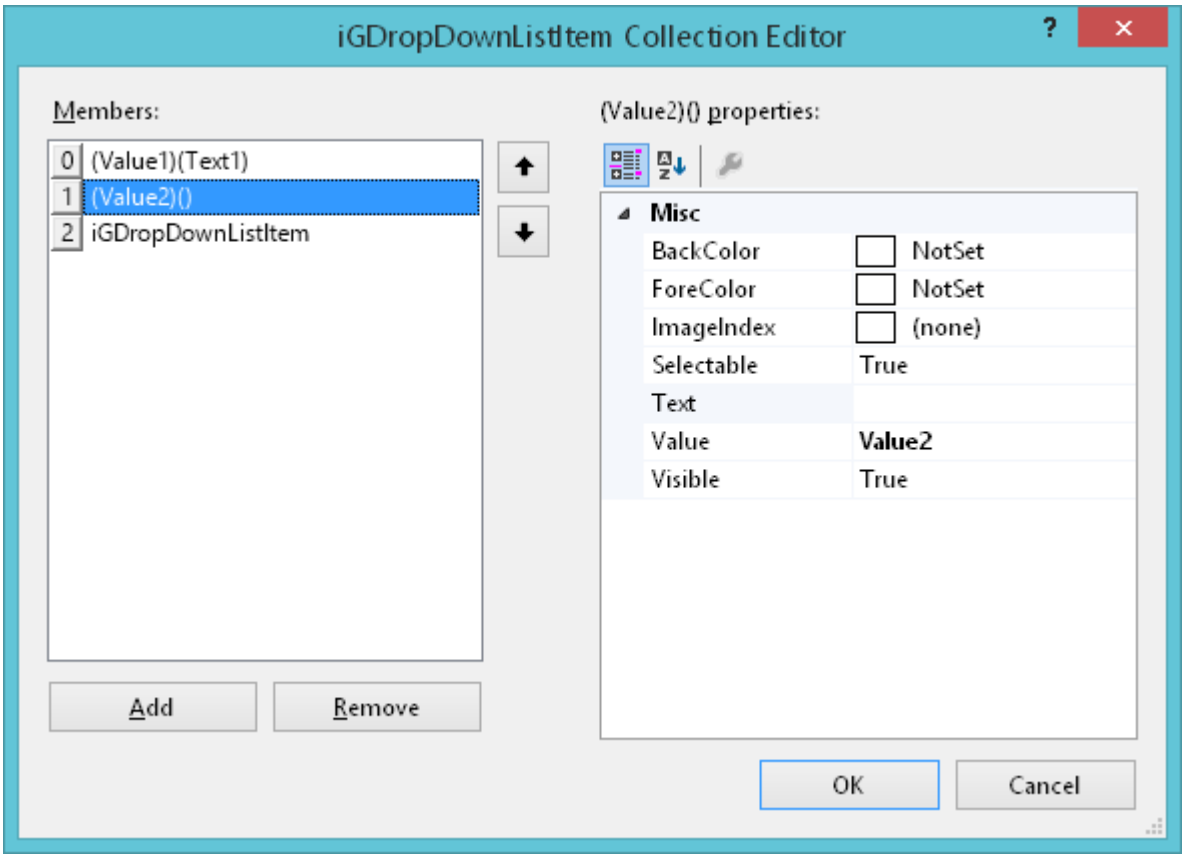
5. [New] Two new properties, **EnsureVisibleAutoHScroll** and **EnsureVisibleAutoVScroll**, were implemented. These properties specify whether to scroll the grid automatically in the horizontal or vertical direction respectively to display the current cell without clipping. The default value for both properties is True.

   If you select a cell which is partially visible or completely outside of the viewport of iGrid, by default iGrid scrolls its contents automatically to display this cell in its viewport. Using these two properties, you can prevent iGrid from automatic horizontal and/or vertical scrolling in this case. To do that, set the corresponding property to False.

   These properties affect both the interactive and programming cell selection (using the **EnsureVisible** methods of the iGrid cell, column or row objects). Note that even if you set these properties to False, you can still scroll the grid manually using its scroll bars.

6. [New] The **iGCell** class that represents a cell was supplemented with the three new properties: **EffectiveTextAlign**, **EffectiveImageAlign** and **EffectiveTextPosToImage**. They return the effective values for the **TextAlign**, **ImageAlign** and **TextPosToImage** properties respectively.

7. [Enhancement] The collection editor of drop-down list items displays the value and text for every item if they have been specified:



Every item is represented using the template "(value)(text)". The class name (**iGDropDownListItem**) is displayed if both properties **Value** and **Text** are not specified.

8. [Enhancement] The **iGrid.SearchAsType.SearchText** property returns the effective search text entered by the user if you access it from the **SearchAsTypeRowSetChanged** event (in the previous versions, the search text without the currently typed character was returned in this case).

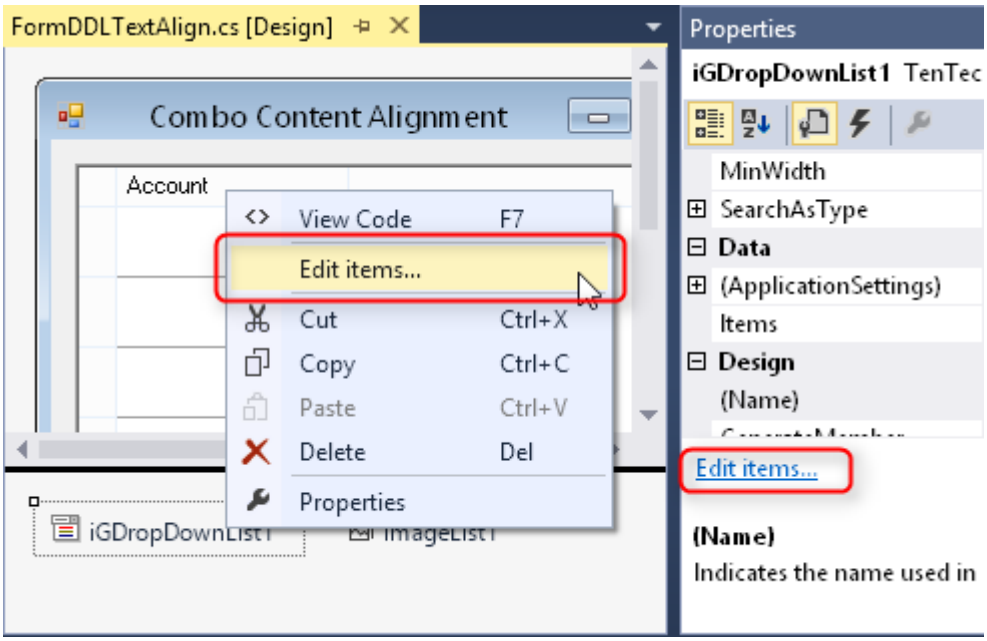
9. [Enhancement] The **iGRowPattern** class contains the new **Expanded** property that maps to the corresponding property of the row object (**iGRow**). The enhanced internal algorithms used to maintain the internal row hierarchy structure allows you to specify this setting for newly created rows now.

10. [Enhancement] If the grid is set to the read-only mode using the **ReadOnly** property, the cell combo buttons and cell ellipsis buttons become hidden. The same concerns individual cells with the **ReadOnly** property set to True. This setting does not affect the combo buttons in the column headers.

11. [Enhancement] In the previous versions, the cell text editor always used the left-top alignment regardless of the effective horizontal and vertical alignment of the cell text. This caused the cell text to jump to the left-top corner of the cell when the user started editing of a cell which used non-default horizontal or vertical alignment.

    Now the text editor always inherits the cell text's horizontal alignment, and if it makes sense, its vertical alignment too. The vertical alignment is inherited if the cell text is displayed and edited as one line text, i.e. neither the word-wrap display flag nor multiline edit flag are used. If the cell text isn't one line text, the editor uses the top alignment and occupies the whole cell allowing you to enter and see as many text lines as the cell height allows.

12. [Enhancement] The **iGDropDownList** component was provided with the ability to quickly open its item collection editor at design-time with the new "Edit items…" command in its context menu, smart tag and verb in the property grid:



13. [Enhancement] The **ToString()** method of the **iGColAutoFilter** object which represents the current filter criteria for a column now returns the text representation of the filter the users see in the tooltip for the drop-down autofilter button in the column's header.

14. [Enhancement] The **Cols.AutoWidth** method now affects all hidden columns. This can be useful in some scenarios – for instance, when you have columns that can be interactively shown/hidden by the user, and you want to adjust the width of all columns beforehand with one **Cols.AutoWidth** call.

15. [Fixed] A serious memory leak problem was fixed in the **AutoFilterManager** add-on: **AutoFilterManager**'s internal objects associated with columns that had been removed from the grid remained in memory and were not cleared by the **Garbage Collector** automatically. The problem

appeared when columns are removed from the **Cols** collection explicitly using its methods or implicitly during such calls as **FillWithData**.

The problem was mainly related to the known peculiarity of the standard **ToolStrip** control used in the interface of the **AutoFilterManager** dialog box. Briefly, the **ToolStrip**'s **Dispose** method should be called manually to release all associated resources, and this fix works only starting from the .NET Framework 3.5.1.

16. [Fixed] The **Grid** property of **AutoFilterManager** could not be changed from one instance of iGrid to another – only from null (Nothing in VB) to an iGrid instance, or vice versa to null.
17. [Fixed] The combo buttons in the header disappeared when the **ShowControlsInAllCells** property was set to False.
18. [Fixed] The **TextBoxTextChanged** event was not triggered when the user started to edit a cell.
19. [Fixed] The "Object reference not set to an instance of an object" error occurred at design time when you hit the Add button in the collection editor of drop-down list items if the list had not been populated yet or after the **Value** property of an item had been cleared.
20. [Fixed] When a new iGrid column is created at design time, its **CellStyle** object property is initialized automatically to allow the developer to set the properties of the column cells. After closing the column collection editor, this property could be cleared because of a bug in iGrid's design-time support code. This release fixes this problem and the similar issue with the column **ColHdrStyle** object.
21. [Fixed] If the user pressed a mouse button when the mouse pointer was over the empty area of iGrid and moved the mouse pointer over a column header holding down the mouse button, the column header movement started. There could be other glitches with drawing iGrid items in this case.
22. [Fixed] If the user pressed a mouse button when the mouse pointer was over the empty area of iGrid and moved the mouse pointer over a cell holding down the mouse button, the **CellMouseEnter** event was triggered. This did not conform to the general mouse event infrastructure, when the **MouseEnter** event is raised if the mouse input is not captured by another item.
23. [Fixed] A **System.IndexOutOfRangeException** could be generated if the grid had been grouped and/or sorted by some columns while removing columns from grid (using, for instance, the **Cols.Clear** method or when the column set is totally cleared during the **FillWithData** call).
24. [Fixed] The **SelectInvisibleCells** property did not work for invisible rows.

v4.50, build 0022 | 2013-Sep-24

Design-time functionality

1. [Improved] Earlier versions of iGrid.NET provided the developers with the ability to resize columns and scroll the grid at design time, but with the release of iGrid.NET 4.5 this functionality was broken because of some changes in the internal grid infrastructure. This build of iGrid restores that useful design time functionality. To enable it for a particular grid, select the grid on the form first.

   In all older versions of iGrid, the cursor was set to the default cursor or special column resize cursor depending on the iGrid item under it, but there was one annoying problem. The cursor was constantly changed back to the standard 4-header arrow used by the Windows Forms Designer after a short period of time so it looked like flickering. The corresponding part of the control designer was rewritten so this problem no longer occurs.

2. [New] The ability to resize rows at design time using row dividers was added.
3. [Improved] The **Cell[]** properties for a row object in the row collection editor were sorted improperly if the grid had more than 10 columns as the VS property grid sorts properties alphabetically (Cell[0], Cell[1], Cell[10], Cell[11], and so on). This build fixes this problem by padding the cell indices with '0'

at the left dynamically depending on the number of columns in the grid (Cell[00], Cell[01], …, Cell[09], Cell[10], Cell[11]).

For better readability in the property grid with its default font, square brackets were replaced with parentheses: Cell(0), Cell(1), …

The design time row property **(RowTextCell)** which represented the row text cell was renamed to **Cell(RowText)**. Due to this change, it is listed in a more logical order after the normal **Cell()** properties and it is placed directly below the **Cell()** properties if the developer removed sort by categories in the property grid.

4. [Fixed] Some parts of iGrid scroll bars (such as arrow buttons) might remain highlighted after the mouse pointer had been moved outside of the scroll bar control.
5. [Fixed] If you cleared the text in the **DefaultCellValue** property of a column, it was set to an empty string in code generated by the WinForms designer instead of clearing the assignment statement at all as it is done for all other **Value** properties (which means that the default null value is used).
6. [Fixed] If you changed the Name property of the component using the Name filed in the smart tag panel, not all references to the component in the source code were renamed and compilation errors occurred after that.
7. [Fixed] The cleanup code of the control designer was rewritten to avoid any potential resource leak problems.

Run-time functionality

1. [Fixed][Change] A series of problems related to the **ColWidthStartChange**/**ColWidthEndChange** events have been fixed:
   a. The **ColWidthStartChange** event was always raised if the user pressed the left mouse button when the mouse pointer was over a column divider, even if the user decided to cancel column resizing by releasing the mouse button without moving the cursor (the behavior was similar to the **MouseDown** event). The logic implemented in this build of iGrid is smarter – the **ColWidthStartChange** event is raised only when real column resizing has been started, i.e. when the user started to drag a column divider.
   b. If the user did not change a column's width at all, or dragged the column's divider and set the same column width, the **ColWidthEndChange** event was not raised though the **ColWidthStartChange** event was triggered. In the new implementation, if real column resizing has been initiated and the accompanying **ColWidthEndChange** event was raised, the **ColWidthEndChange** event is always raised when the user releases the mouse button to finish column resizing.
   c. If the user double-clicked a column divider to resize the column, the **ColWidthStartChange** event was raised twice. Now it is raised once. Moreover, if it was always raised before the **ColDividerDoubleClick** event in which the developer could cancel the column auto-width operation, now **ColWidthStartChange** is raised after **ColDividerDoubleClick** and only if the auto-width operation has not been cancelled.

   To summarize, the **ColWidthStartChange**/**ColWidthEndChange** events are now paired – if **ColWidthStartChange** was raised, **ColWidthEndChange** will be also raised. And **ColWidthStartChange** is raised only once when real column resizing has been started.

   The same changes were made for the events related to interactive row resizing (**RowHeightStartChange**/**RowHeightEndChange** and **RowDividerDoubleClick**).
2. [Fixed] If a cell or column header contained a very long text (1000-2000, or more characters), and the user placed the mouse pointer over the cell to see its full contents in the tooltip, the whole app with iGrid may freeze for about a minute.

In fact, this problem is caused by the native Windows OS tooltip functionality used in iGrid to have the OS look-and-feel. The problem occurs when visual styles are turned on and it is in the system library USP10.dll which handles Unicode layout of characters on screen. This issue exists in Windows Vista or later versions and has not been fixed yet by the moment of this component update (for more info, see http://social.msdn.microsoft.com/Forums/windowsdesktop/en-US/a5a95763-9f10-4640-9d32-afd050a60bb3/tooltip-rendering-extremely-slow-on-vista-due-to-wordwrap).

To fix this problem, the cell text displayed in the built-in iGrid tooltip is automatically cut off if it exceeds 300 characters. To indicate this, the so called horizontal ellipsis character "…" is added to the end of the tooltip text (the hexadecimal Unicode code is 2026).

In fact, the problem depends on many factors - the horizontal screen resolution, whether the cell text has enough spaces to wrap it, etc. But the limit of 300 characters is an empiric value which should guarantee that the user will never face this problem on all modern monitors and devices like tablets. In any case, you as a developer can always redefine this behavior and show the original cell text "as is" or modify it as required using the **RequestCellToolTipText** or **RequestColHdrToolTipText** events.

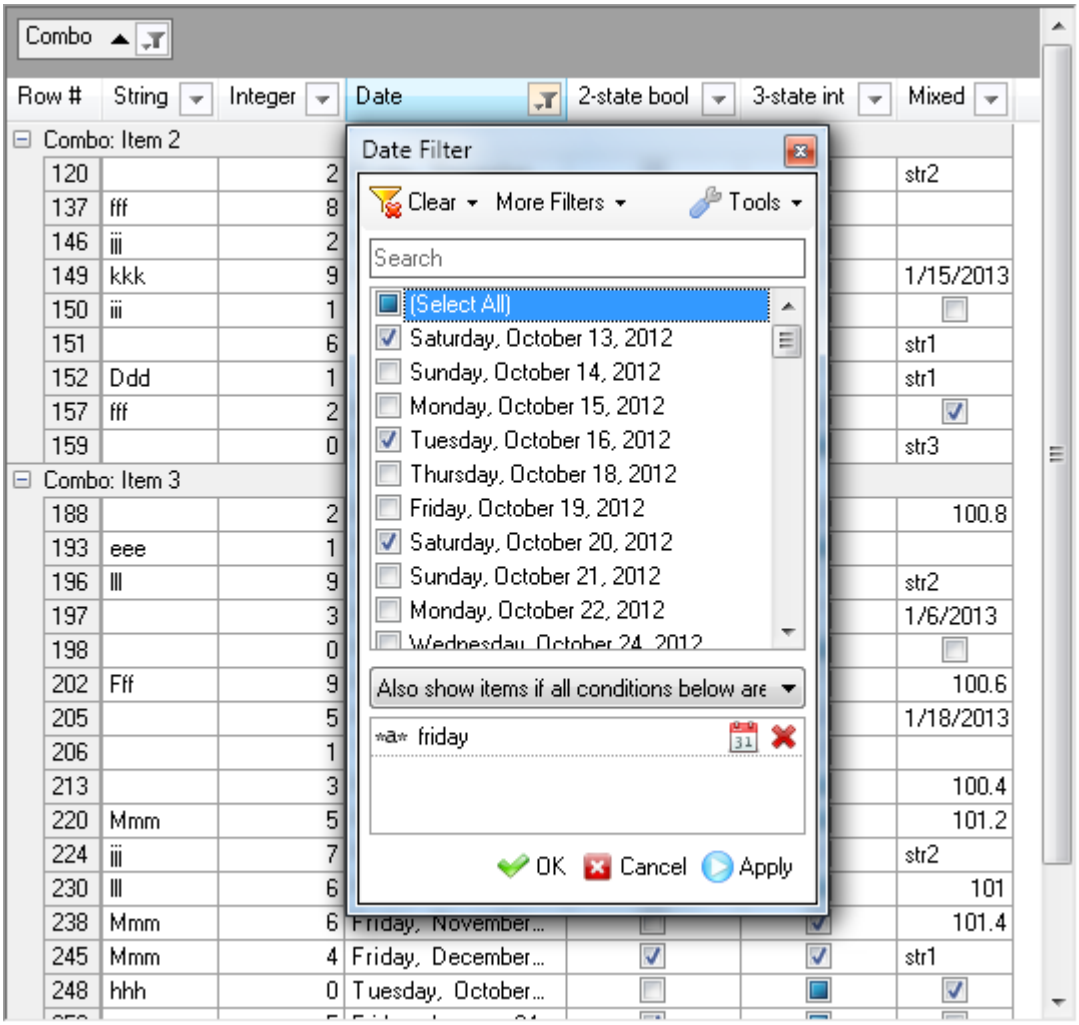
This fix isn't used for other iGrid items which can display tooltips, such as custom scroll bar buttons or cell controls (check box, combo button, etc). All these tooltips are specified explicitly by the developer, and in the vast majority of cases they are not so long so the problem may occur.

3. [Fixed] The **DoubleClick**/**MouseDoubleClick** events were not triggered after displaying a built-in drop-down list or custom drop-down control from an iGrid cell or column header. This was also the case for the AutoFilterManager dialog which is based on the same drop-down infrastructure (the **IiGDropDownControl** interface).
4. [Fixed] In some cases, the whole form with iGrid could not be closed (using the close button or even from code) after a cell drop-down list had been displayed.
5. [Fixed] An iGrid drop-down list used as a cell auto-complete list might stop working after this list had been opened using the cell combo button.
6. [Fixed] When iGrid was hosted on a non-WinForms form (for instance, on an MS Office VBA UserForm using COM Interop), an exception of the System.Exception type was generated when a drop-down control was opened from iGrid.
7. [Fixed] A potential problem in the iGrid drop-down control functionality which might cause an exception on 64-bit platforms was fixed.
8. [Fixed] If the grid was setup to display the current cell in row mode (the **RowModeHasCurCell** property was set to True), it was impossible to select the row text cell using the TAB or arrow keys.
9. [Fixed] In right-to-left mode, iGrid didn't draw the sort icon in a column header if the middle or right horizontal alignment was used for its contents.
10. [Fixed] The Boolean function **IsMouseCaptured** did not return True if the mouse was pressed over an iGrid scroll bar.
11. [Fixed] The **IsPointOverHeader** function did not take into account the visibility of the vertical scroll bar. It might also return incorrect result in right-to-left mode.

v4.50, build 0000 | 2013-Feb-20

New autofilter functionality

The most significant new feature of this release of iGrid is the autofilter functionality. It is implemented in the form of an add-in named AutoFilterManager, similar to the existing PrintManager. When AutoFilterManager is attached to an iGrid.NET, the column headers contain special filter buttons which are used to open the autofilter dialog to specify filter criteria for the columns:



To find out more about the iGrid autofilter functionality, read the documentation and examine the supplied demo sample.

Drop-down controls infrastructure

1. [New] The **IiGDropDownControl** interface was supplemented with the new **OnShowing** method. It is called by iGrid when the drop-down control is about to be displayed on the screen (in contrast to the **OnShow** method which is raised when the control has been displayed).

2. [New] Two new properties, Boolean **CloseButton** and String **Text**, were added to the **IiGDropDownControl** interface. If you set one of them for a sizeable drop-down control, a title bar with the corresponding contents is displayed at the top of the control as if it were a regular window.

3. [New] One more new property, Boolean **HideColHdrDropDown**, was implemented in the **IiGDropDownControl** interface. iGrid requests the value of this property when it is about to close the drop-down control displayed from a column header (while performing grouping, etc). Set this property to False to prevent the control from closing in such cases.

4. [Removed] The almost unused **MinWidth** and **MaxWidth** properties were removed from the **IiGDropDownControl** interface. The logic provided by these properties can be easily implemented in the implementation of the **Width** property of this interface.

5. [Fixed] In some cases iGrid might unpredictably change the saved size of a custom drop-down control built with the **IiGDropDownControl** interface after another similar drop-down control had been displayed. This problem is fixed in this release, and now iGrid correctly saves and restores the individual size of every custom sizeable drop-down control.

6. [Fixed] The width of the sizeable drop-down control based on the **IiGDropDownControl** interface was increased by 20 – 30 pixels after another non-sizeable drop-down control had been opened.

7. [Fixed] You could see the unneeded bottom-right corner of the shadow rectangle near the rounded bottom-right corner of a sizeable drop-down control in Vista and Windows 7 when the Aero theme was in effect.

8. [Fixed] iGrid's drop-down controls might have drawing problems because of the system implementation of the fade animation effect used on opening (especially in Vista and Windows 7). The opening animation effect is no longer used.

9. [Fixed] The button of the application with iGrid was drawn inactive as if the application had lost the input focus (came to background) when opening any drop-down control.

General grid functionality

1. [New] The **FillWithData** method has the new overloaded version that allows you to specify the start row and row count to upload to the grid. These are the last two parameters, **dataStartRow** and **dataRowCount**, in the method with the following signature:

```
public void FillWithData(object dataSource, bool useCurColSet,
  string rowKeyCol, bool addRowKeyCol,
  string rowLevelCol, bool addRowLevelCol, bool addTreeButtons,
  int dataStartRow, int dataRowCount)
```

   This can be useful in many situations. For instance, when you need just to preview some top rows from the data source, or you have a grid that displays rows by pages using a Next Page command.

   The data source's row indexing starts with zero, and to retrieve the very first logical row, use zero as the value of the **dataStartRow** parameter.

   The number of rows to fetch in **dataRowCount** cannot be negative, and the zero value is also possible. In this case no rows are retrieved, and **FillWithData** can just create the column set that corresponds to the specified data source (this can be useful to view its structure in the grid). If the **dataRowCount** parameter is greater than the number of the available rows, the method simply uploads all rows to the end of the data source.

2. [New] The **FillWithData** method generates the new **FillWithDataRowAdded** event as soon as it has created and populated a new row in the grid. The parameters of this event provide access to the data source row used to populate the grid row this event is raised for. For instance, if you need to store the value of the field "Country" in row tags while populating grid rows using **FillWithData**, use code like this:

```
private void iGrid1_FillWithDataRowAdded(
  object sender, iGFillWithDataRowAddedEventArgs e)
{
  iGrid1.Rows[e.RowIndex].Tag = e.DataRow["Patient"];
```

```
}
```

The event data argument has the following type:

```
public class iGFillWithDataRowAddedEventArgs : EventArgs
{
  public readonly int RowIndex;
  public readonly DataRow DataRow;
  public readonly IDataReader DataReader;
}
```

The **RowIndex** property returns the index of the grid row the event is raised for.

To access the corresponding row of the data source, use either the **DataRow** or **DataReader** property. **DataRow** is used if you specified a **DataTable** or **DataView** as the data source in the **FillWithData** method, **DataReader** is used if the data source is an **IDataReader** or **IDbCommand** object. In the latter case the **DataReader** object is a whole DataReader used to read the data source. It is positioned on the row used to populate the corresponding grid row, and you can use all its methods – even ones which give you the best performance (the type-specific methods like **OleDbDataReader.GetInt32()**). However, don't do any other things which could close the **DataReader** or change its current row as it will affect the proper functionality of the **FillWithData** method.

3. [New][Change] The **ColsAdded** and **ColsRemoved** events were implemented. Like the **RowsAdded** and **RowsRemoved** event, they occur when iGrid columns have been created/destroyed.

   To notify the developer that some columns/rows are about to be removed from the grid, two more new events, **ColsRemoving** and **RowsRemoving**, were implemented.

   The **RowsAdded** and **RowsRemoved** events had the **iGRowsAddedEventHandler** and **iGRowsRemovedEventHandler** types with the arguments **iGRowsAddedEventArgs** and **iGRowsRemovedEventArgs** in the previous versions. In this release of iGrid these events and the new **RowsRemoving** event are unified and have the same new type **iGRowSetChangeEventHandler**. Its unified **iGRowSetChangeEventArgs** event arguments provide the same fields, **RowIndex** and **RowCount**, for all these events.

   The **ColsAdded**, **ColsRemoving** and **ColsRemoved** events also have the new unified **iGColSetChangeEventHandler** type. Its event arguments, **iGColSetChangeEventArgs**, give you the information about the columns which are created or removed (the **ColIndex** and **ColCount** fields).

4. [New] The **iGrid.CurColHdr** property was added. It returns an **iGColHdr** object that represents the column header which is currently active (highlighted). In contrast to the current cell and the corresponding **CurCell** property, a column header is active if the mouse pointer is over it or a drop-down control attached to this column header is opened.

   The **iGrid.CurColHdr** property is read-only and cannot be assigned to change the current active column header (the other distinction from the similar **CurCell** property).

5. [Fixed] iGrid crashed when the left mouse button was hold down in the grid area which does not contain cells, and then the pointer moved over a column divider in the header.

v4.00, build 0007 | 2012-Oct-25

1. [Improved][Fixed] In multi-threaded apps some interactive actions, such as row resizing or column moving, cannot be performed in the general case as another thread can update the data and related internal structures in the grid. As a part of this, previous versions of iGrid did not allow column moving and resizing in multi-threaded apps when rows are added/removed (a crash might occur), but this version was improved to allow this.

2. [Fixed] The **iGColHdr.DropDownControl** property returned an empty reference or another improper object even if a proper object had been assigned to it.

3. [Fixed] If a column header had a drop-down control, the value selected in it (its **AuxValue** property) was used as the prefix in group rows instead of the column header text.

4. [Fixed] If a form with the demo version of iGrid was closed, an unhandled exception of accessing a disposed object might occur.